Universidade Nova de Lisboa

Faculdade de Ciências e Tecnologia

Departamento de Informática

Integrating Early Aspects with Goal-Oriented Requirements Engineering: The Case of KAOS

Por

André Tiago Varejão Gil

Dissertação apresentada na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa para obtenção do grau de Mestre em Engenharia Informática

Orientador

Prof. Doutor João Baptista da Silva Araújo Junior

Lisboa

2008

Acknowledgements

Personally I like when I am praised for something, and I think there are some people deserving to be praised for helping me completing this milestone in my life.

Firstly a very special thank you to Ana who gives a meaning to my life and makes it worth to get up in the morning. To her another special thanks for cope with the hours stolen to our common life.

Then to my parents and to my family who had always supported me.

A special thanks to Professor Doutor Engenheiro João Araújo Junior, for tutor me in this thesis and for always being helpful and always trying to improve my work in the pursue of my Msc.

Without naming, friends from FCK (they know who they are) and all the other friends (they also know who they are) a special thanks for hours spent on your company.

A special thanks to Raquel and Pedro, you know why.

Thank you all, I will never forget all the support you gave me.

Resumo

A engenharia de requisitos consiste em elicitar, analisar, especificar, validar e gerir os requisitos de um sistema. Quando se pretendem elicitar os requisitos de um sistema, existem várias abordagens possíveis, entre elas temos a orientada a objectivos e a orientada a aspectos. Contudo estas abordagens raramente são utilizadas em conjunto.

Quando se utilizam abordagens orientadas a objectivos, um problema muito comum é o facto de existirem objectivos que se repetem nos respectivos modelos dificultando a sua evolução. Uma redução desta complexidade poderá ser obtida através da utilização de uma aproximação orientada a aspectos.

A grande vantagem de se utilizar uma aproximação híbrida é a possibilidade de fazer com que o mesmo objectivo, que se encontra espalhado pelo modelo, possa ser modularizado num aspecto. É assim possível simplificar o modelo, tornando a sua percepção mais fácil para quem tem de o consultar ou analisar numa fase posterior. O resultado final é a possibilidade de uma evolução mais controlada, com menos riscos de erros.

Esta dissertação apresentará uma abordagem que permitirá a modelação do sistema através de objectivos e aspectos. Para isto será utilizada a metodologia KAOS estendida através de mecanismos inerentes ao desenvolvimento orientado a aspectos.

Abstract

Requirements engineering aims at eliciting, analyzing, specifying, validating and managing system requirements. When eliciting system requirements, it is possible to use various approaches, including goal-oriented and aspect-oriented approaches. Although those are two well-known approaches, they are seldom used in conjunction.

On the other hand, when using goal-oriented approaches, one common and usual problem is the fact that some of the goals repeat themselves all over the system.

This makes goal-oriented models to have a boost in complexity because of the repeating goals, and thus, making the evolution of this model harder than necessary. This complexity could be minimized if an aspect-oriented approach would be used. The big advantage of using a hybrid approach, in our case goal-oriented and aspect-oriented one is the possibility to identify all the scattered goals and modularize them as aspects. In this way we can represent this kind of goal (now an aspect) only once in the model. This means the complexity of the model will be greatly reduced and the readability of the model will also be improved. The final result will be an evolution that could be easily controlled, thus minimizing errors.

Although this seems a good idea, there are some challenges to overcome when merging goals and aspects. First of all, a notation and a set of rules must be built in order to compose the model. In order to do this we will use patterns based on roles, as these will help elaborating the model.

This work will present an approach that will make possible after modeling the system with a goal-oriented approach, identify aspects and then refine the model taking into account the aspects. In order to accomplish this, the KAOS methodology will be extended with aspects.

Acronyms

AoGRL - Aspect-Oriented Goal Requirements Language

AORE - Aspects-Oriented Requirements Engineering

AORA - Aspect-Oriented Requirements Analysis

AOSD - Aspects-Oriented Software Development

GBRAM - Goal-Based Requirements Analysis Method

GORE - Goal-Oriented Requirements Engineering

GRL - Goal Oriented Requirements Language

IPS - Interaction Pattern Specification

KAOS - Knowledge Acquisition in autOmated Specification

NF – Non-Functional

NFR - Non-Functional Requirements

PSs - Pattern Specifications

RE - Requirements Engineering

SD - Strategic Dependency

SR - Strategic Rationale

UML - Unified Modeling Language

Table of Contents

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

This chapter presents an overview of what requirements engineering is and what are the current problems with goal-oriented approaches. These problems provide the motivation for the current work. Then, the objectives and contributions are delineated. Finally, in section 1.4, the organization of the document is given.

1.1 REQUIREMENTS ENGINEERING

In the context of a software system, the stakeholders are the ones with specific interests in the system, be them the ones who sponsor the system or who will use the system to do their job, or even who will maintain the system.

This poses a number of inherent difficulties to this process. The stakeholders of the system may be numerous and distributed. Their own expectations may vary and collide with each other. Expectations can collide due to the interpretation stakeholders have of the environment where they work and the tasks they wish to accomplish.

To make the task harder, their own expectations may not be explicit, or may be difficult to explain. This poses a serious challenge to the analyst, as the satisfaction of the expectations that are not explicitly stated may be constrained by a variety of factors that the analyst does not control. The Requirements Engineering (RE) discipline addresses the early stages of software development and tries to guarantee the stakeholders' expectations will be satisfied. Therefore, the context in which requirements engineering take place is usually a human activity system, and the problem owners are people. Requirements engineering needs to take into account how people see the world and how the system will interact with the world.

One definition for Requirements Engineering says that it is "the process of establishing the services (functional requirements) that the customer requires from a system and the constraints (non-functional requirements) under which it operates and is developed" [36]. This highlights the main kinds of requirements involved in RE.

Another good definition can be found in [46] as being: "Requirements engineering is the branch of software engineering concerned with the real world goals for, function of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families."

This definition provides some clues on what requirements engineering really is. It is about "meeting real world goals" and it also provides the rationale of it. Requirements engineering has as main activities, elicitation, analysis, specification, validation/verification and management.

The first activity relates to the understanding of the organizational situation the system will target and hopefully improve. It must also describe the needs and constraints the system will overcome and face. Analysis and specification has to do with real world needs mapping into a requirements model. The validation task tries to ensure that the derived specification corresponds to the original needs. Management deals with requirements traceability and requirements evolution.

There are several requirements engineering approaches such as viewpoint-oriented [3], use case driven [51], problem frames [49], aspect-oriented [34] and goal oriented [6] ones. A viewpoint is

a local object containing only partial knowledge about the system and domain. Here we will focus on goal-oriented and aspect-oriented requirements engineering.

Goal-oriented requirements engineering is an approach to requirements engineering whose focus is on identifying and defining systems goals. When doing requirements engineering, goals are present, as they usually identify what the requirements for the system are, and what the system must accomplish. A definition of what a goal is can be found in [5] as being a "prescriptive statement of intent about some system whose satisfaction in general requires the cooperation of some of the agents forming that system".

When using goals to analyze a system, the advantage is the refinement of a goal into sub-goals, making the model incrementally clearer to stakeholders. Also, when doing requirements engineering, the approaches based on goals, link the needs to both functional and non-functional concerns. Handling non-functional requirements using a goal-oriented approach has the advantage of eliciting and specifying earlier and then facilitating the later phases of software development. When it comes to validation, goals can be validated by identifying or generating scenarios that are covered by them.

In [6] it is possible to realize the importance of goals when the author writes "Goals have long been recognized to be essential components involved in the requirements engineering (RE) process." However, goal-oriented models present some problems, namely the crosscutting concerns problem described next.

1.2 MOTIVATION

When doing requirements engineering a usual problem is modularizing concerns and especially cross-cutting concerns. Concerns are properties that the system must take into account like functional and non-functional requirements.

A cross-cutting concern is a concern that affects other concerns. This have a negative impact on software as these types of concerns are hard to separate from the rest of the system, which makes them tangled or scattered all over the system specification. Cross-cutting concerns are hard to

deal with as their scope is the whole system, and as such, modifying them might have a high impact on other concerns. Here concerns can be mapped to goals.

Current approaches that deal with goals, for example KAOS [8], GBRAM [1], i* [27] or NFR [38] do not address the problem of cross-cutting concerns. Besides not identifying and modularizing cross-cutting concerns explicitly, they also do not cover concern composition in a more systematic way, which specifies how a concern affects other concerns.

With the evidence that cross-cutting concerns always happen in most of systems and that, in general, the current goal oriented approach do not provide a systematic way to deal with them, the present work will try to handle these gaps.

Dealing with cross-cutting concerns early in the software development phase is important as it will not leave them intact until implementation phase, where any change will affect and impact the maintenance and evolution process to a greater degree than if it were done at the requirements level.

Finally when addressing cross-cutting concerns in goal models, they are easier to understand by the development team and stakeholders.

1.3 OBJECTIVES AND CONTRIBUTIONS OF THE WORK

The objective of this work will be to create an approach to model systems through the use of goals and aspects. Aspects will be used to further simplify the model, so stakeholders will understand better when looking at the model and modeling the system in such a way that requirements evolution will be facilitated since aspect-oriented provides efficient modularization mechanisms.

When the analyst is gathering the requirements of the system, the work at hand, should at least, accomplish all the objectives stated by stakeholders, and those objectives should be presented to them in a clear way.

Also the system to be developed must have its requirements organized in such way that any change can be realized in a short period of time and keeping the requirements document consistent. For what has been written till now, an approach based only on goals will sooner or later have to deal with goals that crosscut other goals, i.e. it will have to deal with goals that are tangled or scattered all over the model.

Therefore the contribution of this approach is to enhance the results produced by a goal oriented requirements approach, namely KAOS, by introducing a more efficient modularization mechanism, the so called aspect. The aimed target is a more evolvable and understandable system requirement.

1.4 ORGANIZATION OF THE DOCUMENT

Besides this chapter, this document is organized as follows:

- Chapter 2 – In this chapter main GORE concepts are introduced, the KAOS approach is described in detail. Then we look at other GORE approaches such as i*, NFR Framework, GBRAM and GRL;
- Chapter 3 – It presents main AOSD concepts. We look at AORE approaches such as Scenario Modeling with Aspects, Patterns Specifications, ARCADE, ThemeDoc, Use Cases and Aspects, and AORA. We also discuss approaches integrating aspects into GORE. We present an overview of VGraph, Aspects and i* and finally AoGRL;
- Chapter 4 – The approach we developed is presented. Our approach, AspectKAOS aims at using KAOS and aspects. This chapter provides guidance on how to apply our approach;
- Chapter 5 – A case study is introduced and discussed. AspectKAOS is applied to a real system that has been previously modeled with UML. This chapter also presents a comparison between the developed approach and some widely used approaches is given based on some criteria we find relevant to each approach, and providing a side-by-side comparison for each criteria;
- Chapter 6 – Conclusions of the developed work are presented.

CHAPTER 2

GORE (GOAL-ORIENTED REQUIREMENTS ENGINEERING)

In the first sub-sections GORE and KAOS are introduced. Afterwards, other approaches are described and finally we summarize the chapter.

2.1 GORE (GOAL-ORIENTED REQUIREMENT ENGINEERING)

Goals are objectives. Objectives are what the system must accomplish. These goals can go from actions to generic achievements. One example of generic achievement would be “the system must be safe”. Of course a goal like this cannot have an agent responsible for it. A requirement represents one particular way of achieving some specific goal.

Goals can be both functional and non-functional. Functional goals are bound to actions that the system must accomplish like the goal “taking passengers from floor a to floor b”.

Non-functional goals relate to generic achievements, like security, safety and so on. The goal “the system must be safe” actually is a non-functional concern. Non-functional goals are usually

cross-cutting and as the model gets closer to the final stage, these goals will become an objective throughout the model.

When formulating goals, one can use different levels of abstraction, or even formulate them at different levels of abstraction. This means that it is possible to write a high-level or low-level goal. When formulating a high-level goal it would be something like “take more passengers to the desired floor”, for an elevator system, and when formulating it in lower-level it would resemble something like “stopping command timely delivered” for the same elevator system.

With goals having an important role to play in requirements engineering, we can summarize their importance as follows:

- First of all, with goals it is possible to verify the completeness of a requirements specification. The specification is complete when all the goals can be achieved from the specification.
- Second, the specification is used to prove a goal. In this way it is possible to validate all the requirements.
- With goals identified, if we need to talk to stakeholders and show them the current modeling of the system, they will look at it and understand it quite easily as the learning curve is not big.
- When refining goals, they provide a natural mechanism for structuring complex requirements. Goal refinement means goals are further divided into sub-goals. When refining goals, new kinds of goals will be identified.

Some types of goals are satisfaction goals, information goals, accuracy goals, performance goals, security goals and so on [18, 2, 38, 14]. Goals are classified in those types because each type has certain specific characteristics or deal with specific type of information or concerns. They are described below:

- Satisfaction goals are functional goals with the objective of satisfying agent requests.
- Information goals are also functional goals but they are responsible for maintaining agents informed of object states.
- Accuracy goals are non-functional goals, ensuring that the state of software objects correctly represents the state of the real hardware being monitored.

- Performance goals deal with the robustness, response time and throughput of the system.
- Security goals are responsible for things like confidentiality, integrity and availability.

Of course, when using goals, every goal will apply to someone or something. This means it should be possible to validate them. It is possible to validate goals through the use of scenarios (behavioral descriptions of a system and its environment arising from restricted situations).

These scenarios must be generated or identified and be comprehensive so that they will be covered by the goals we are trying to validate.

In the next section KAOS will be described. KAOS is the goal oriented approach that will be used in this work.

2.2 KAOS

KAOS [8] stands for Knowledge Acquisition in autOmated Specification [2]. KAOS was developed with four goals in mind [4]:

- Fitting problem description by defining and manipulating concepts relevant to problem description.
- Improve problem analysis process by providing a systematic approach for discovering and structuring requirements.
- Clarify the responsibilities of all the project stakeholders.
- Let the stakeholders communicate easily and efficiently about the requirements.

When using KAOS we can both use a top-down or bottom-up analysis. Actually the advised strategy is to use both bottom-up and top-down at the same time. This is useful as it will allow the analyst to reach a point where some goals are modeled from the detail and others are modeled from general goal deep to the detail.

As goals are responsibilities of agents, this means that if a goal has no agent responsible for it, the goal should and must be refined until an agent is found. Of course a goal is refined into various sub-goals.

This is the reason why both bottom-up and top-down analysis should be used. When an agent is known and the responsibility for an objective can be attributed to that agent, the bottom-up approach is the way to go, as usually it is easier to model and get it right the first time.

When the agent responsible for the goal is not known, it is advised to model top-down as it will make it clear along the way who would be candidate to be responsible for the goal. At the very end of the path, only one responsible agent will subsist, and that agent will become responsible for the given goal and sub-goals.

As requirements engineers live in a world where conflicts are the rule and, not the exception [9], conflicts generally arise from multiple viewpoints and concerns, as such, they will appear sooner or later and must be dealt with.

Conflicts should be detected as soon as possible as it will help building a more robust system and to increase the probability of getting it right at first shot.

In KAOS the notation to represent goals is different than the other approaches. Goals are represented with specific graphical representations (Figure 2.1).

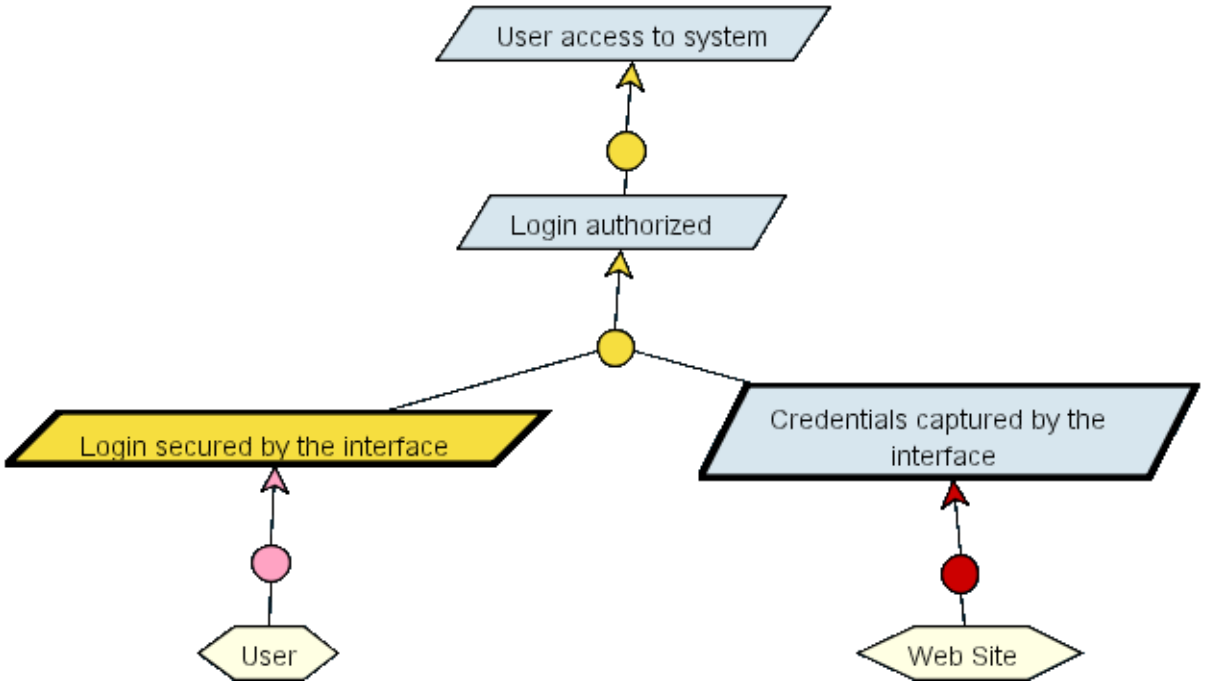


Figure 2.1 – Basic representation of KAOS goals

The previous example has some graphical representations of entities in KAOS. Goals are refined into sub-goals until an agent is responsible for it. Then they become requirements or expectations. The "yellow" goal is an expectation, and means that something is expected by the user (the agent). This means the system should always try to meet user requests and expectations.

Sub-goals, expectations and requirements are always connected to higher goals through the use of yellow circles. Expectations are connected to agents (as they represent the expectation of an agent) with pink circles.

In KAOS it is also possible to set responsibilities to agents. This is done with the use of red circles connecting the agent and the desired goal, which is considered now a requirement. "Credentials captured by the interface" is a requirement because a piece of software has the responsibility to guarantee that the goal will be achieved.

KAOS models obstacles as well. Obstacles are represented by a parallelogram with different orientation of a goal (Figure 2.2). Obstacles are represented in pink color, connected with an arrow to the goal the obstacle applies to.

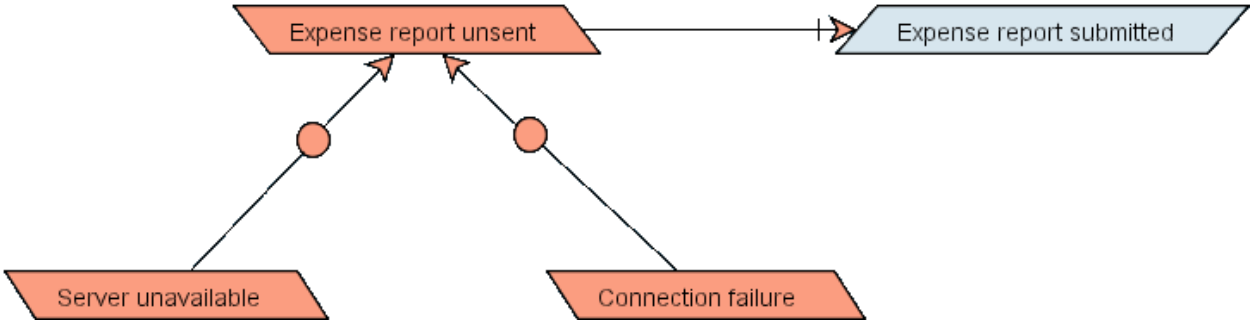


Figure 2.2 – What obstacles are and how to represent them

KAOS also has three more models, object model, operations model and responsibility model.

2.2.1 Object Model

To document and define the concepts of the domain that composes the environment of the application, we use an object model. The object model has both objects belonging to the domain

of stakeholders. Other objects are just introduced to express requirements or constraints of the system.

In this model, we should have three types of objects.

- Entities, representing independent, passive objects. Being independent and passive means descriptions do not need to incorporate or reference other objects and they cannot perform actions on their own;
- Agents, which are independent and active objects. The fact that they are active, implies that they can release operations on their own;
- Associations, which are dependent and passive objects. They are dependent because they refer to other objects.

In order to identify objects we must recur to the goal definition process. The objects found during inspection of goal model, should be related to existing concepts, or new ones if stakeholders have not yet looked at the object model.

Another approach to identify objects is looking at the requirements and trying to satisfy requirements with system components.

KAOS object model is compliant with UML class diagrams as KAOS entities correspond to UML classes and KAOS associations correspond to association links or association classes.

2.2.2 Operation Model

The objective of "Operation Model" in KAOS is to describe all the behaviors the various agents must accomplish in order to cover their requirements. These behaviors are expressed by operations agents can perform.

Operations can be identified in two ways.

- During interviews with stakeholders, as they usually describe processes not goals, it is responsibility of the analyst to ask specific questions in order to identify operations;
- Trying to explain the current requirements is another good way to identify operations as operations describe how the requirements should be realized.

It is possible for a requirement to be operationalized by objects or/and behaviors.

- Requirements describing static properties are operationalized by objects;
- Requirements describing dynamic properties are operationalized by operations;
- Requirements describing both static and dynamic properties are operationalized by objects and operations.

Operations can be triggered by events. If no event is specified, the operation will be triggered when all the data needed by the input is present.

Operations are used to constrain the solutions that can be used to design the system, thus making the requirements document specific.

2.2.3 Responsibility Model

This model has the relation between agents and requirements or expectations for which those agents are responsible.

In KAOS goals can be assigned to many agents or be the responsibility of only one agent.

- Assignment is used when various agents may be made responsible for some requirement or expectation;
- Responsibility is used when only one agent is responsible for it.

This makes it easier to refine goals, and to know when to stop. When the goal is responsibility of an agent, then it is the right time to stop refining it.

Responsibilities diagram can be derived from the goal model, thus each diagram will show all the requirements or expectations an agent is responsible for. Although, we have these three types of models, in the current work we will use only the goal model.

2.3 Other Goal-Oriented RE Approaches

2.3.1 i*

i* [27] is an agent and goal-oriented modeling framework that can be used for requirements engineering, business process reengineering, organizational impact analysis and software process modeling.

This framework can be used both at the early or late phases of requirements gathering. In the requirements engineering process, when applied to the early phase the framework will be responsible for modeling the environment of the new system. It makes it easy to analyze the domain, as it allows representing the stakeholders of the system, their objectives, and their relationships. The analyst can, therefore, visualize the current processes in the organization and examine the rationale behind them. On the other hand, at later stages, the models help to ensure the system is robust enough so that, with new system configurations and new processes, the system meets the needs and evaluate to a good classification in meeting both functional and non-functional requirements of the users.

The central notions of i* are intentional actor and intentional dependency [24, 21]. Actors are intentional if they have motivations, intents, and rationales behind their actions. On the other hand, a dependency is intentional if it results from agents trying to achieve their goals. Actors are described by their organizational settings and have attributes as goals, abilities, beliefs and commitments.

An actor depends on other actors. Only with this dependency can it achieve the completion status of its goals, the execution of tasks, and the supply of resources. When an agent is trying to achieve its goals and creates a dependency to another agent, then that dependency is marked as intentional.

This means that each actor should and can use various opportunities to achieve more at less cost, as they use other agents too. At the same time, the actor is vulnerable if the actors it depends upon are not working or doing their job as they should.

As actors are central to this framework they are seen as strategic, as they are concerned with the achievement of their objectives and could represent both the stakeholders or the agents of the new system.

Roles, agents and positions can all be actors. Agents are real actors, systems or human beings with specific capabilities. A role is an "abstract actor embodying expectations and responsibilities" [43]. When an agent plays a set of socially recognized roles; that is called a position. This classification of roles and positions is especially useful when analyzing the social context for software systems.

In i*, besides intentional dependency, we can find three more types of dependencies. Those dependencies are [22, 52, 25, 23]:

- Goal, which means that in order for the goal to be achieved it depends on other goals of the system. "The meeting initiator's dependency on participant's attendance at the meeting is a goal dependency";
- Soft-goal, requiring that some goal would be met if other aspects of the system are also met. "For important participants, the meeting initiator depends critically on their attendance, and thus also on their assurance that they will attend";
- Task, if in order to complete something (an activity for example); it depends on some other component of the system to accomplish something too. "Physicians depend on Claim Managers to Get Approval of Treatment";
- Resource, when the availability of something is dependent on another part of the system. "… the general manager's dependency on the customer for payment, the tester's dependency on the programmer for the code,…".

The i* framework has two main models: the Strategic Dependency (SD) model and the Strategic Rationale (SR) model. Dependency relationships can be learned from looking at the SD model. The Strategic Dependency model is a network of dependency relationships among actors.

The intentionality of the processes in the organization can be captured with the use of the SD model. With the SD gathering together the information about the processes in the organization, it

will be possible to know what is important to its participants, while at the same time it would be possible to abstract over all the details.

Strategic Rationale models help to understand the motivation behind the processes in systems and organizations. To describe the motivation behind the processes in the SR model we will find model elements such as goals, soft-goals, tasks and resources. Obviously the relations between these elements will also figure in the model.

SD and SR models complement each other. While the first is used to explicit the external relationships among actors, the second is used to analyze in great depth the internal processes within each actor.

Figure 2.3 presents an SD model. This is a small subset of the SD model presented in [24], which describes a system to manage meetings.

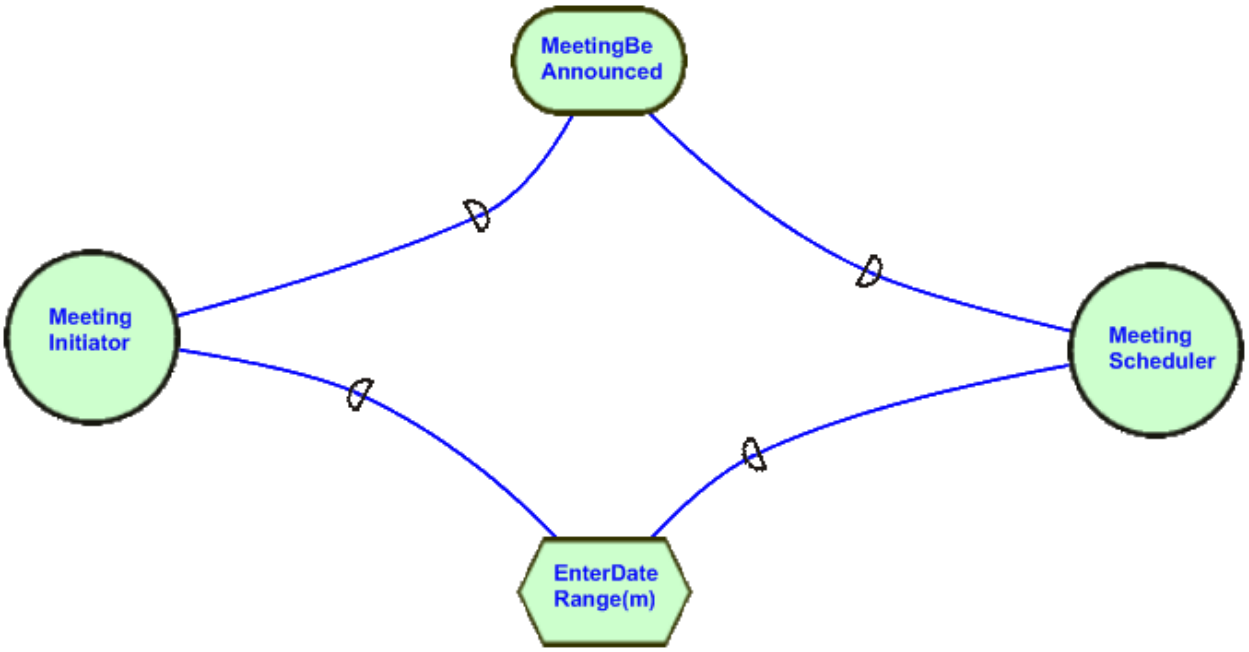


Figure 2.3 – Strategic Dependency Model

In this SD model, the Meeting Initiator receives from the Meeting Scheduler the date ranges available for meetings and then, announces the meeting to the proposed date.

This means SR models will help the analyst to assess possible alternatives in the definition of the processes to better address the concerns of the actors as the model allows for a deeper understanding of what each actor's needs are, and how these needs are or should be met.

SR elements are related by links, modeling AND and OR decompositions of the process. The links modeling AND are known as decomposition links while the ones modeling OR are known as means-ends. As means-ends links model OR relations, they apply mostly to goals and provide a means to figure alternative ways to achieve those same goals. On the other hand, decomposition links connect goals and tasks with its components (subtasks, soft-goals, etc). This is understandable as decomposition links model AND relations.

Although we have these two kinds of links there are a third kind that connects soft-goals, goals or tasks to soft-goals. This links are known as contribution links.

Contribution links provide two levels of both positive and negative types of contributions to the soft-goals. This means contributions to the soft-goals go from satisfying a soft-goal to executing a task and passing by achieving a goal.

In this way, soft-goals are used so it will be easier to choose the alternative process that meets non-functional requirements. A SR model only includes elements considered important enough to affect the achievement of some goal. It basically refines the SD model. Figure 2.4 shows the SR model. This SR model is also a subset from [24].

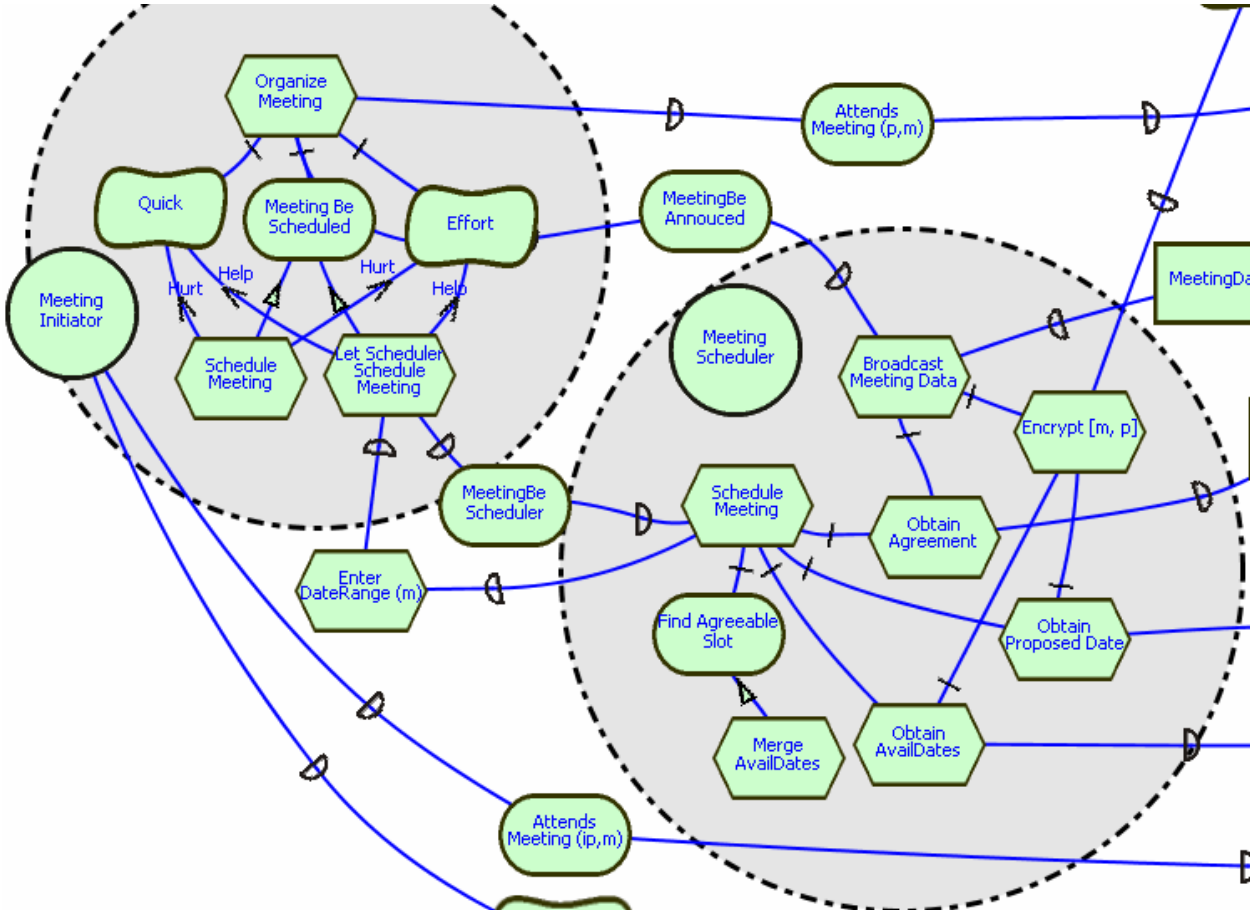


Figure 2.4 – Strategic Rationale Model

The Meeting Scheduler actor represents a software system that performs the meeting scheduling. On the other hand, the Meeting Initiator has the responsibility to provide or receive information to or from the system.

The task Schedule Meeting pertains to the Meeting Scheduler actor. The task Schedule Meeting is decomposed into three sub-components using task-decomposition relationship. The sub-components represent the work that will be accomplished by the scheduler system.

2.3.2 NFR Framework

NFR Framework was proposed in [38]. This framework concentrates on modeling and analysis of non-functional requirements. The objective of NFR [33] is to consider non-functional requirements from the very first beginning. NFR has four main activities:

- Capturing NFR for the domain of interest;
- Decomposing NFRs;

- Identifying possible NFR operationalizations.

This approach attempts to model and refine systematically non-functional requirements and to expose positive and negative influences of different alternatives on the requirements.

Soft-goals play an important role in the NFR Framework. Soft-goals represent the non-functional requirements that must be considered. Operationalizing soft-goals model lower-level techniques for satisfying a NFR soft-goal. Finally claim soft-goals enable the analyst to gather design rational for soft-goal refinements, soft-goal prioritizations and soft-goal contributions.

This framework provides the soft-goal interdependency graph or SIG. This represents soft-goals, soft-goal refinements (AND and OR), soft-goal contributions (positive and negative), soft-goal operationalizations and claims.

NFR supports catalogues that serve as a means to index design knowledge. There are three catalogues:

- Type catalogue which incorporates particular types of non-functional requirements, for example, performance.
- Method catalogue contains information that helps in soft-goal refinement and operationalization.
- Rule catalogue helps in detecting implicit dependencies and inter-dependencies between and among soft-goals.

Finally, this framework provides a process-oriented approach for dealing with non-functional requirements.

2.3.3 GBRAM (Goal-Based Requirements Analysis Method)

The Goal-Based Requirements Analysis Method [1, 50] aims at the identification and abstraction of goals from various sources of information. This method provides guidance in identifying and analyzing the goals that determine the requirements of the software system.

It assumes that the stakeholders had not provided goals, be them elicited or documented. This means it will be possible to use existing diagrams, textual statements and interview transcripts. GBRAM involves goal analysis and goal refinements as activities.

This method advocates the text analysis strategy and scenario analysis strategy. Text analysis strategy is based on the analysis of existing documents. The scenario analysis strategy is executed against the analysis of scenarios with common problems and where a goal may fail or be blocked, thus providing a way to solve those problems.

Analyzing goals is about exploring information sources in order to identify goals, followed by organization and classification of goals. The goals analysis task is divided into exploration and identification activities. The first activity has the objective to browse the available information. The second activity, identification, has to do with extraction of goals and finding the agents responsible for them from the available information.

The organization task aims at classifying and organizing goals according to the goal dependency relations. GBRAM also incorporates constraints. These must be met in order to achieve a given goal. While refining goals, this methodology requires that goal precedence be established in this phase. This means, while refining goals we will find which ones precede the others. In GBRAM each component (goals, agents, stakeholders) are textually specified in goal schemas.

2.3.4 GRL (Goal Oriented Requirements Language)

Goal-oriented Requirements Language is defined in [42] as “a language for supporting goal and agent oriented modeling and reasoning of requirements, especially for dealing with Non-functional Requirements (NFRs).”

This approach is important as it provides means to express wide range of concepts appearing during the requirement and high-level architectural design process. The concepts include intentional elements, links and actors.

The intentional elements are composed of goals, tasks, soft-goals and resources. The reason why they are called intentional has to do with the fact that they are used for models that answer

questions such as why such behaviors exist, what kind of alternatives were considered and what were the criterions used when choosing among alternatives, and what were the reasons. It is a similar approach to i*.

In this context, a goal represents a condition in the world that stakeholders would like to achieve. Usually how it is achieved is not specified so alternatives can be considered. Tasks specify particular ways of doing things. If a task is a sub-task of another task, then it restricts the task that is this tasks parent, so the parent can only take the course of action specified by the sub-task. Tasks are essentially solutions that will satisfy soft-goals or help achieving goals.

Soft-goals model conditions the actor would like to achieve. Although this is what the actor would like to achieve there is no obligation for the analyst or the developer to enable the achievement of the soft-goal. The resource is a physical or informational element, with availability being the principal concern.

Intentional links in GRL include means-ends, decomposition, contribution, correlation and dependency. They are similar to i* links. Means-ends links exists both in GRL and i*. In i* means-ends links model OR relationships between goals. As in i*, in GRL means-ends links also provides alternative ways to achieve goals. Decomposition links also exist both in i* and GRL and they model AND relationships. This means they describe what sub-elements are needed in order for specific tasks to be achieved. Contribution links also exist in i* and connect soft-goals, goals or tasks to soft-goals, having the same effect as contribution links in GRL.

2.3.5 Discussion

KAOS has formal support for goal refinement. In this way, KAOS introduces generic refinement patterns that are proven correct and complete. Such patterns are defined in [47]. These patterns also provide temporal logic, such as milestone-driven or case-driven refinement.

A recurrent problem is goal satisfaction. In [38] absolute goal satisfaction is presented.

Approaches like GBRAM, i*, NFR Framework model goal satisfaction with the help of contributions. One problem with those contributions is their qualitative nature. By being

qualitative they provide ambiguous ways for decision support. They are qualitative as they are positive or negative, and thus do not answer an important question like why a positive contribution is more important than another positive contribution.

KAOS provide formal analysis, meaning goals can be assigned to agents; generate alternative responsibility assignments of goals to agents, and to generate agent interfaces [20]. Still on the formal side of KAOS, in [10, 4] an approach is proposed to resolve conflicts among different views gathered during requirements analysis.

2.4 SUMMARY

In this chapter we looked at KAOS and other goal-oriented requirements engineering technique. KAOS enables us to use both bottom-up and top-down approaches to goal refinement. This means we can model some goal from the detail and other from general goal knowledge to the detail. KAOS has goal model, object model, operation model and responsibility model.

Other goal-oriented approaches include i*, NFR, GBRAM and GRL.

i* is an agent and goal-oriented modeling framework. The central notions in i* are actors and dependencies. NFR framework model and analysis non-functional requirements. The objective of NFR is to consider non-functional requirements from the very first beginning. GBRAM aims at the identification and abstraction of goals from various sources of information. This method provides guidance in identifying and analyzing the goals that determine the requirements of the software system. GRL is used to support goal and agent oriented modeling and reasoning of requirements. As the aiming of our work is to incorporate aspects into KAOS, the following chapter will look at aspects oriented architectures.

CHAPTER 3

AORE (ASPECTS ORIENTED REQUIREMENTS ENGINEERING)

In this chapter we introduce the main Aspect Oriented Software Development (AOSD) concepts. We will also look at some AORE approaches.

3.1 AOSD (ASPECTS ORIENTED SOFTWARE DEVELOPMENT)

The principle of separation of concerns [31] can actually be considered one of the key principles in software engineering.

This principle states that every problem involves different kinds of concerns, and as such, these concerns should be identified and separated into different matters. This will help with complexity, performance, robustness, adaptability, maintainability and reusability. This principle can be applied in various ways.

Separating the right concerns is hard and complicated as the set that must be dealt with is larger and there are a bigger variety of concerns. This is a problem with concerns.

Aspects have certain characteristics, of which the following two are the relevant ones:

- An aspect affects multiple and different components.
- An aspect goes from end-to-end on the software, so it is spread over different components.

Generally speaking, aspects usually represent non-functional concerns, but they can also be functional. This is due to the nature of non-functional concerns, as generally these type of concerns are the ones that cross-cuts other concerns.

With this in mind, two concerns cross-cut if they intersect each other [7], meaning that at least one of them could be seen as an aspect. A big draw-back of aspects is that they usually are covered during the late stages of software development cycle. But they are also found in requirements analysis, domain analysis, not only during design, implementation and testing. This draw-back made research efforts in the field of aspects to turn to other directions and originated early-aspects, described next.

3.2 AORE (ASPECTS ORIENTED REQUIREMENTS ENGINEERING)

Early aspects are intended to identify all the relevant aspects as early as possible in the development life-cycle. This means that it is possible to identify all the relevant aspects from the domain knowledge, so they are identified in the very beginning of software development. This work will propose an early aspect approach based on goals.

Identifying aspects is a somewhat cumbersome task. Aspects usually are identified using common sense and are detected later on the software development process, usually during development. Aspects can be classified/divided in three types [19].

- Early aspects, found during the requirements and architecture design phases.
- Intermediate aspects, during analysis and design phases.
- Late aspects, during implementation and testing.

Both intermediate and late aspects are usually referred as aspects. Intermediate aspects can turn to early aspects or late aspects, but usually they turn into late aspects.

Early aspects are responsible for identifying aspects as soon as possible in the development life-cycle. Early aspects made their way into research and then into the big scene of aspect oriented software development. They focus on identifying aspects as soon as possible, usually in domain analysis or requirements engineering.

Using the aspect oriented approach from the very beginning of the project, can help capture cross-cutting concerns sooner, thus enhancing overall product quality and even making the product costly effective.

Identifying aspects early in the development life-cycle is important because it will make possible to construct an architecture as stable as possible from the very beginning. As cross-cutting concerns (aspects) found late on the development life-cycle have a profound impact on the design of the whole system, a systematic approach should be used to identify these aspects as soon as possible.

Also the later an aspect is found, the higher the cost of the project. Early Aspects involves the requirements, architecture and domain analysis activities. Here we will focus on requirements.

Aspect Oriented Requirements Engineering (AORE) is based upon the principle of separation of concerns where special attention is given to the identification, modularization and composition of crosscutting concerns at requirements level.

The AORE objective is to provide a way to identify, modularize, represent and compose crosscutting requirements, be them functional or non-functional [13, 28].

Composition mechanisms help AORE provide a detailed specification about how an aspect at requirements level constraints or influences other requirements. Deep understanding of these kinds of relationships between aspectual and non-aspectual requirements also improves the perception of interactions, conflicts and relationships between them.

At requirements level an aspect is a concern that applies to different parts of the system, affecting multiple requirements at the same time. For example, one security requirement may affect the whole of an e-commerce system.

An AORE approach is characterized by the following characteristics:

- An effective means to identify cross-cutting concerns as soon as possible.
- Separate cross-cutting concerns.
- A way to compose both aspectual and non-aspectual concerns so it will be possible to understand the effect of aspects on other requirements.

Here a concern expresses a set of strong related requirements. When gathering requirements there will frequently be discovered cross-cutting requirements, as some requirements are by nature cross-cutting and influences other requirements

A summary of the main AORE approaches are described below.

3.3 Scenario Modeling with Aspects

This approach was introduced in [40] and defines a scenario as "a trace of an execution through an existing system or system under development."

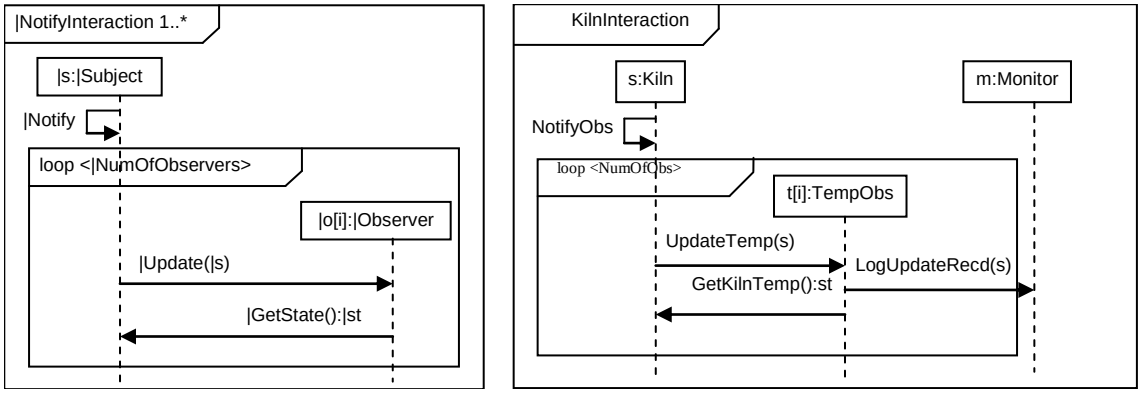
The stated approach starts with a set of requirements, gathered through all the means available to requirements engineers be them interviews, questionnaires, manuals and other documents. Thus the relevant information must be captured and organized. With these requirements it is possible to identify both system functionalities and non-functional concerns.

Then with the scenarios in place, they are converted into interaction diagrams and later on into a set of state machines. Some scenarios are aspectual that are separated from the non-aspectual ones. The aspectual scenarios [24] are modeled as patterns. Actually, the modeling of aspectual scenarios is made through interaction pattern specifications [48, 37].

3.4 PATTERN SPECIFICATIONS (PSS)

Pattern Specifications (PSs) are introduced in [48] to formalize the structural and behavioral properties of a pattern. The notation for PSs is based on the Unified Modeling Language (UML), but can be adapted to any modeling language. PSs specialize the UML metamodel [45] by describing what model elements must participate in the pattern. Each element in the PS is a role, that is, a UML metaclass specialized by additional properties that any element fulfilling the role must possess. Hence, a role specifies a subset of the instances of the UML metaclass. A PS can be instantiated by assigning UML model elements to the roles in the PS. A model conforms to a PS if its model elements that play the roles of the PS satisfy the properties defined by the roles.

PSs are used to specify pattern structure (Static Pattern Specifications), interactions (Interaction Pattern Specifications) and state-based behavior (State Machine Pattern Specifications). For example, an IPS defines a pattern of interaction between its participants. It has lifeline roles and message roles, which are specializations of the Lifeline and Message UML metaclasses. Each lifeline role is related to a classifier role, a specialization of a UML classifier. Figure 3.1 shows an IPS and a conforming sequence diagram (taken from [48]), for the Observer pattern.



(a) IPS for Observer Pattern (b) Conforming Sequence Diagram

Figure 3.1: An IPS and a Conforming Sequence Diagram.

Role names are preceded by a vertical bar to denote that they are roles. A conforming sequence diagram (see Figure 3.1(b)) must instantiate each of the roles with UML model elements satisfying any multiplicity and other constraints (e.g., given in the Object Constraint Language [39]). Note that any number of additional model elements may be present in a conforming sequence diagram as long as the role constraints are maintained.

In order, Figure 3.1, (b) to conform to (a) the instantiations below have to be made.

1. Bind |NotifyInteraction to KilnInteraction
2. Bind |s to s
3. Bind |Subject to Kiln
4. Bind |o[i] to t[i]
5. Bind |Observer to TempObs
6. Bind |Notify to NotifyObs
7. Bind |Update to UpdateTemp
8. Bind |GetState to GetKilnTemp
9. Bind |st to st
10. Bind |NumOfObservers to NumOfObs

Note that additional modeling elements are allowed in the resulting sequence diagram (namely, m, Monitor and LogUpdateRecd).

As aspectual scenarios are specified as IPSs, there must exist a binding statement for each non-aspectual scenario that they cross-cut. As it was said earlier these scenarios are transformed into a set of state machines. After this binding (or instantiation step) a composition operator is applied. This operator can be any of OR, AND and IN. The OR operator specifies alternative scenarios. The AND operator specifies that both scenarios should execute concurrently. Finally the IN operator specifies that one of the scenarios should be inserted into the other.

We will use this notion of PS in our approach. The other approaches, although important, do not provide a composition mechanism with the same strength as this one for diagrammatical models.

3.5 ARCADE

In [13] an approach on modularization and composition of cross-cutting and aspectual requirements is given.

The techniques used in [13] are viewpoints [29], in order to identify the requirements of stakeholders, and XML as the language to define the requirements, candidate aspects and the composition rules. The basis of this approach is the separation of the specification of aspectual requirements, non-aspectual requirements and composition rules following concrete templates.
Early separation of cross-cutting requirements helps at later stages as it simplifies the task of determination of their mappings.

After composing the candidate aspects and stakeholders requirements using the defined composition rules, identification and resolution of conflicts are carried out.

This step is accomplished by:

- Building a contribution matrix with positive or negative contributions from each aspect;
- Attributing weights to the previous aspects that have negative contributions;
- Solve conflicts with stakeholders, using the previous prioritizations.

The final activity in this approach is identification of the dimensions of an aspect.

3.6 THEMEDOC

ThemeDoc [19, 44] is based on the notion of a theme, which represents a feature of a system. Different themes can be combined in order to make the whole functioning better than the first.

- Base theme, which may share some structure and behavior;
- Cross-cutting themes, which have behavior that overlays the functionality of base themes;
- Cross-cutting themes are aspects.

ThemeDoc provides views that expose which behaviors are in requirements. These views assist the developer in determining what kind of relationship exists between behaviors and where those behaviors are based on aspects.

Action views do not have high and low level actions. In views, all actions have the same level. These views are used to determine whether actions should be themes, or just behavior inside themes.

In order to find which actions are not major enough to be a theme a very intuitive process is used. The actions are scanned and questioned whether it makes sense to have each of them as a feature in the system. If they are not worthy, then we will remove them from the model. The major action view is used in order to determine which themes are base, and which are aspects.

3.7 Use Cases and Aspects

Use cases represent functional concerns, as they model what a system should provide based on specific actions from agents or other components of the system.

In [35], Ivar Jacobson presents an approach on how to integrate use cases and aspects. Use cases decompose a system into parts, specifying its usage. This works well for use cases that are peers: none is more basic that the other. However some use cases depend on other, more basic, use cases to work. A way to separate these use cases is then presented in [35]. In order to modularize use cases by the way of aspects both a separation mechanism and a composition mechanism are needed.

The separation mechanism is based on slicing the current model. These slices are called use-case modules and cross-cuts the component modules.

On the other hand, in order for the system to work, it is needed to compose the slices, which means, joining the separated use cases into a consistent whole. As some use cases depend on other use cases, being these other use cases the one providing the basic features, an extension mechanism to separate them is used. This mechanism allows a basic use case to be built and then build other use cases on top of this one, only extending the functionalities on it, and never directly changing it. These types of extensions are called nonintrusive.

With this, the system should grow in an easy to understand design and code by structuring them from a base and let the system grow by itself.

3.8 AORA (ASPECT ORIENTED REQUIREMENTS ANALYSIS)

The AORA approach [32] defines three main tasks that can be further divided into sub-tasks. These sub-tasks can be more or less and have a limited or broad range. There are no limits to the number of sub-tasks a task can have.

The main tasks are:

- Identify concerns;
- Specify concerns; and
- Compose concerns.

In this context, concerns refer to what stakeholders see as problems to be solved. The task identify concerns consists of eliciting requirements. In order to specify concerns one must collect various types of information about each concern. The gathered information is then stored in a template. At last, at the composing concerns task, an incremental approach should be taken until the whole system is composed.

Composition rules are used as match points to help compose the concerns. Match points indicate which of the concerns should be composed together. These concerns can be both cross-cutting or not.

When conflicts are found, because of composition of different concerns, this approach advocates a simple process to solve them has to do with attributing different priorities to the concerns that conflict with each other. This is an intuitive process, depending only on who is attributing the priorities, as there is no guidance stated on how to give those same priorities.

3.9 GORE APPROACHES WITH ASPECTS

3.9.1 VGraph

This is a goal-oriented approach where crosscutting situations are identified. The name of the approach is based on the diagram used which is a graph with a shape of V representing respectively functional and non-functional requirements in terms of goal models.

Then non-functional requirements are represented in terms of soft-goals. The models incorporate AND or OR decomposition trees with the correlations links. The bottom vertice of the V represent tasks that contribute to the satisfaction of both goals and soft-goals.

This model provides a way to describe intentional nodes (goals or soft-goals) as well as operational ones. A topic gathers information for the goal/task/soft-goal. In VGraph model, topics also confine the point where the concern cross-cuts, between functional goals/tasks and non-functional soft-goals. To describe a generic function, a type for a goal or task is used. On the other hand, to describe a non-functional requirement, a type for a soft-goal is used.

The refinement of a VGraph ends when all the base (or root) goals and all soft-goals are satisfied.

At the end, VGraph enables any practitioner to treat aspects at a higher level of abstraction, in terms of goals, and do it in a systematic and formal way.

3.9.2 Aspects and i*

In [27] an approach to incorporate aspects into i* is given. Cross-cutting aspects are identified and kept in separate models. This greatly simplifies the original model.

There are four steps in the approach:

- Identification and representation of candidate aspects;
- Identification of the relationship among candidate aspects;
- Composition;
- Trade-offs analysis.

The first two steps are performed in parallel.

As cross-cutting concerns are elements required by other model elements. In order to detect cross-cutting concerns in the SD and SR models, three guidelines were defined.

In order to simplify the representation cross-cutting concerns are modularized and taken out of the original model. This means, there was the need to have a different kind of SR model. Thus, aspectual SR models were introduced.

In order to define “how” and “where” a cross-cutting concern affects other concerns, a set of composition rules were defined. This means that before composition, there is the need to identify which elements are related with the cross-cutting concerns. This identification of related elements is carried on by looking at the SR model. On the other hand by looking at the SD model, it is possible to find which actors depend on the operationalization of the previous concerns.

3.9.3 AoGRL

Aspect-Oriented Goal Requirements Language [30] adds aspect-oriented modeling to GRL. AoGRL is comprised of four basic components: joinpoint model, aspect, advice graph and pointcut graph, and composition rules. The joinpoint model comprises GRL nodes residing within the boundary of each actor. This means it is possible to identify any GRL node and also to add advice to any GRL node. With the joinpoint model, there were three concepts that were new to AoGRL: aspect, advice graph, and pointcut graph.

An aspect contains all goal graphs required to describe a concern. It contains:

- Advice graphs specifying cross-cutting goal graphs;
- Any number of pointcut graphs identifying the joinpoints where the cross-cutting occurs.

Pointcut and advice graphs can be reused independently as they are separated from each other.

Advice graph are similar to GRL catalogues where the catalogue describes the goal model of only one concern. AoGRL allows including GRL catalogues multiple times into GRL models.

In order to do this, we need to use the pointcut graphs, which describe pointcut expressions.

A pointcut expression is identified by pointcut markers. Wildcards and logical operators are used in order to parameterize the pointcut expression. Pointcut graphs contain both pointcut

expressions and other elements as elements that are not marked with a pointcut marker. The composition rule for the aspect is defined by connecting these previous elements with the elements of the pointcut expression.

3.9.4 Discussion

Approaches like i* with aspects, AoGRL and VGraph do not model obstacles. The modeling of obstacles can be achieved with a work-around, but it is not an explicit way as KAOS and consequently AspectKAOS support.

VGraph do support composition, but with such contrived rules that it becomes very hard to get it right without some experience, although the set of well defined rules do provide guidance and help.

In [27] a set of rules to composition is given. This introduces in i* the possibility to compose cross-cutting concerns back into the system. Composition is also supported by AoGRL. Those approaches are close to ours as they use in some way composition expressed in the model diagrams.

Modularization and modeling of cross-cutting concerns are provided by default, as all of the approaches (i* with aspects, AoGRL and VGraph) incorporate aspects thus taking all the advantages of the use of aspects.

3.10 SUMMARY

In this chapter we looked at some AORE approaches including GORE approaches with aspects. The next chapter will present our proposed solution AspectKAOS.

CHAPTER 4

MODELING ASPECTS IN KAOS

One current limitation of KAOS is the fact that cross-cutting concerns are not handled properly. The consequence is that they get scattered and tangled in goal models. If we incorporate aspect modeling mechanisms into KAOS, this will enable us to simplify the model as the crosscutting concerns will be kept separated. In our approach, called AspectKAOS, we take advantage of patterns and model some of the aspects as patterns. Then with patterns in use, we could make use of pattern matching to compose aspects back into the model.

An advantage of our approach is the simplification of the overall model. Also, it facilitates the evolution of goal models as it provides a better modularization of those models. Of course a basic understanding of KAOS will be needed but that is an essential requirement to apply our approach.

In this work we will use two different tools in order to build the models involved. Firstly, we will use “Respect-IT Objectiver” to model the base of our approach. Then we will use “Microsoft

Visio" in order to model the part of the notation that we introduce and that does not belong to KAOS.

4.1 ASPECTKAOS OVERVIEW

Our approach incorporates aspects into KAOS and provides a process to guide the developer. At the end of the process we will have our aspects fully integrated into our model.

The first step is to identify goals and requirements. In order to accomplish this step, we could and should use any of the usual techniques, e.g. interviews, current system analysis, ethnography. We will not discuss this in more detail as it is out of the scope of this work: our main objective is to address the cross-cutting goals.

Next we need to refine the goals that were identified in the previous step. Eventually, the refinement of goals will uncover some new goals which need to be refined too. This could happen, as goals can be modeled either top-down or bottom-up. This is realized the same way KAOS does.

The following step will be to identify aspects. To identify aspects we must do two activities in parallel. Identify and reuse patterns, and as such these basic building blocks would be potential aspects. At the same time we should also find cross-cutting goals and obstacles. Doing this in parallel has the advantage that we will be looking at the current design and we can always refine existing parts and find new features.

Then we need to build the aspects model in order to take aspects out of our current model, and then the final step is to compose aspects back into our model.

The last step in figure 4.1 is grayed out because in our work we will not touch that last step. It will remain as future work.

Figure 4.1 presents all the steps we must take in order to successfully accomplish the approach we are presenting in this work. Note that we only address the KAOS goal models. Operation, responsibility and object models will be addressed in future work.

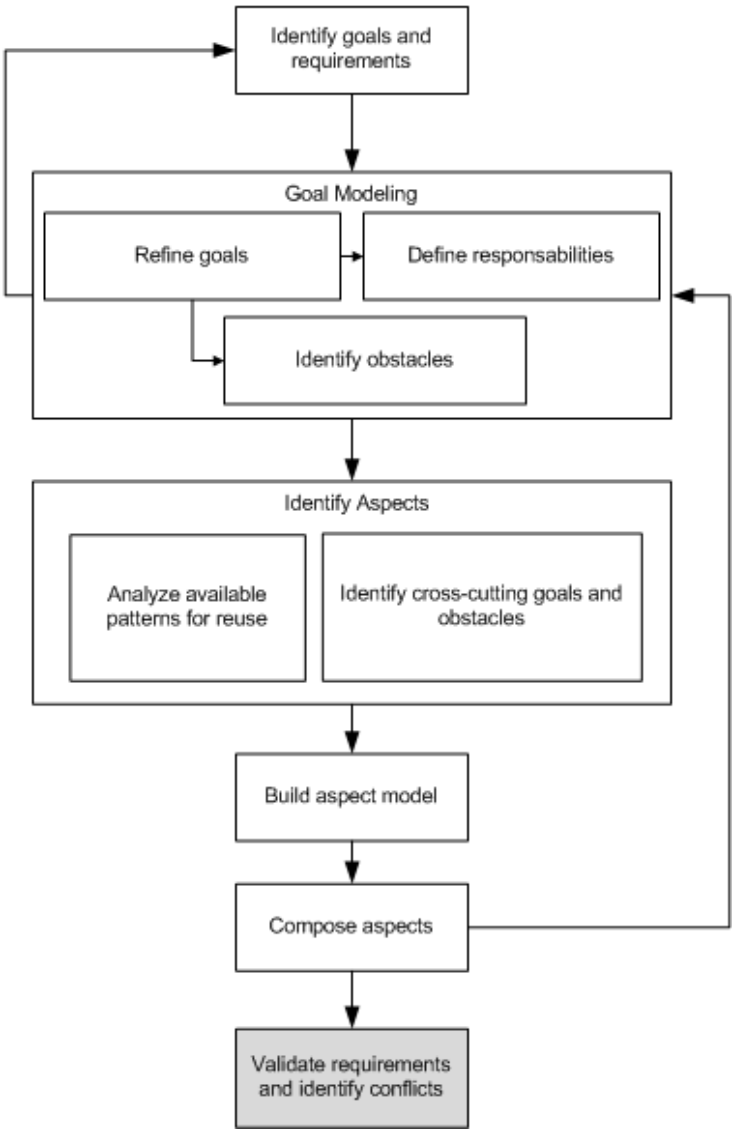


Figure 4.1 – Steps needed to apply our approach

4.1.1 Identify Goals and Requirements

Goal identification must be realized using current requirements engineering techniques such as interviews, ethnography, requirements and goal mining, etc. It is not the aim of this work to cover this in detail as it is not the main contribution of it. We assume that this activity is realized by applying the techniques mentioned above.

In this part of our work we will use the elevator case study from KAOS Tutorial [7], described as follows:

“The goal of the system is to build a system that satisfies all stakeholders' needs: functional and non-functional ones. Non-functional goal are classified as follows:

- the system must be safe, cheap, usable, and efficient;
- the system must preserve its environment;
- the system must respect the laws in force.

To satisfy a request for a given service, the passenger must issue a request and maintain it until executed; the system must respond by providing a service that satisfies the passenger's request.

In order to satisfy a service request, an infrastructure to perform the service must be available.

To reach the system state in which a request for a service is satisfied, the system has to reach a first state in which the service has been requested, a second state in which a service has been provided in response, in such a way that the service satisfies the request (the request is also expected to be maintained until the service is provided).”

4.1.2 Goal Modeling

Goal refinement. It involves decomposing goals into sub-goals until expectations and requirements are met. First we present the goal “Elevator called”. This goal is accomplished when the user presses the button to make the elevator stop at the floor where the user is standing. Figure 4.2 presents the goal.

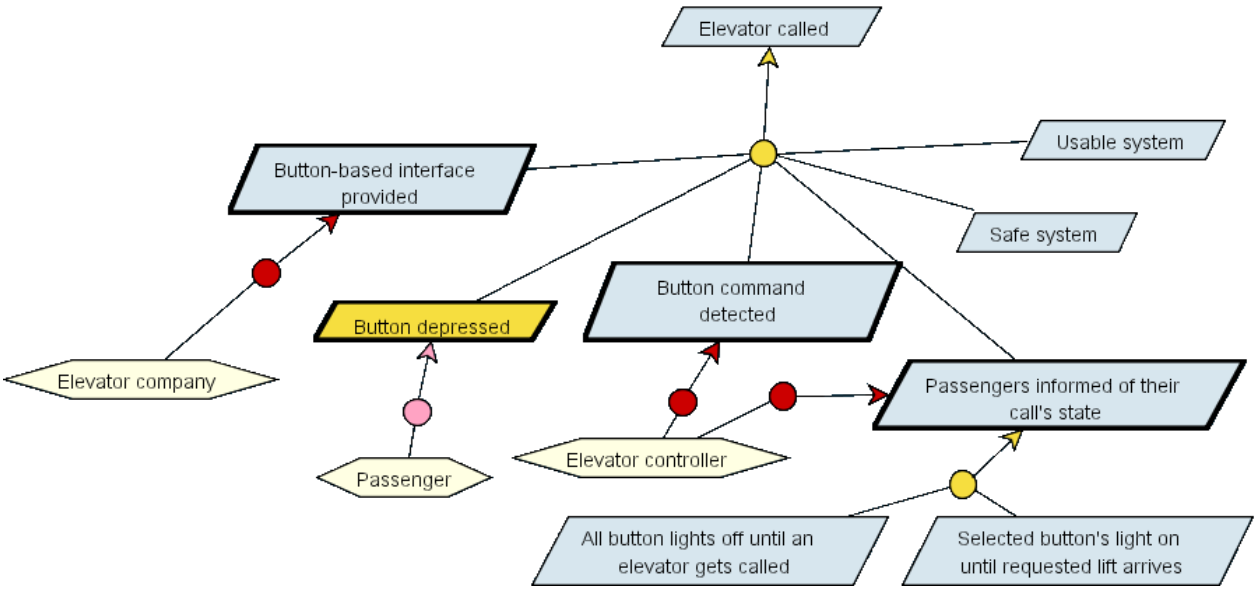


Figure 4.2 – “Elevator called” goal

Responsibilities. In this goal, the system has the expectation that the button will be by the user. On the other hand, the elevator controllers have both the responsibilities of detecting the command issued by the pressed button and inform the passengers of their call's state. Finally the elevator company has the responsibility of providing a button interface for the elevator.

In order to meet the goal "Passengers informed of their call's state" two other goals need to be accomplished. If both goals "All button lights off until an elevator gets called" and "Selected button's light on until requested lift arrives" are accomplished, then the other goal will be accomplished too.

The next goal to model is "Passengers brought to requested destination". This goal is the one detailing how to move from the passenger from the source floor to the destination floor. Figure 4.3 shows this goal.

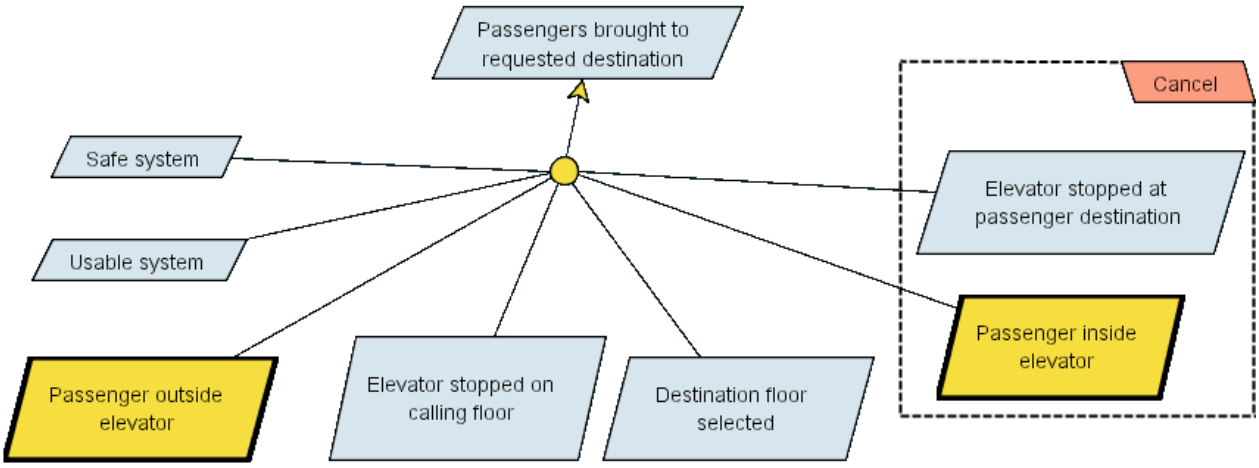


Figure 4.3 – "Passengers brought to requested destination" goal

In figure 4.3 we have the goal "Cancel" and a box around two sub-goals. This is not part of current KAOS notation and is introduced in this work. This type of goal is an *interruption* goal. We added this new type of goal to indicate the possibility to interrupt the current action anytime in any of the goals inside the box.

Other goal that was identified is the non-functional goal. In the example this goal is "Usable system".

Interruption Goals. In figure 4.3 the goal "Passengers brought to requested destination" includes another goal named "Cancel" which draws a box around some of the sub-goals of the top-most one. This goal in the top right corner of the box represents an interruption goal.

The need to create a new type of goal has to do with the fact that if we model a goal and it is intended to be an interruption, then we must connect it to the other goals. This would make the design very heavy and hard to follow. The way we solved the problem helps keep the design as simple as we can and we also add a new layer of abstraction, as we can represent interruption goals in a new way.

The way the user can interrupt the action is by pressing the stop button inside the elevator. To make it simple and not include a new type of goal, we could simply model this action as an obstacle. Although this was a possibility it seemed strange to model user actions as obstacles. In reality those kinds of actions cannot be considered obstacles, as there is nothing our system can do to overcome these difficulties. Actually we could avoid having a stop button inside the elevator, but that would greatly reduce usability besides also introducing safety problems.

As such, the need to incorporate interruption goals is explained by actions that could be took by the user, and cannot easily be modeled with the current types of goals. For the notation of this kind of goal, we just place the goal in there, and then draw a box around the goals at which the interruption goal can override. As all the other goals, interruption goals can be refined. Figure 4.4 presents this goal. We opted to model the goal with the same representation of an obstacle because it seems natural. An interruption goal is something that is not supposed to happen frequently and being represented with the same box as an obstacle, draws attention to it by the designer.

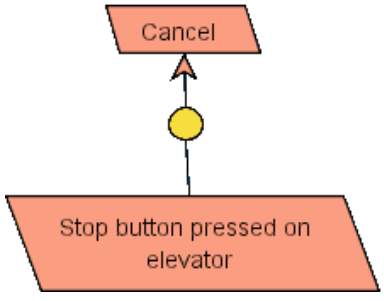


Figure 4.4 – "Cancel" goal refined

If this goal was scattered all over our model, we could also model it as an aspects, referencing it only once, thus simplifying the process.

At this point we have part of the elevator system modeled. We have introduced a new type of goal, in order to overcome an obstacle we found during this work. We needed a way to model an action by the user that would end on an interruption. One possible approach was to model it as an obstacle, but that was not adequate. If the user presses the stop button inside the elevator, does it represent an obstacle to our system? Actually in our understanding it cannot be an obstacle, it is a special kind of goal, as this depends directly on the user, and there is nothing we, as system designers, could do to prevent this.

In the event of this difficulty and because in KAOS we could not find a way to correctly implement this, we decided to add an interruption goal, with a simple and easy syntax.

We can reasoning that if we have scattered goals in such a small model, on a bigger and complex system, they will happen with such a frequency that will render the model illegible.

To reduce the complexity of the model we will rework the model and transform the scattered and cross-cutting goals into aspects. In this way, we will first get them out of the model, and then compose them back into the model.

The question is how will we find which goals will be our aspects. There are at least two ways to identify aspects. The first is common sense. In engineering, common sense can always be a good place to start, and in this case it usually is useful as the final result will be the identification of most of the aspects. Common sense has a big advantage, no fancy representations and no learning curve. Thought it could be improved by experience.

In our work, we will use a table to fill in which goals affect and cooperate to achieve other goals. We provide a tool to automate this task and to easily find which goals can be modeled as aspects. At the end of the process our table will have only the goals that can be modeled as aspects.

Identify Obstacles. We live in a world where we must interact with everything, and computer systems live in it too. As such, they must interact with other things, and they also face constraints in their operation. As expected, every system has obstacles they must overcome, and our system is not an exception.

The elevator must inform the users which is the direction it is taking. If for some reason it would not be possible to read the display, then this will be an obstacle. Figure 4.5 shows this obstacle.

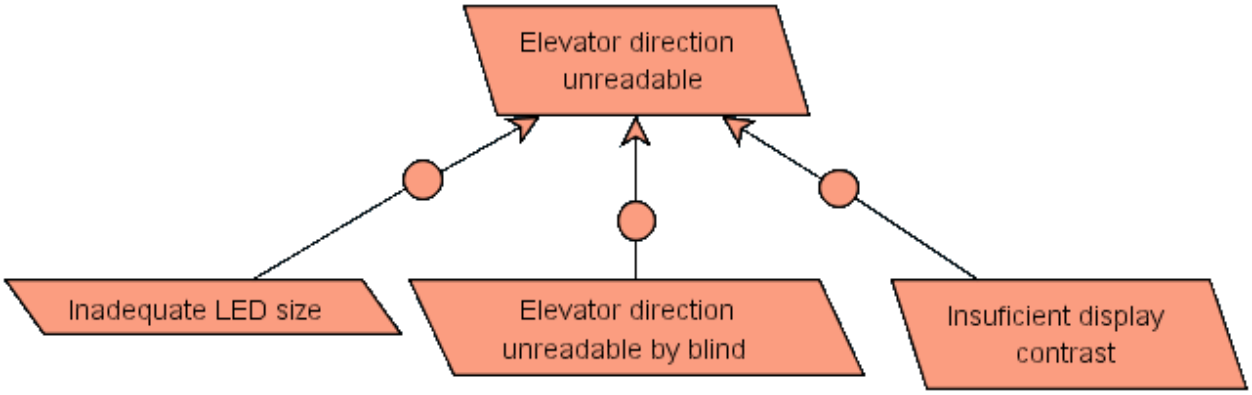


Figure 4.5 – "Elevator direction unreadable" obstacle

In order to solve the potential problems this obstacle poses to our system, we have to resolve them. The resolution of the previous obstacle is shown next in figure 4.6.

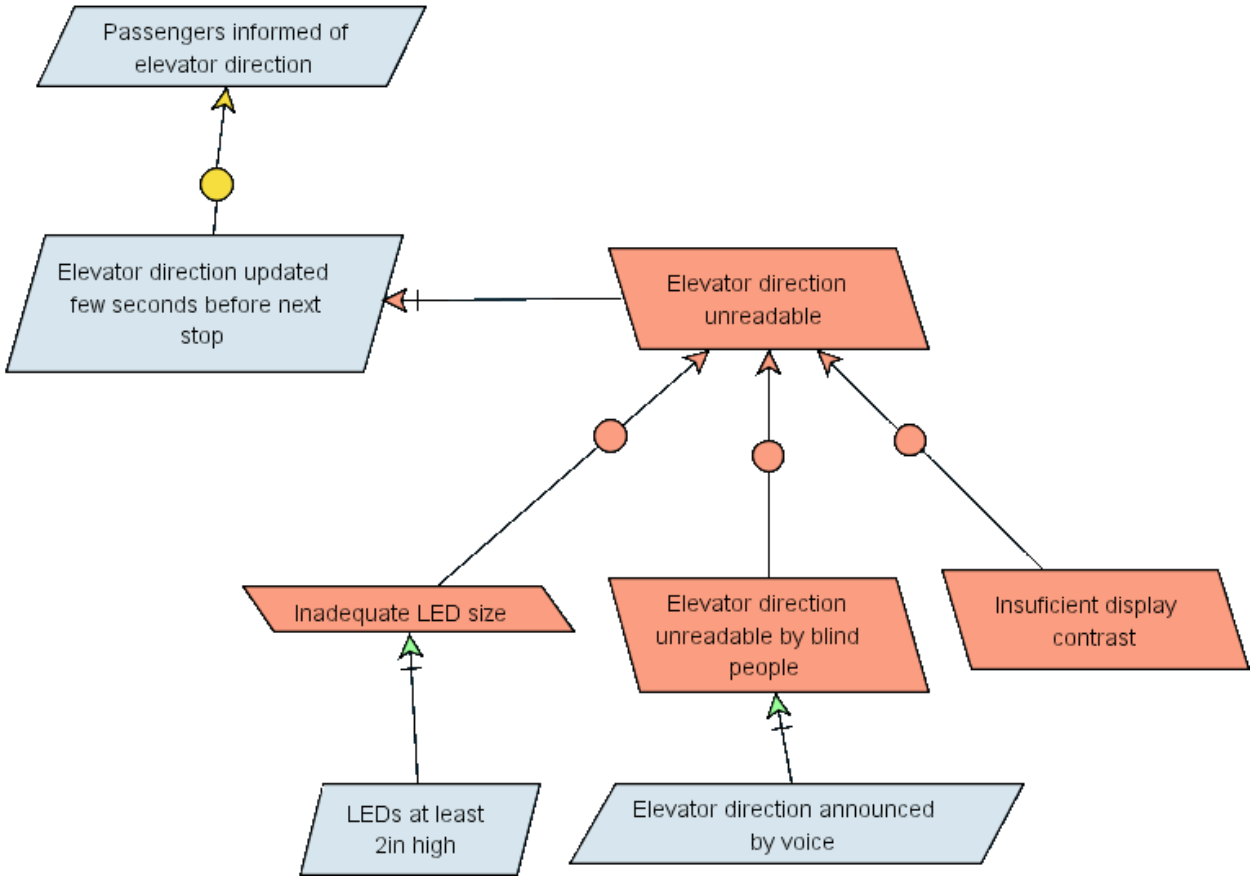


Figure 4.6 – Resolution of the obstacle

To improve the readability and meaning of obstacles we will describe relations between obstacles and goals. This is something that KAOS does not specify in more detail, and is also a contribution of our work.

Specifically, we will make possible to specify when, why or in which circumstances obstacles apply to goals. If more than one relation is needed they can be combined as a list separated with commas. This means that if we needed to use both "At" and "After" we would represent the relation between the obstacle and the goal as "[At, After]". Table 4.1 shows the possible relations between obstacles and goals.

Table 4.1 – Relations between obstacles and goals

At	The obstacle happens while some goals is being executed
Forbid	The obstacle prohibits some goal of being accomplished

Before	The obstacle precedes some goal
After	The obstacle is in succession of some goal
Break	The obstacle changes the functionality of some goal making it fail
|x	The obstacle reveals itself when a role is met by the goal the obstacle is applied to
(Condition)	The obstacle reveals itself if the condition is met

The |x will be explained in section 4.1.4 when we introduce the notion of roles. The previous obstacle now becomes (with the help of the relationship "at") as show in figure 4.7.

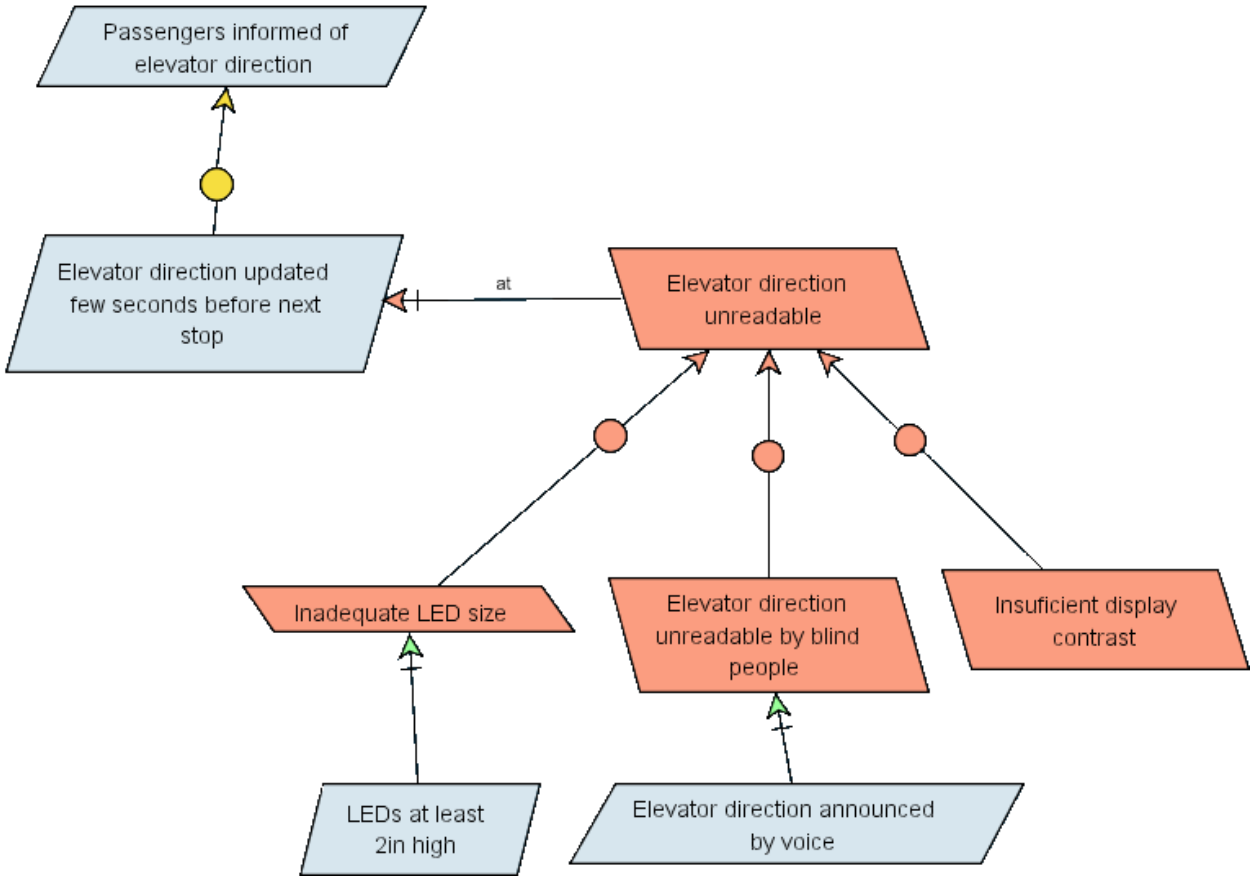


Figure 4.7 – Obstacle using relationship "at"

With obstacles identified, it is time to identify the aspects.

4.1.3 Identify Aspects

Analyze Available Patterns for Reuse. Here we look at the non-functional goals identified and look for patterns that are available for them. The goal "Safe system" is a generic goal that can be instantiated to several system applications, including the elevator system, characterizing a pattern. In order to achieve this goal, some sub-goals must be achieved first. The goal "Secure system" is composed of two sub-goals which must be achieved in order to the parent goal be achieved too. Figure 4.8 illustrates this goal.

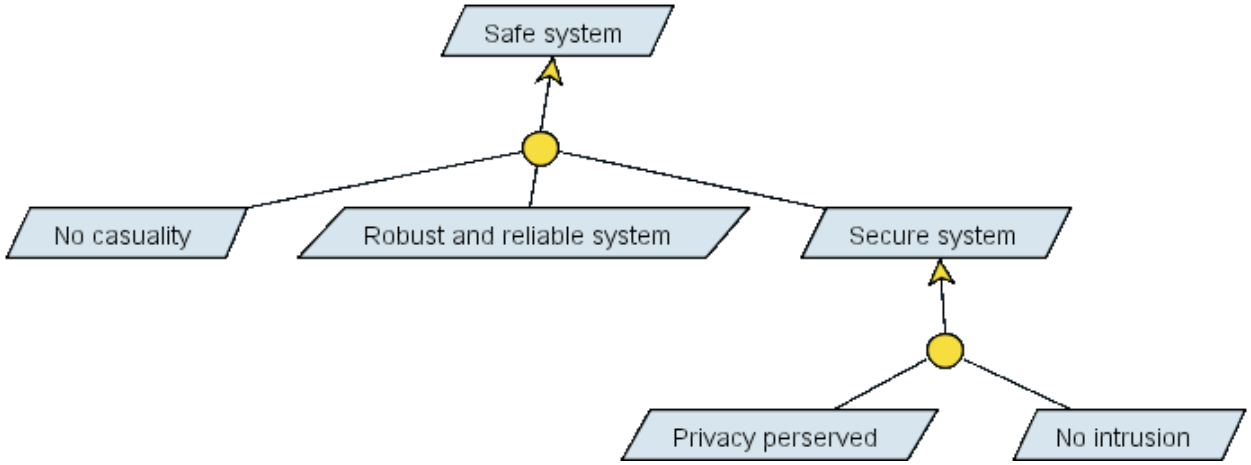


Figure 4.8 – "Safe system" goal

The next goal (also generic) to model is "Usable system". In order to accomplish this goal, four sub-goals must also be accomplished. Figure 4.9 shows the goal.

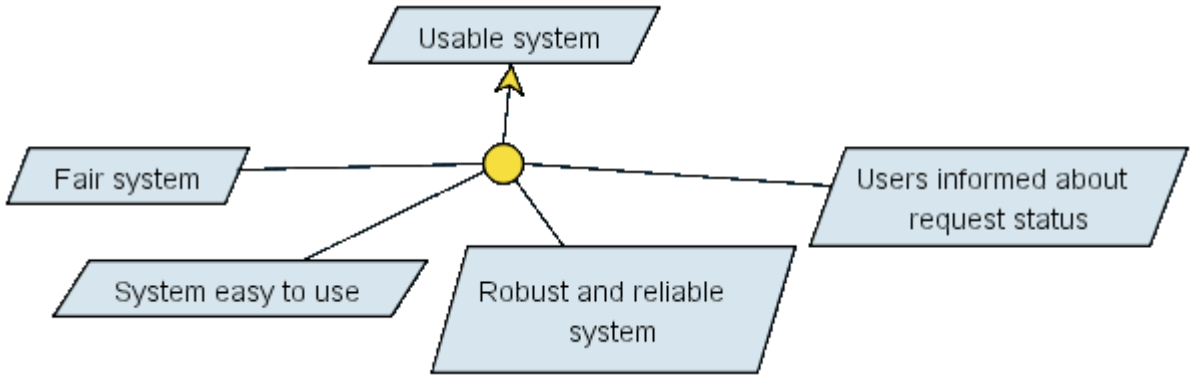


Figure 4.9 – "Usable system" goal

Identifying cross-cutting goals and obstacles (i.e. aspects). In order to help identify aspects, tool support is provided. Basically the tool automates the process of building a table to see the

relationships between goals on our system. Then we can hide the rows and columns where the goals are not cross-cutting other goals.

The table would have all the goals in each column, and then each row will have all the goals needed by the goals on the columns. Then we would check the cells where the goal on each row needed a goal on the columns. At the end, both the rows and columns having only one check, could be erased, and the others would remain there, and the goals would become the aspects.

Of course this would get very hard to do on pen and paper for very large and complex systems. This is the reason why a tool is provided. Figure 4.10 shows the table built with all the aspects and the cells checked to indicate which goals need which goals. Of course the table is just representative and only applies to the goals we have modeled in this short example.

	Elevator called	Passengers brought to requested destination	Usable system	Safe system	Passengers informed of their call's state	Secure system
Button-based interface provided	☑	☐	☐	☐	☐	☐
Button depressed	☑	☐	☐	☐	☐	☐
Button command detected	☑	☐	☐	☐	☐	☐
Passengers informed of their call's state	☑	☐	☐	☐	☐	☐
All button lights off until an elevator gets called	☐	☐	☐	☐	☑	☐
Selected button's light on until requested lift arrives	☐	☐	☐	☐	☑	☐
Usable system	☑	☑	☐	☐	☐	☐
Safe system	☑	☑	☐	☐	☐	☐
Passenger outside elevator	☐	☑	☐	☐	☐	☐
Elevator stopped on calling floor	☐	☑	☐	☐	☐	☐
Destination floor selected	☐	☑	☐	☐	☐	☐
Passenger inside elevator	☐	☑	☐	☐	☐	☐
Elevator stopped at passenger destination	☐	☑	☐	☐	☐	☐
No casuality	☐	☐	☐	☑	☐	☐
Robust and reliable system	☐	☐	☐	☑	☐	☐
Secure system	☐	☐	☐	☑	☐	☐
Privacy perserved	☐	☐	☐	☐	☐	☑
No intrusion	☐	☐	☐	☐	☐	☑
Fair system	☐	☐	☑	☐	☐	☐
System easy to use	☐	☐	☑	☐	☐	☐
Robust and reliable system	☐	☐	☑	☐	☐	☐
Users informed about request status	☐	☐	☑	☐	☐	☐

Figure 4.10 – Table to identify all the goals and dependencies between goals

After this first step, the tool can automatically reduce the table to include only the goals that will become aspects. The final table is shown next in figure 4.11.

	Elevator called	Passengers brought to requested destination	Usable system	Safe system	Passengers informed of their call's state	Secure system
Usable system	☑	☑	☐	☐	☐	☐
Safe system	☑	☑	☐	☐	☐	☐

Figure 4.11 – Table reduced to only the goals that could become aspects

As obstacles can be modeled as aspects, the same tool can be used to check which obstacles are aspects and which are not. We will not show the procedure applied to obstacles as it will not bring anything new, it analogous to the way we do with goals.

Now that all our aspects are identified, we can take the next step and build the aspect model.

4.1.4 Build the Aspect Model

In order to build the aspect model we will use roles. The first step is to model our aspects and incorporate roles into the model.

By using roles [16, 41], discussed in section 3.4, modeling could be supported by the programming language, if it is an AO language. A role enables the aspect to have extra capabilities. It automatically gains extra knowledge about the environment, as the role can only replace the elements of the model, where the pattern is met.

What is the real importance of incorporating roles into aspects? With the introduction of roles, we can have late binding which means that we would not introduce complexity into the model while that complexity is not needed.

When incorporating roles into our aspects, we will create them on the point-cuts. Point-cuts will be represented by a dashed box, and the connection between that box and the aspect will be a dashed line.

A point-cut [17] is a set of join points, where the join point is where both the goal and the aspect meet. First we will model our “Usable system” aspect and incorporate roles into it. Figure 4.12 shows it.

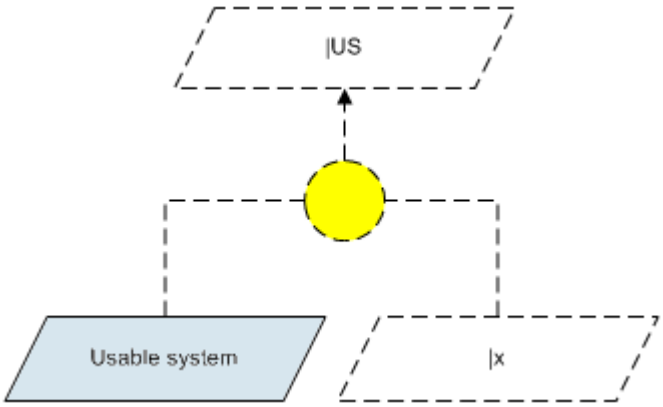


Figure 4.12 – “Usable system” modeled as aspect using roles

In the previous figure we added both roles |US and |x to be instantiated to concrete goals. In the end roles work as model variables. This is intended as a way to do pattern matching on our model. This way, we have to match it to something where “Usable system” goal is connected to some other goal (the other leaf goal in figure 4.12) and is under some top goal (|US).

4.1.5 Compose Aspects

This is the last step of our approach. In order to compose aspects, we must bind the roles, i.e. instantiate them to concrete model elements. Doing some pattern matching we find two places where we can apply our aspect. At this point, and with pattern matching already done, we only need to bind the roles to our goals.

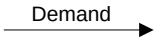
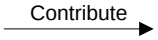
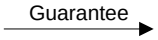
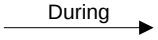
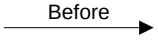
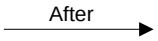
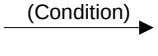
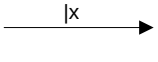
- Compose “Usable system”
    - Bind |US to “Passengers brought to requested destination”
    - Bind |x to “Elevator stopped on calling floor”

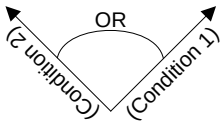
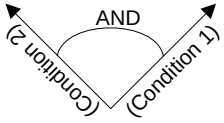
As we can see by figure 4.3 the final result after composing aspects is the same, reason why we do not show the composition of “Passengers brought to requested destination” goal.

In order to provide a clear representation of aspects and the goals to which they apply, we will introduce a way to specify the relationships among them.

If more than one relation is needed they can be combined as a list separated with commas. This means that if we needed to use both “During” and “After” we would represent the relation between the aspect and the goal as “[During, After]”. Table 4.2 shows these relationships.

Table 4.2 – Relations among aspects and goals

	The aspect/goal requires something from another goal/aspect so it can be accomplished. The way it is applied is given by the arrow. This relation can only be used between siblings.
	The aspect/goal in conjunction with other goals, helps the top goal to be accomplished. The way it is applied is given by the arrow. This type of relation implies a stronger commitment from the goal where the arrow starts.
	The aspect/goal provides something so another goal/aspect can be accomplished. The way it is applied is given by the arrow.
	The aspect/goal is executed during another goal/aspect. The way it is applied is given by the arrow. This relation can only be used between siblings.
	The execution of the aspect/goal precedes the execution of another goal/aspect. The way it is applied is given by the arrow.
	The execution of the aspect/goal is executed in succession to another goal/aspect. The way it is applied is given by the arrow. This relation can only be used between siblings.
	This is aimed at executing some aspect/goal if a condition is met. This can be any kind of condition. The way it is applied is given by the arrow.
	This executes the goal/aspect if the aspect/goal that is on the origin of the arrow meets some role. The way it is applied is given by the arrow.

	This executes one or another goal/aspect depending if Condition 1 or Condition 2 is met. The way it is applied is given by the arrow.
	This executes one and another goal/aspect if Condition 1 and Condition 2 are met. The way it is applied is given by the arrow.

We can now rebuild the aspect shown earlier and include the relationship between the aspect and the goal. Figure 4.13 shows just that.

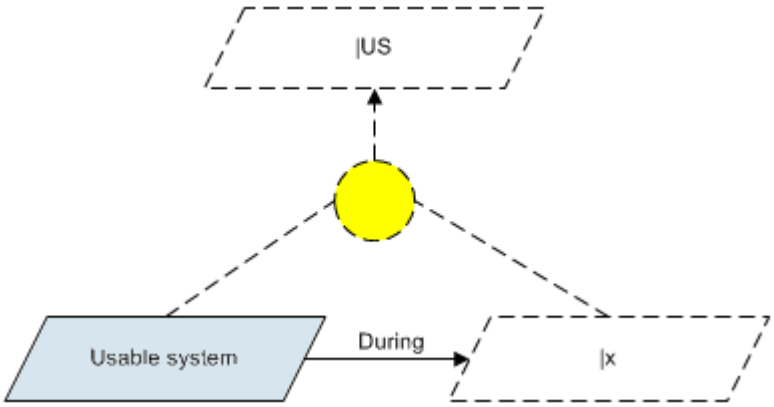


Figure 4.13 – "Usable system" aspect with relationship

The use of the "During" relationship is important in one way. Even when using "AND" constructs, we do not state anything about temporal execution. When we have an "AND" construct, we know that goal A must be achieved in conjunction with goal B in order for the parent goal to be achieved too. When we add the "During" relationship, we do state that both goals A and B must execute and be achieved at the same time, or at least, the execution of one of the goals must be carried out and the goal achieved while the other goal is still executing and before that other goal being achieved.

4.2 SUMMARY

In this chapter we introduced our approach. The base of our approach is the KAOS modeling method, to which we added a new type of goal, "interruption goal". We also extended the base modeling of KAOS in order to incorporate aspects and to use roles.

We added the feature of both adding aspects to the goal model, and also to the obstacle model, in order to simplify both. New kinds of relationships were introduced both at obstacles and aspectual levels, in order to explicitly state what a relationship meant and to make the learning curve softer.

In order to identify aspects, a tool was provided so the process of identification of aspects between all the goals and all the candidate aspects were simplified, easier and faster than if done with paper and pencil. Finally we built the aspect model, recurring to composition, in graphical and textual form. Next chapter apply the approach to a case study.

CHAPTER 5

CASE STUDY AND COMPARISON WITH OTHER APPROACHES

A case study is applied to our approach. This way it will be possible to validate it and understand the advantages of applying our developed approach to a complex system. Finally, we will compare AspectKAOS with other approaches, highlighting the main differences.

5.1 EXPENSE REPORTS

The case study will be a web application with the objective of submitting expense reports. This is a real case study which was developed by the author as the member of development team of the company ”Etnaga”, where the author of this dissertation works. In this application, enterprise employees will be able to submit their expense reports and have them approved or rejected by management. Actually who will approve these reports will be the administrator of the application.

The application provides a way to the user login into the system, after it has been added by the administrator. To login and access the system, the user must provide his credentials and the credentials must exist on the system. If the login was successful then the user will be given

permission to access the system. When the user is logged on to the system, he will be able to submit his expense reports, download the template to fill the expenses or view the approval status of his reports.

If the logged on user is the administrator, he will have access to all the reports. As such he is able to list all the reports and check for the approval status each report currently has. Approval status has three statuses, pending, approved and rejected. When a report is uploaded the status of the report is pending. Then when the administrator approve or reject the report, it will get the approved or rejected status.

To fill the expense report first the user must have some kind of template of the report. In the web site, there is a link to download the template of the expense report. The template is an excel file, the user can fill in the appropriate fields.

When the user has filled his report, he can submit the report through the Submit Report link.

If the user desires to see all of his reports one link with the name "All Reports" is provided. If on the other hand the administrator follows the "All Reports" link, then he will see all the reports on the system. If he goes into the detail of each report he will be able to approve or reject the report. From this description we can already find some goals.

- Submit expense report – this will be responsibility of the user.
- Approve or reject expense report – this will be responsibility of the administrator.
- View reports – responsibility of both users and administrator. Users can only see their own reports. Administrator can see all reports in the system.

From the description just given we can infer that the system should be secure for the users. This means that one user should not be able to see other user reports, and that it would not be possible for someone other than the administrator log in as the administrator.

This means that privacy should be preserved at all costs. On the other hand, to enforce privacy, the system should validate all input it gets. Validating input is something that will be scattered around all system. Actually input validation will be one cross-cutting concern, as we need to

validate input both at the login step and we also need to validate the contents of the report submitted.

This kind of cross-cutting concern really represents an aspect. From now on, cross-cutting concerns will be referred to as aspects.

5.1.1 Goal Identification and Modeling

To this point the problem has already been described and the first step is to model it using our approach. This means that first we must identify goals and requirements. This is done by reading the description of our case study. The identified goals are:

- User must login into the system;
- User must submit expense report;
- User must visualize his/her reports;
- Administrator must visualize all reports;
- Administrator may approve or reject report.

The next step is to refine goals. This has the two sub-tasks that should be held in parallel. As such it is possible at the same time to define responsibilities and identify obstacles.
We have as NF goals that the system must be safe and it must also be efficient and usable, as an unusable system will not be any good for users. The patterns associated with them are described in section 5.1.3.

The next goal we need to model is the “Expense report obtained by system”. In order to achieve this goal, some sub-goals must be achieved first. Firstly, the user has the expectation that the interface will make the task of submitting expense reports easier by providing a feature that will enable online submission.

The web site has the responsibility to capture the requests made by the users of the web application. With both goals met, the goal “Report selected for submission” can be successfully accomplished.

Then the system designer has the responsibility to provide an interface. In conjunction with the previous goal, it will be possible to meet “Report uploaded” goal. On his side, the web site also has the responsibility to validate input, and inform the user of submission state.

At this point in time, the goal “Report uploaded” when successfully achieved in conjunction with “Input validated” will make it possible that the goal “User informed of successful report submission” be carried on thus finally accomplishing the goal “Expense report obtained by system”. Figure 5.1 displays our goal.

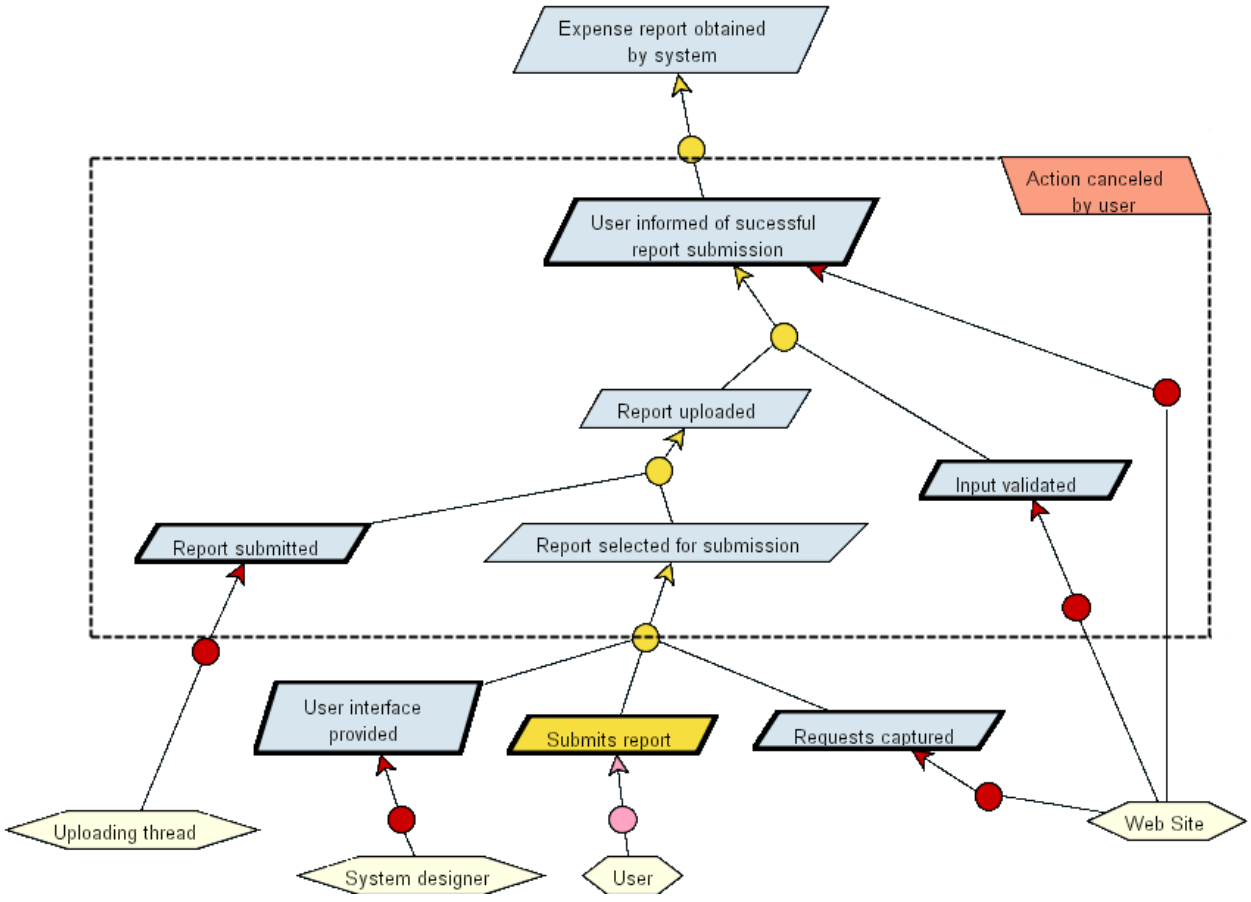


Figure 5.1 – “Expense report obtained by system” goal

In this model we specify an interruption goal (inside a box, which has an orange box on the right upper corner).

The next goal, “Expense report visualized” is easy to understand and as such we will not get into the details of it. In order for the report to be successfully visualized, it must validate all the input and also the desired report must be chosen. This can be seen in figure 5.2.

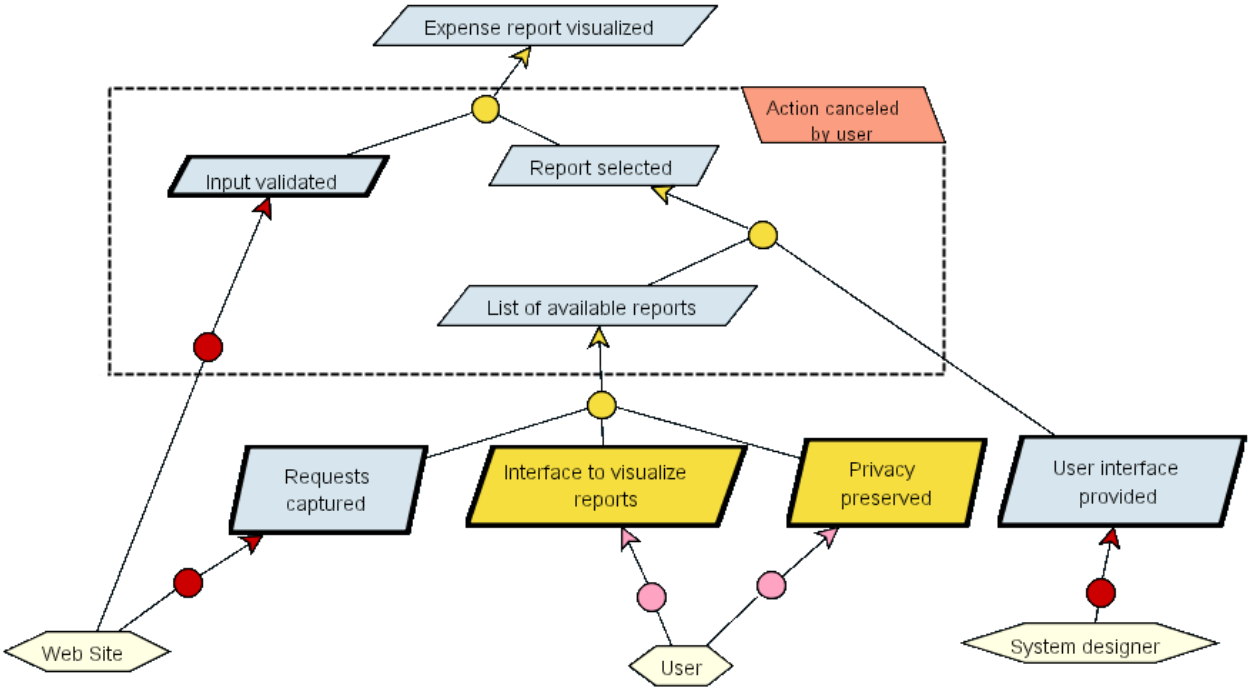


Figure 5.2 – “Expense report visualized” goal

The next figure (5.3) shows goal “Report accepted”. The approval of a report is done by the administrator and the user must be informed of the approval state.

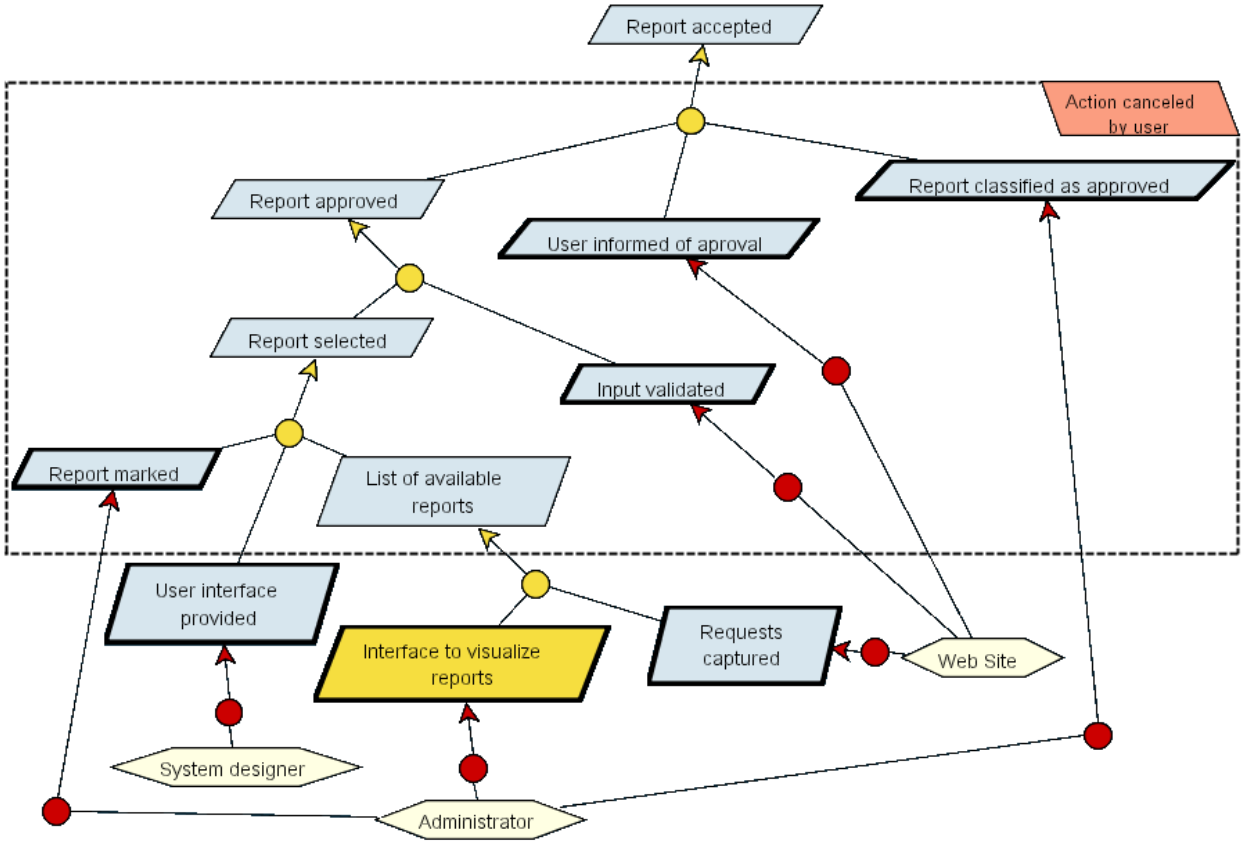


Figure 5.3 – "Report accepted" goal

In figure 5.4 we present the goal "Report rejected". It is identical to the previous goal "Report approved"; thought the only difference is that the goals now reject the report.

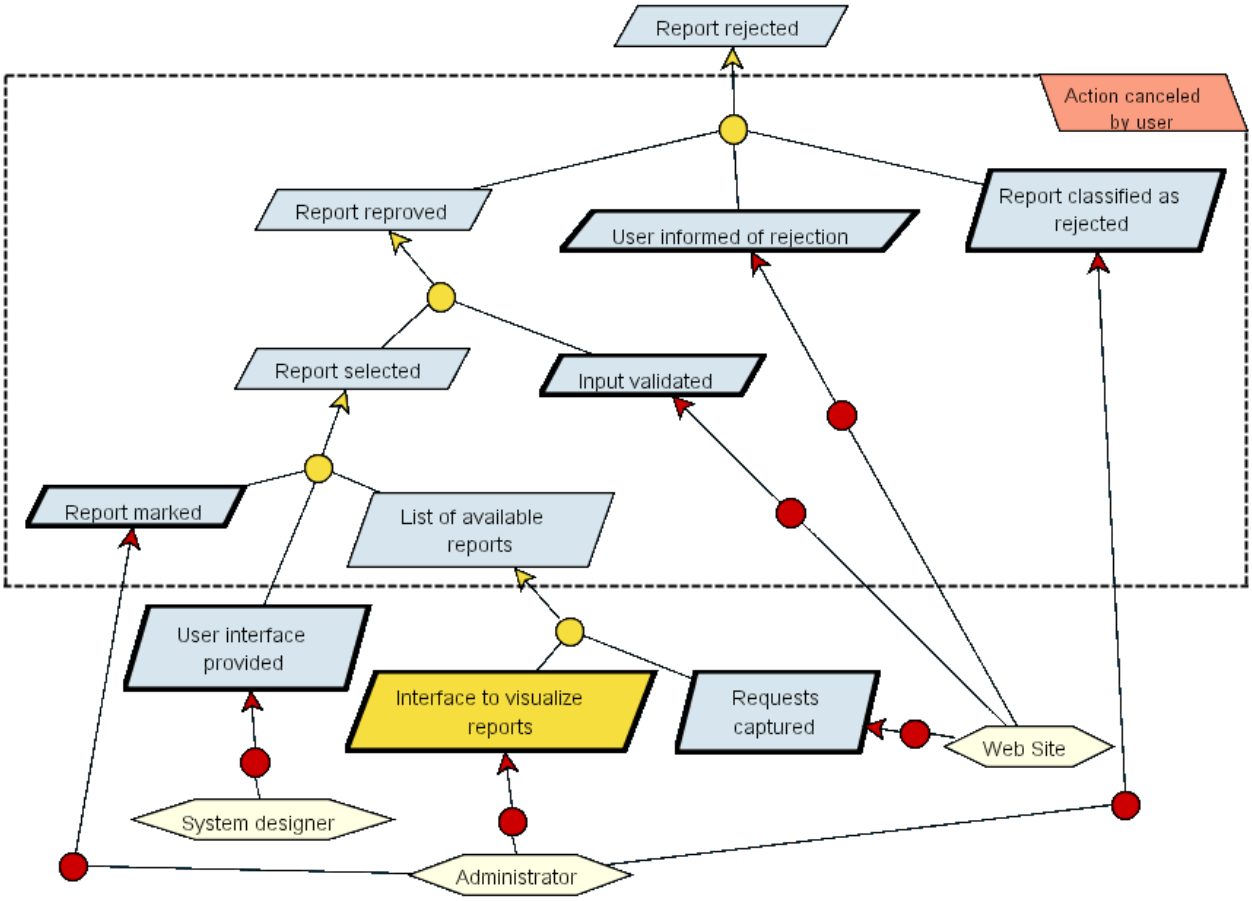


Figure 5.4 – "Report rejected" goal

Every model includes a dashed box with a goal on the right upper corner with the background in orange. As we stated earlier, those are interruption goals. It was important to create a new type of goal since on a web application the user can cancel his actions anytime.

There are at least two ways the user can cancel his actions. If the web application interface provided a cancel button, the user could always use that button. The current interface of the case study does not provide a cancel button. On the other hand, the user can always close the web browser and in this way, his actions are canceled (actually not quite, as if the data has already traveled to the server, at least that part of the action will be taken).

The need to incorporate and create interruption goals is explained by this actions that could be took by the user, and cannot easily be modeled with the current types of goals.

To understand the notation of the goal, a quick look at it and its use in the model will make it clear. We just place the goal in there, and then use the dashed box to delineate where the action could have any impact, or when could the action be taken.

As all the other goals, interruption goals can also be refined. Figure 5.5 presents this goal.

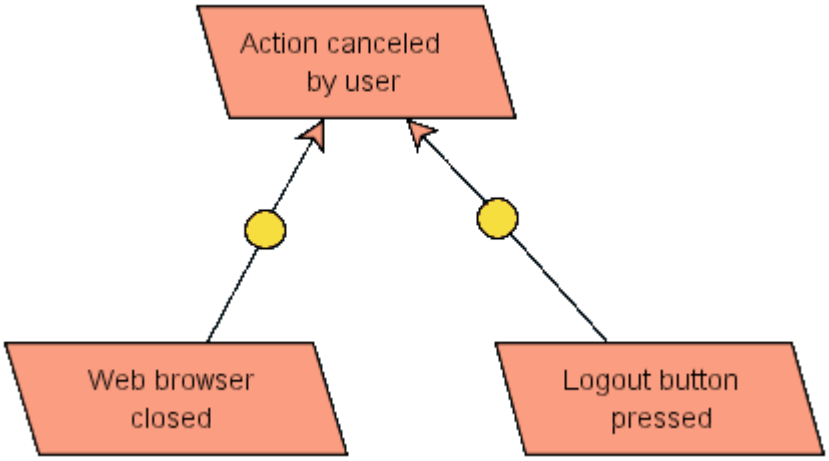


Figure 5.5 – "Action canceled by user" interruption goal

We have introduced a new type of goal, in order to overcome an obstacle we found during this work. To overcome this obstacle we created a new kind of goal because in KAOS we could not find a way to correctly implement this. We decided to add an interruption goal, with a simple and easy notation, and this new type of the goal, makes the modeling extremely easier.

Although the model is small it is also somewhat complex. The reason we say the model is complex is because for such a small model there are some goals that are scattered all over the model.

We can reasoning that if we have scattered goals in such a small model, on a bigger and complex system, they will happen with such frequency that will render the model illegible. To reduce the complexity of the model we will rework the model and transform the scattered and cross-cutting goals into aspects. In this way, we will first get them out of the model, and then compose them back into the model.

The question is how we will find which goal will be our aspects. There are at least two ways to identify aspects, as talked earlier. In our work, we will use a table to fill in which goals affect and cooperate to achieve other goals. We provide a tool to automate this task and to easily find which goals can be modeled as aspects.

5.1.2 Identify Obstacles

As every system has obstacles, ours is not an exception. As this is a system to use on an intranet or maybe the internet, if for some reason the connection has a problem, or if the server is not available, an obstacle to the use of the system will emerge. Of course these obstacles will apply to specific goals and not to the system as a whole. We could model obstacles that apply to the whole system, like a power outage, but it would not represent added value for our work.

If we look at all the points where a failure of the system can result in an obstacle then generic obstacles will emerge. We will model one generic obstacle related to the failure of hardware and connection. Figure 5.6 shows this obstacle.

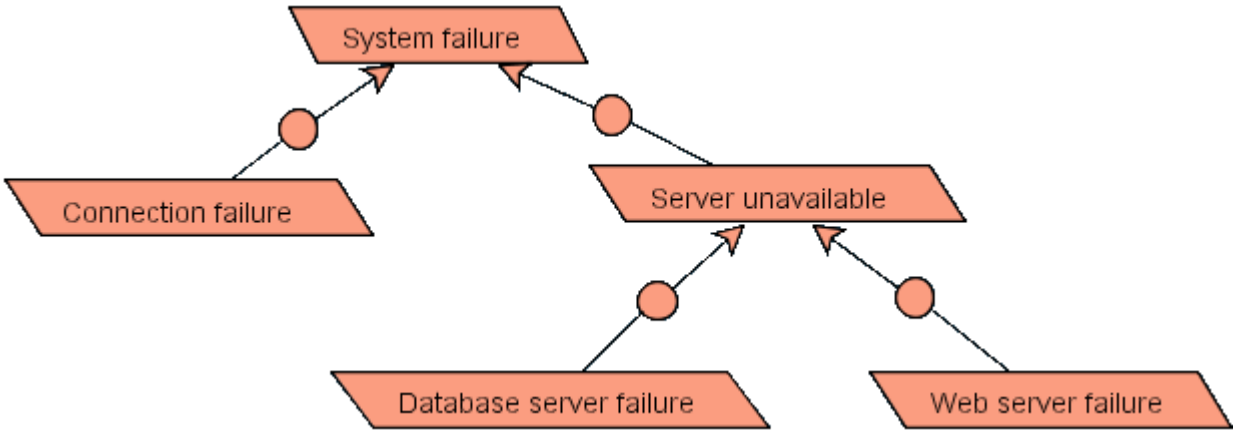


Figure 5.6 – Generic obstacle “System failure”

This obstacle is built from two other obstacles, “Connection failure” and “Server unavailable”. These two obstacles surely would render our system unusable. Furthermore the obstacle “Server unavailable” is also composed of two other obstacles. The reason to have circles connecting to each of the children obstacles is that in order for the parent obstacle to manifest itself, it would be sufficient for only one of the children obstacles to manifest. For example, if the web server fails, then the server will be unavailable and as such we will have a system failure. This means that the

system will fail even if the connection does not fail. These obstacles must be resolved because we cannot take the risk of being taken offline because we do not have any redundancy. The proposed resolution for the obstacles is shown in figure 5.7.

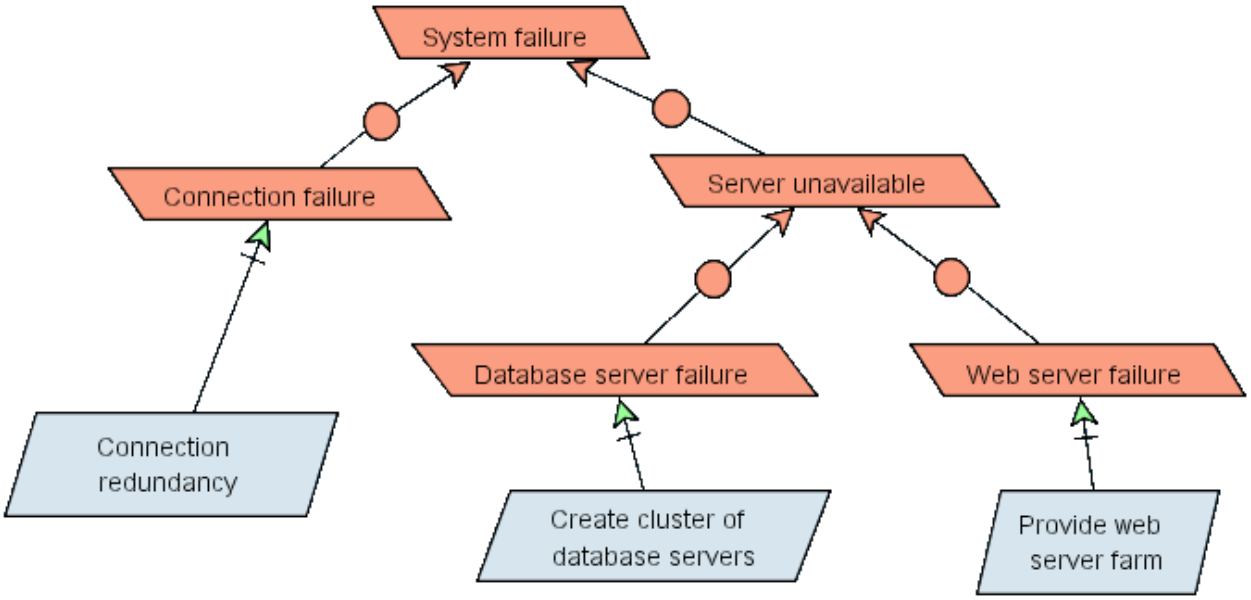


Figure 5.7 – Resolution of "System failure" obstacle and its children obstacles

The previous obstacle can be applied to our system. This means we should adapt our obstacle to our system. The next figure (5.8) shows the application of the generic obstacle "System failure" to our system and to the goals which have it an obstacle.

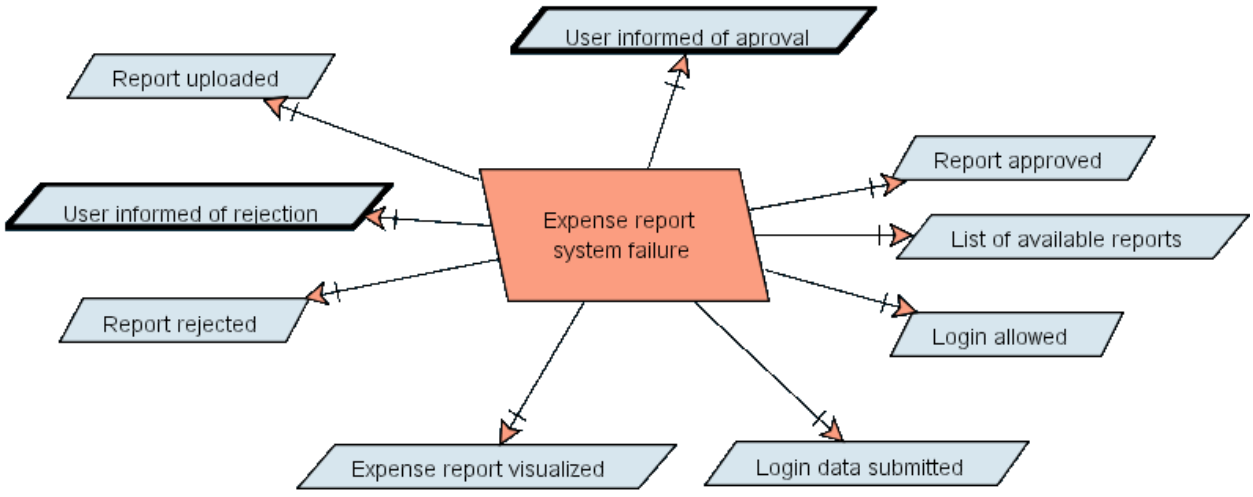


Figure 5.8 – Goals to which "Expense Report System failure" obstacle applies

Although we already have one obstacle (derived from the generic obstacle "System failure") there are more obstacles to our system. The other goals that certainly have obstacles are "Report approved" and "Report rejected". Those obstacles are modeled on figure 5.9.

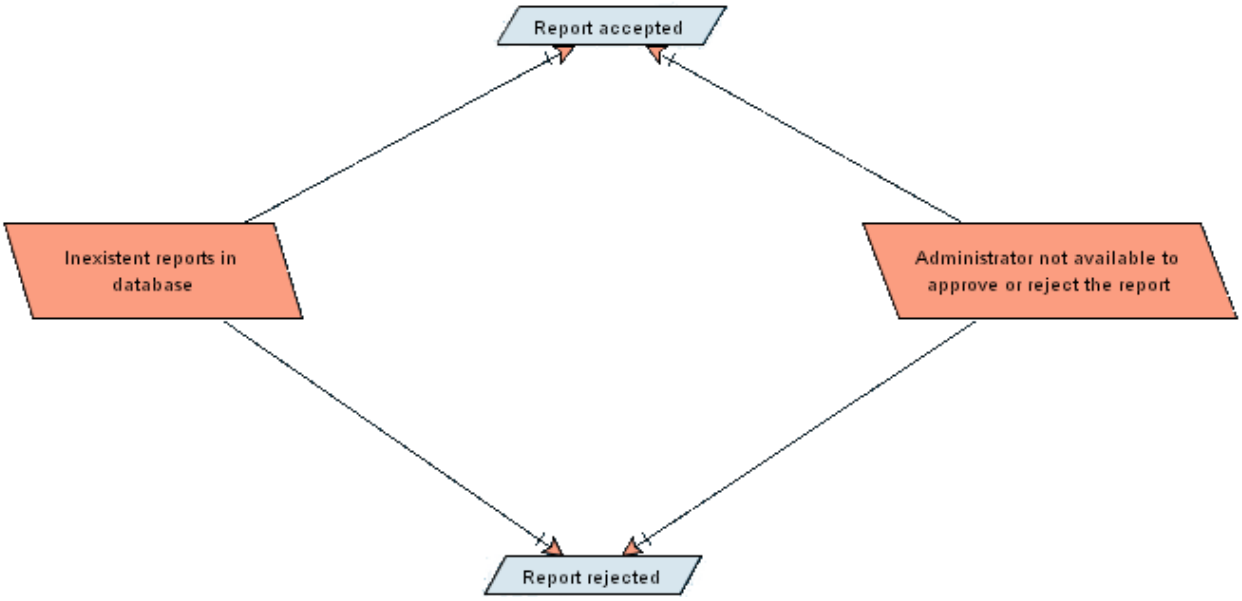


Figure 5.9 – Obstacles "Report approved" and "Report rejected"

To improve the readability of obstacles we will describe the relations between obstacles and goals. This is something we introduced in our work. The details of the relations can be seen in chapter 4 in section 4.1.2 in table 4.1. Figure 5.10 presents obstacle "Expense Report System failure" with the relations to the goals shown in the model.

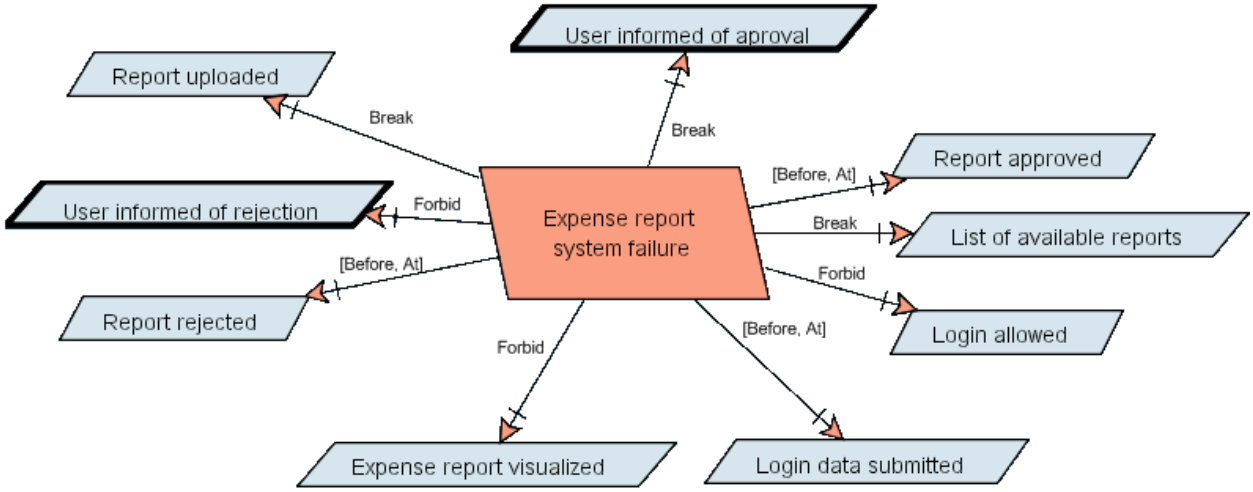


Figure 5.10 – Relations between "Expense Reports System failure" obstacle and the goals

The relationships in the previous figure clarify the effect the obstacle has on some of our system goals.

When the “Expense Report System” has a failure, it has an impact on all of the goals of our system. Although it impacts all of our goals, the impact is different in each one.
If the system fails then the functionality to list all the available reports will not be possible. Actually the objective of listing all the available reports will not be accomplished, and as such, we can say the obstacle “Expense report system failure” breaks the objective “List of available reports”.

On the other hand, we can possibly have the obstacle “Expense report system failure” not allowing for some objective to be met. In the specific case we can say when the expense report system fails, it will not be possible to visualize the selected expense report, and as such, “Expense report system failure” forbid “Expense report visualized” objective.

It is also possible for an obstacle to happen at some specific time. Actually when the system fails, we can have a problem with the login of the user. If the system fails before or while the user is submitting his login credentials, then it will affect the objective “Login data submitted”. This objective will be affected before or at (while) the objective is trying to be accomplished.

Figure 5.11 presents obstacles “Inexistent reports on database” and “Administrator not available to approve or reject report” and the relations to the goals they affect.

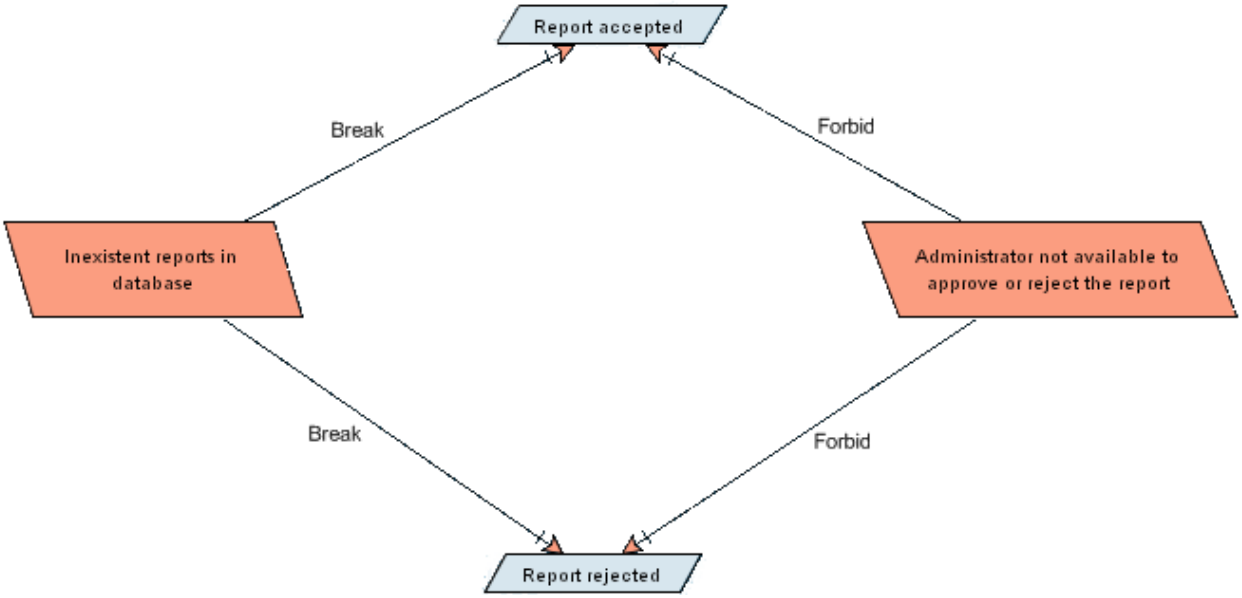


Figure 5.11 – Relations between obstacles and goals

With obstacles identified and the relations between obstacles and goals identified, it is time to identify the aspects. Next section will go deep into it.

5.1.3 Identify Aspects

Analyze available patterns for reuse. The first goal we will model is “Login allowed”. Actually this goal can be seen as a pattern, as it could easily be reused in other system. This goal is accomplished when the user is allowed to log into the system. Figure 5.12 shows the goal. As it can be seen the user has an expectation for the system to keep its privacy. On the other hand, the web site has both the responsibilities of capture user requests and validating input.

On its turn, the system designer has the responsibility to provide a user interface. With the goal “User interface provided” and “Login data inserted” accomplished, we have the goal “Login data submitted”. This last goal in conjunction with “Input validated” goal, make the goal “Login allowed” a reality.

The user can cancel his action anytime he wants. This means he cannot even insert his credentials.

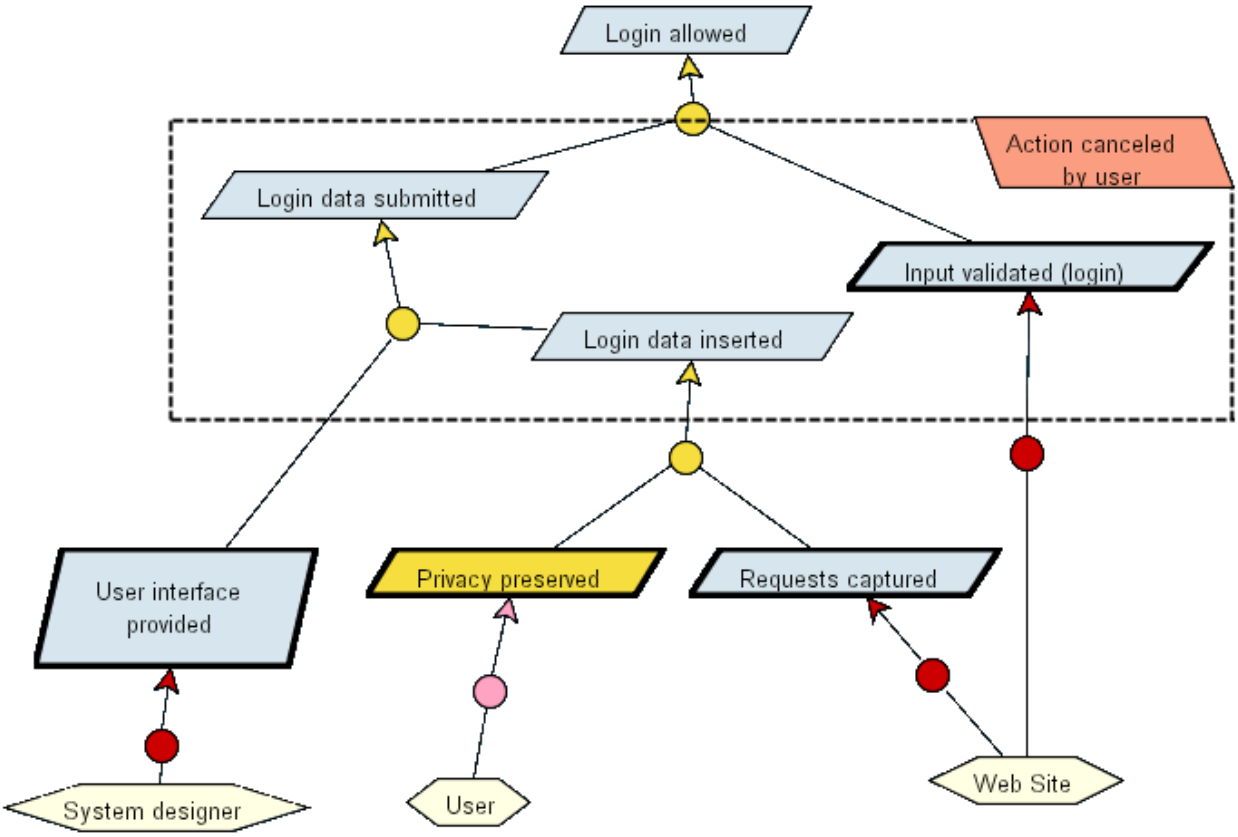


Figure 5.12 – "Login allowed" goal

To model the NF goal "Secure System" first the goals "Privacy preserved" and "Input validated" were placed and connected to "Secure system" goal. Then the bottom goals were introduced. But the three goals at the bottom made only the "Data validated" goal, and when validating input, one should always validate both the data and also validate any path the user could insert to transfer a file. In order to preserve the privacy of the user we must be sure that the cookie will not be stolen and that we use https in order to protect at least the user credentials.

Figure 5.13 shows the goal. Note that the goal roles defined. They will eventually be instantiated to concrete goals. Also in a pattern, roles and non-roles can co-exist in the same model.

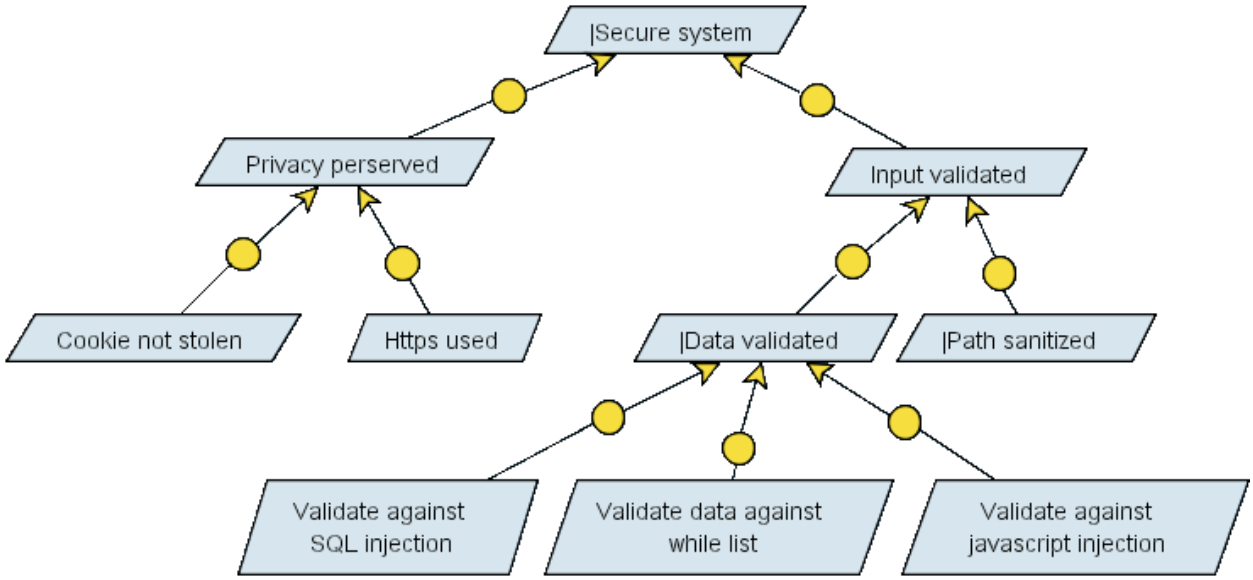


Figure 5.13 – Model of "Secure System" goal

Figure 5.14 shows the goal "Usable and efficient system".

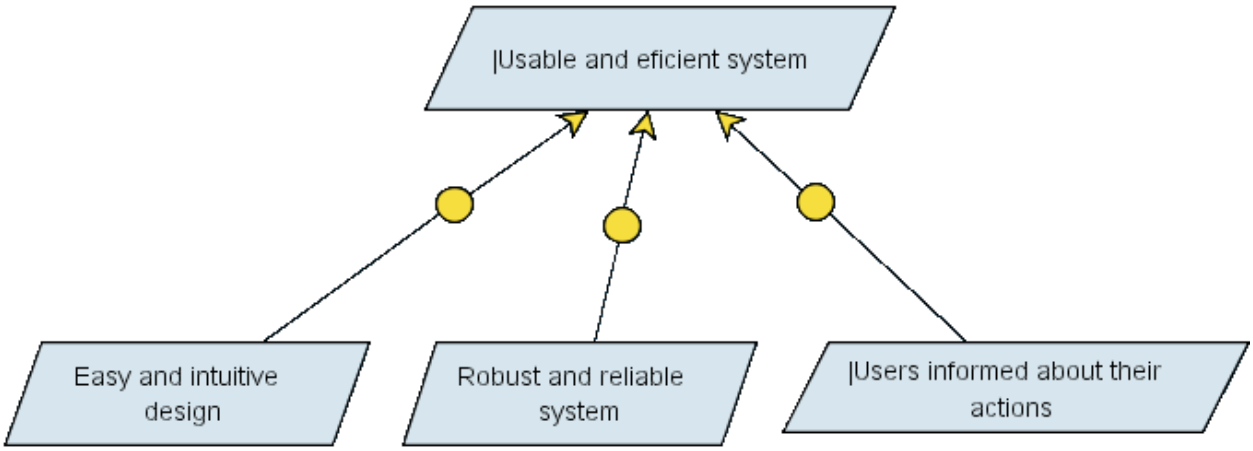


Figure 5.14 – Model of "Usable and efficient system" non-functional requirement

Identifying aspects. In order to help when we need to identify aspects with the tool that was introduced in chapter 4 section 4.1.3. After the table is complete we will use the feature the tool provides to hide the rows and columns where the goals are not cross-cutting goals.

Figure 5.15 shows the table built with the tool. This figure shows the table with all the goals in our system.

Goals	Login data inserted	Login data submitted	Login allowed	Report reproved	Report rejected	Report accepted	Report approved	List of available	Report selected	Expense report visualized	Report selected for submission	Report uploaded	User informed of submission state	Expense report submitted
User informed of su...	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☑
Input validated	☐	☐	☑	☑	☐	☑	☐	☐	☐	☑	☐	☐	☑	☐
Report uploaded	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☑	☐
User interface provi...	☐	☑	☐	☐	☐	☐	☐	☐	☑	☐	☐	☑	☐	☐
Report selected for ...	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☑	☐	☐
Requests captured	☑	☐	☐	☐	☐	☐	☐	☑	☐	☐	☑	☐	☐	☐
Interface to submit r...	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐
Report selected	☐	☐	☐	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐
List of available rep...	☐	☐	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐
Interface to visualiz...	☐	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐
Privacy perserved	☑	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐
Report accepted	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐
User informed of ap...	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐
Report reproved	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐
User informed of rej...	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐
Login data submitted	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Login data inserted	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

Figure 5.15 – Table with all the goals in our system

After this first step, the tool can automatically reduce the table to include only the goals that will become aspects. The final table is shown next in figure 5.16.

Goals	Login data inserted	Login data submitted	Login allowed	Report rejected	Report approved	List of available reports	Report selected	Expense report visualized	Report selected for submission	Report uploaded	User informed of submission state
Input validated	☐	☐	☑	☐	☐	☐	☐	☑	☐	☐	☑
User interface provided	☐	☑	☐	☐	☐	☐	☑	☐	☐	☑	☐
Requests captured	☑	☐	☐	☐	☐	☑	☐	☐	☑	☐	☐
Privacy perserved	☑	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐

Figure 5.16 – Table with only the goals that will become aspects

Obstacles can also be modeled as aspects. The same tool can be used to check which obstacles are aspects and which are not. We will not present the procedure applied to obstacles as it will not show anything new.

5.1.4 Modeling Goals as Aspects

Now it is time to model the goals found on the previous step as aspects. In order to do this we will also use roles. The first step is to model our aspects and incorporate roles into the model.
In order to make the model easier to follow we will use the relationships introduced in chapter 4, section 4.1.5.

First we will model our “Privacy preserved” aspect and incorporate goals into it. Figure 5.18 shows the new model. Note that, they already make part of the pattern described above. But since we are interested in just a part of it we just reuse that part to build the aspect.

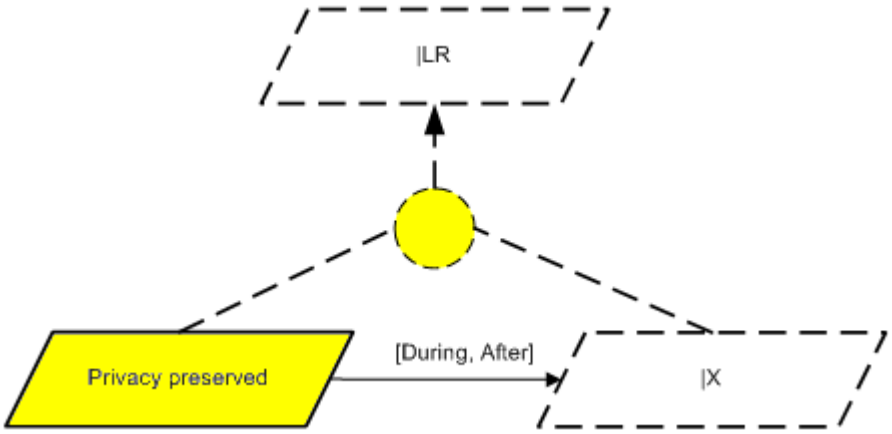


Figure 5.18 – Aspect “Privacy preserved” with roles and relationship

The modeling of this aspect includes roles. Roles are both the brother and the father of the aspect, as we need to do pattern matching with the aspect in order to find where it belongs. When doing pattern matching we must bind the roles to some goal on the system. Then by defining the roles as one brother of the goal that is our aspect, and another role as the parent of our aspect we make it possible to pattern matching with current goals.

All the following aspects are modeled with roles in this way in order to avoid confusion when it comes to pattern matching.

Another feature of the previous image is the use of a relationship between “Privacy preserved” and “|X” role. This implies that the goal will be executed both at the same time and after the execution of the sibling goal. Of course the execution must terminate before the execution of the parent goal. As in this case it is an expectation, this indicates the agent expects privacy to be preserved during and after the goal represented by the role “|X” be achieved.

The next aspect “Input validated” is also modeled in the same way as the previous one. Although the modeling is somewhat the same as the previous aspect, we say that our “Input validated” aspect will be run at the same time as is sibling. Figure 5.19 represents this goal modeled as an aspect.

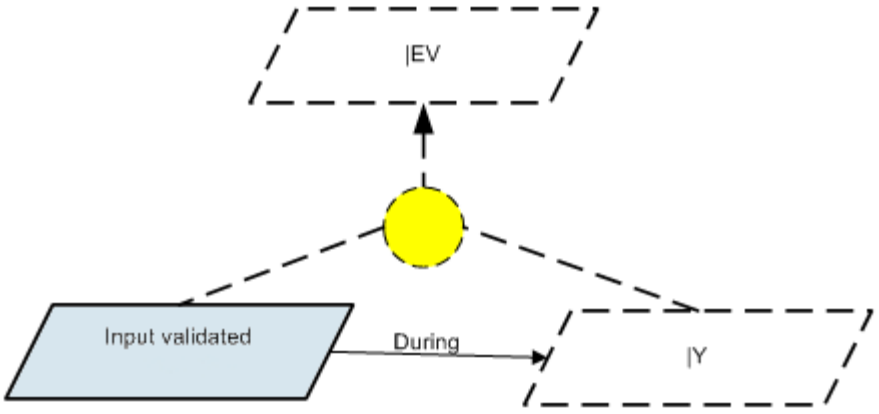


Figure 5.19 – Aspect “Input validated” with roles and relationship

This aspect can be used either to validate reports or to validate the login data. In order to use one or another, only one piece of the pattern will be used, the piece that can carry the job at hand. This simplifies the overall model as we do not add one more model just because of a variation in some aspect. The next aspect to model is “Requests captured”. It is shown in figure 5.20.

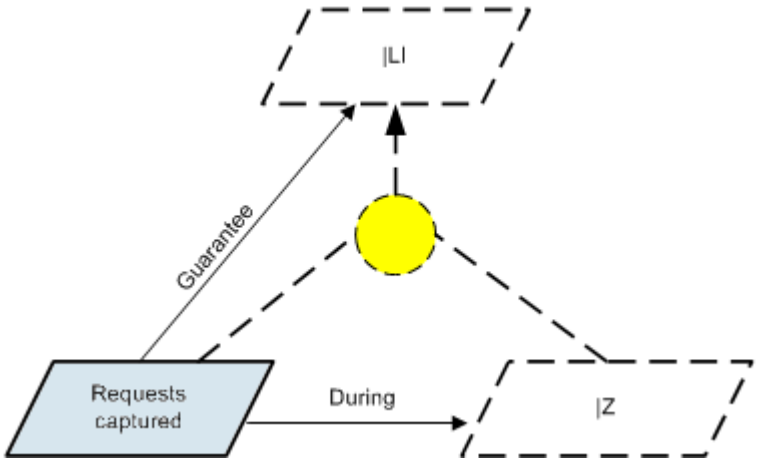


Figure 5.20 – Aspect “Request captured” with roles and relationships

This aspect shows a new kind of possibility when creating relationship. The aspect “Requests captured” has both a relationship with the parent and with the sibling. This means that our aspect “Requests captured” provides something to his parent, and is executed at the same time as his sibling. By providing something to his parent, it helps the parent on achieving itself, this means, that probably without what our aspect provides to the parent, the parent will not be achieved.

The next one is “User interface provided” aspect. It can be seen in figure 5.21.

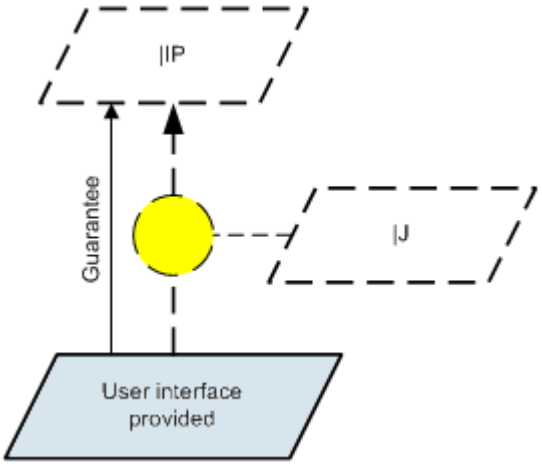


Figure 5.21 – Aspect "User interface provided" with roles and relationship

The previous aspect also provides something to the parent goal in order for the parent to be achieved.

There is still one more before the final one, so we must model "List of available reports". The model can be seen in figure 5.22.

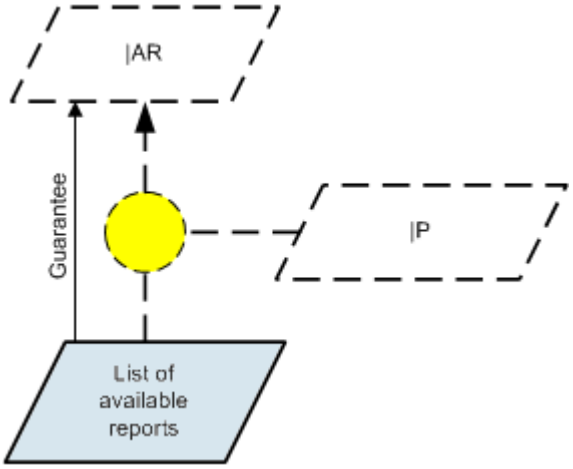


Figure 5.22 – Aspect "List of available reports" with roles and relationship

Finally we had only left the "Report selected" aspect. Figure 5.23 shows this last goal modeled as an aspect.

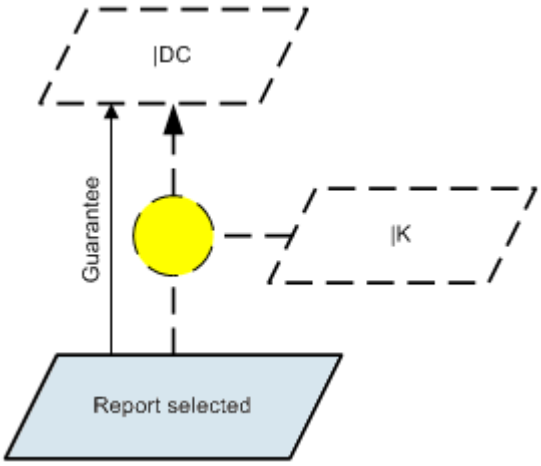


Figure 5.23 – Aspect “Report selected” with roles and relationship

After the goals modeled as aspects, we could and should model the obstacles as aspects too.

5.1.5 Modeling Obstacles as Aspects

In order to find which obstacles could become aspects we could use the provided tool. Although this would not be necessary as it is easy to identify which goals will become aspects without the tool. First we will model “Inexistent reports in database” obstacle as an aspect. It is shown in figure 5.24.

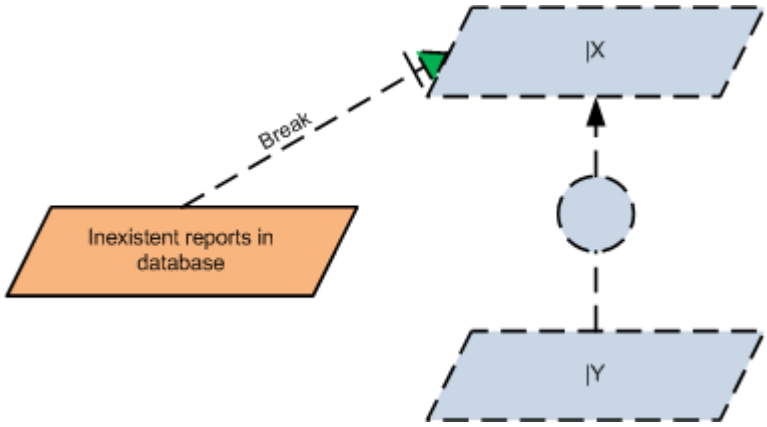


Figure 5.24 – Obstacle “Inexistent reports in database” modeled as an aspect

Previous figure represents an obstacle modeled as an aspect. This means, we will be able to apply pattern matching to it. Having the relation between our obstacle and the role when we do pattern matching and bind the role of |X we will break the functionality of whatever goal it binds to.

The second obstacle to model as an aspect will be “Administrator not available to approve or reject report”. Figure 5.25 presents this obstacle.

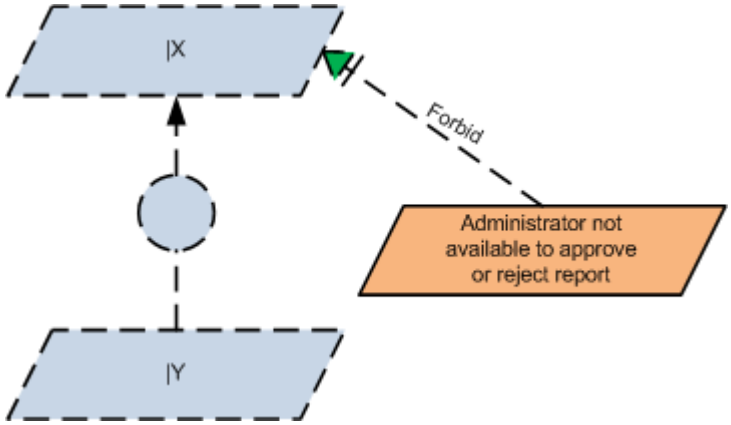


Figure 5.25 – Obstacle “Administrator not available to approve or reject report” modeled as an aspect

Lastly we must remodel the obstacle “Expense Report System failure”. This obstacle will be modeled in three different models. This has to do with the fact that the relations between this obstacle and the goals are not always the same.

Figure 5.26 shows the obstacle “Expense Report System failure” with the relation forbid.

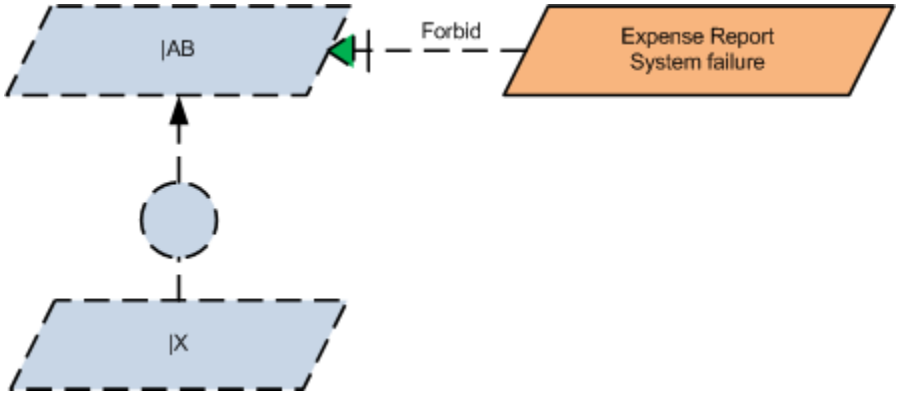


Figure 5.26 – Obstacle “Expense Report System failure” modeled as an aspect with forbid relation

Then we proceed to model “Expense Report System failure” with the relation break. Figure 5.27 shows that situation.

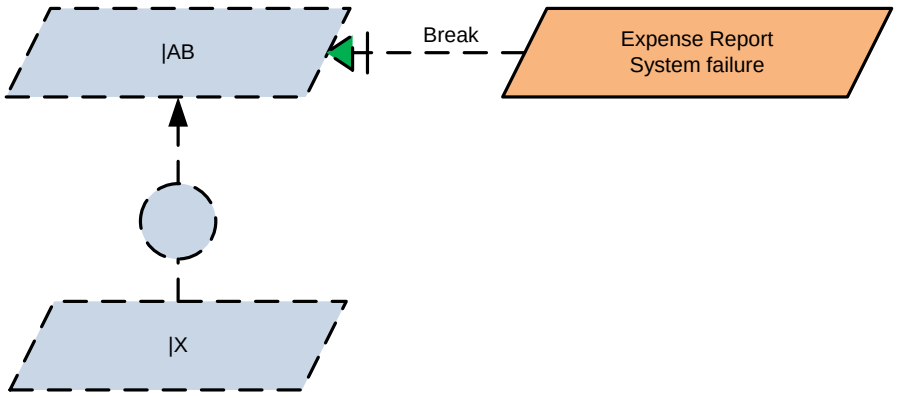


Figure 5.27 – Obstacle "Expense Report System failure" modeled as an aspect with break relation

Finally we can model "Expense Report System failure" with the relationships before and at. Figure 5.28 depicts that case.

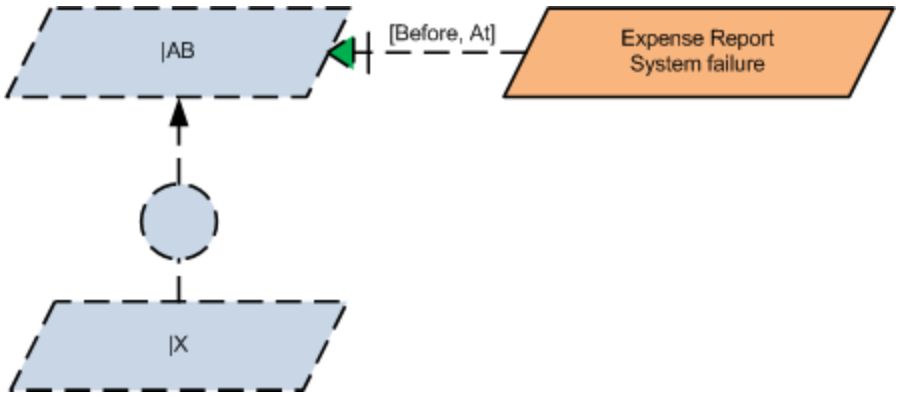


Figure 5.28 – Obstacle "Expense Report System failure" modeled as an aspect with before and at relationships

5.1.6 Compose Aspects

This is the last step of our approach. In order to compose the aspects, we must bind the roles.
In this step, if we plan to use only part of a generic aspect, then this is where a new model will be included only with the parts we need. In the current system, we will not cut down our generic aspect, as we are already using part of "Secure system" goal.

At this point we only need to bind the roles to our goals.

The binding of roles to goals has a hidden step. That hidden step is pattern matching. We will pick the aspects from section 5.1.4 and 5.1.5 and match them against the goals we have in section 5.1.1 and 5.1.2. Of course if we had an extensible tool, this could be automated inside the tool

and make automated pattern matching a reality. This would greatly simplify the whole process. The binding of roles is presented in table 5.1.

Table 5.1 - Binding roles

Compose “Privacy preserved”	Bind |LR to “Login data inserted” Bind |X to “Requests captured”
	Bind |LR to “List of available reports” Bind |X to “Interface to visualize reports”
Compose “Input validated”	Bind |EV to “User informed of successful report submission” Bind |Y to “Report uploaded”
	Bind |EV to “Expense report visualized” Bind |Y to “Report selected”
	Bind |EV to “Report approved” Bind |Y to “Report selected”
	Bind |EV to “Report reproved” Bind |Y to “Report selected”
Compose “Requests captured”	Bind |LI to “Report selected for submission” Bind |Z to “Submits report”
	Bind |LI to “Login data inserted” Bind |Z to “Privacy preserved”
	Bind |LI to “List of available reports” Bind |Z to “Interface to visualize reports”
Compose “User interface provided”	Bind |IP to “Report selected for submission” Bind |J to “Submits report”
	Bind |IP to “Login data submitted” Bind |J to “Login data inserted”
	Bind |IP to “Report selected” Bind |J to “List of available reports”
Compose “List of available reports”	Bind |AR to “Report selected” Bind |P to “User interface provided”
Compose “Report selected”	Bind |DC to “Expense report visualized”

	Bind |K to “Input validated (report)”
	Bind |DC to “Report approved” Bind |K to “Input validated (report)”
	Bind |DC to “Report reproved” Bind |K to “Input validated (report)”
Compose “Administrator not available to approve or reject report”	Bind |X to “Report accepted” Bind |Y to “Report approved”
	Bind |X to “Report rejected” Bind |Y to “Report reproved”
Compose “Inexistent report in database”	Bind |X to “Report accepted” Bind |Y to “Report approved”
	Bind |X to “Report rejected” Bind |Y to “Report reproved”

In table 5.2 we present the binding of roles where we have relations.

Table 5.2 – Binding of roles with relations

Bind “Expense Report System failure” with “Forbid” relation	Bind |AB to “Expense report visualized” Bind |X to “Report selected”
	Bind |AB to “Login allowed” Bind |X to “Login data submitted”
Bind “Expense Report System failure” with “Break” relation	Bind |AB to “Report uploaded” Bind |X to “Report selected for submission”
	Bind |AB to “List of available reports” Bind |X to “Request captured”
Bind “Expense Report System failure” with “[Before, At]” relation	Bind |AB to “Report rejected” Bind |X to “Report reproved”
	Bind |AB to “Login data submitted” Bind |X to “Login data inserted”
	Bind |AB to “Report approved” Bind |X to “Report selected”

After composing aspects, we can see by the next figure that we have the same figure as figure 5.2. The only difference is that now we have a relationship between our aspect and his sibling goal “Interface to visualize reports”. Figure 5.29 displays this new model.

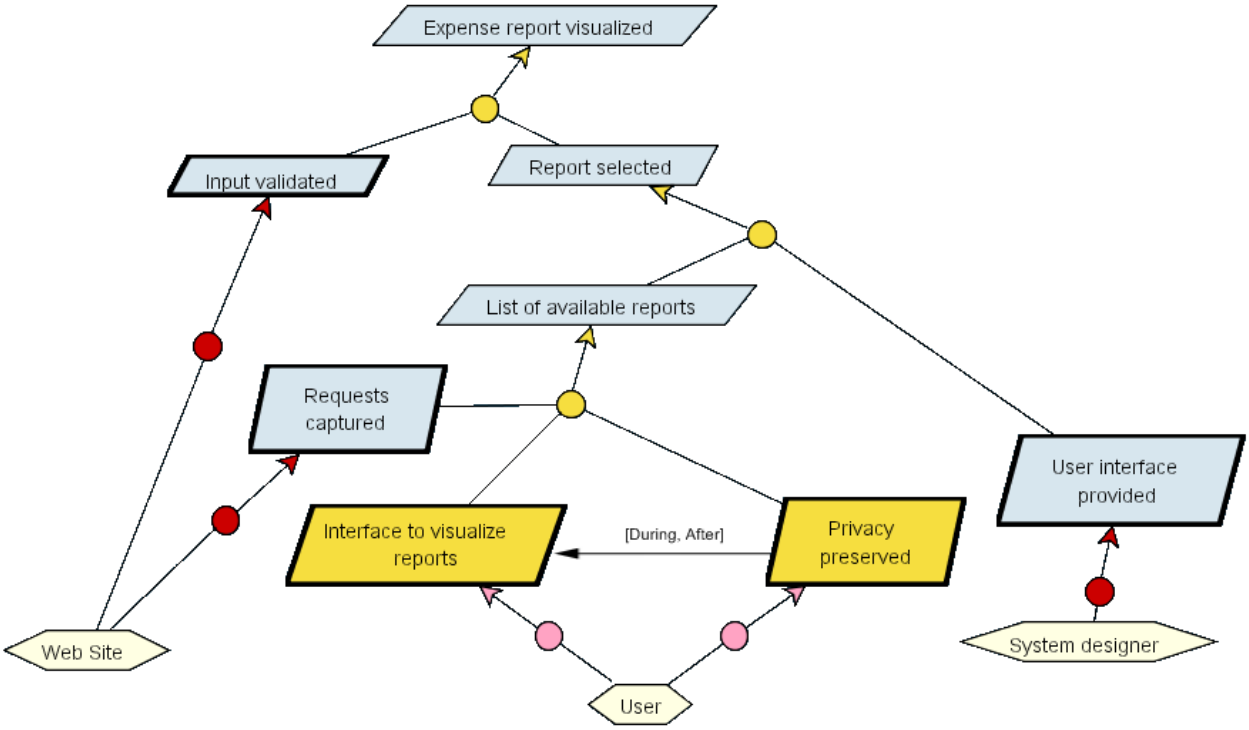


Figure 5.29 – “Expense report visualized” after inserting aspect back into the model and with relationship

5.2 Comparing Approaches

In this section we will compare our approach, AspectKAOS, to other approaches. Comparison will be between AspectKAOS to goal-oriented approaches with and without aspects.

5.2.1 Criteria

The rationale behind the criteria chosen to provide a comparison between approaches was to try to cover the relevant features that each approach should provide. Specifically, we want to compare our approach against other approaches reasoning how all the approaches satisfy the criteria. They are obstacles modeling, conflict modeling, tool support, modularization, cross-cutting concerns modeling, explicit relationships and composition mechanism, that are discussed as follows.

Obstacles modeling is an important feature of each approach, obstacles do exist at all times, be them provided by nature or created by humans. We as software engineers must live with them and overcome them, so any system will have to overcome obstacles at some point.

Conflict modeling is another important subject. Conflicts happen when two non-functional requirements constrain each other. Also, requirements from one stakeholder will create conflicts with requirements from another stakeholder.

When modeling systems, we need tools in order to create a representation of our model. Tool support facilitates and encourages modeling of the proposed approach and as such is an important criterion when comparing approaches.

Other important features we must support are modularization and modeling of cross-cutting concerns. These features apply to the whole system and are crucial for the system's architecture. Modeling of cross-cutting concerns is a particular kind of modularization.. Modeling of cross-cutting concerns aims at taking out of the model the concerns which are tangled and scattered all over the system. On the other hand, modularization is needed in order to make these cross-cutting concerns reusable. Those are the reasons why we included these two features in the comparison tables.

The next criterion to take into account is the ability of the approach to express different kinds of relationships. This emerged as a criterion because explicit relationships are more expressive than implicit ones, as if the relationship is explicit it will be interpreted similarly by several people, on the other hand, implicit relationships can be interpreted differently by different people.

The composition mechanism was a really necessary mechanism to have if the approach supports ways to model cross-cutting concerns. As such, it made perfect sense to take it into account as a criterion.

5.2.2 AspectKAOS vs. GORE Approaches

In this section we will compare our approach with other GORE approaches. KAOS is the base of our approach and as such, our approach does take advantage of all the features of KAOS. On the

other hand, four important approaches exist in GORE universe, being i*, GRL, GBRAM and NFR Framework, and as such we find it natural to do a comparison between our approach and those approaches.

Table 5.3 presents the comparison of AspectKAOS against other goal-oriented approaches.

Table 5.3 – Comparison between our approach and GORE approaches

	AspectKAOS	**KAOS**	**I***	**GRL**	**GBRAM**	**NFR Framework**
Obstacles Modeling	Yes	Yes	Not explicitly	Not explicitly	Yes	Not explicitly
Conflict Modeling	No	Yes	Yes	Yes	No	Yes
Tool Support	Yes*	Yes	Yes	Yes	Yes	Yes
Modularization	Yes	Yes	Yes	Yes	Yes	Yes
Cross-cutting Concerns Modeling	Yes	No	No	No	No	No
Explicit Relationships	Yes	No	Yes	Yes	No	Yes
Composition Mechanism	Yes	No	No	No	No	No

* tool to identify aspects.

Being based on KAOS, AspectKAOS do take advantage of all the features supported by KAOS. In this way, modeling obstacles is similar to KAOS, as it possesses an obstacle model just to focus on modeling obstacles [11]. i*, GRL and NFR Framework do not support obstacle modeling directly; the only one which does support obstacle modeling is GBRAM [15].

Although they do not support obstacle modeling they all can model constraints. In i*, GRL and NFR Framework conflict modeling is possible through the use of contribution links, in a graphical and detailed way. In order to model conflicts in GBRAM one must turn to constraints.

When using AspectKAOS and KAOS they both support conflict modeling, as KAOS has that feature and as such AspectKAOS benefices of it. Although in AspectKAOS it is not possible to model conflicts and incorporate aspects into it. This will be suggested as future work. In [12] Lamsweerde, Darimont and Letier suggest an approach on how to model conflicts in requirements engineering guided by goals. So, this could be used in KAOS.

AspectKAOS achieves an improved modularization thanks to the identification and modularization of the cross-cutting concerns, i.e. the aspects. In KAOS it is possible to achieve a good modularization with the help of generic goal models, but it does not completely model cross-cutting concerns. However, by using generic goal models, cross-cutting concerns modeling can be done. Any of the other approaches, i*, GRL, NFR Framework and GBRAM do support modularization. GRL and i* achieve modularization using agents, and NFR Framework provides a catalogue organized like patterns.

Although they support modularization, only AspectKAOS supports modeling of cross-cutting concerns, as it integrates the mechanisms of aspect orientation. Because of this support for aspects we need a way to compose aspects back into the model. As such, throughout the use of roles, that composition is achieved. All the others do not model cross-cutting concerns. It could be achieved, but not directly. As such, they do not have the need to compose cross-cutting concerns back into the model and this means KAOS, i*, GRL, GBRAM and NFR Framework do not have composition mechanisms.

In our approach we added explicit relationships between aspects, and between obstacles and goals. KAOS has implicit or not so explicit relationships. i*, GRL and NFR Framework do have explicit relationships because of the way they define contribution links, which is very detailed. In GBRAM we get explicit relationships only when modeling constraints.

One important aspect of modeling approaches is tool support. AspectKAOS tool support is minimal, as the only tool built was a small application to simplify the process of finding which goals would become aspects. Although major support is given by Objectiver, we cannot use it to model all the components of our approach.

The tool for modeling KAOS is mainly Objectiver which implements all the functionality of KAOS. There are tools for i*, GRL and NFR Framework, but they are not ready for industrial use yet. For i*, GRL and NFR Framework we have OME[1], OpenOME[2] tools.

If we are targeting only i* we can use J-Prim[3] tools. On the other hand, if we only want to use NFR Framework, the tool Softgoal Profile[4] can be used. GBRAM provides also tool support with the tool called GBRAT[5].

5.2.3 AspectKAOS vs. Aspect and Goal Oriented Approaches

In [26] an approach is developed in order to simplify i* methodology. The main difference of our approach from theirs is that in our case the base model does not need to know about the aspect. In their case the aspect is explicitly specified in the refactored model.

In [30] the Aspect-Oriented GRL (AoGRL) approach is presented as a way to manage the complexity of goal models such as GRL. AoGRL is part of AoURn, a unified framework for modeling cross-cutting goals. This is an approach similar to ours that makes use of graphical mechanism for composition and provides tool support. Compositions rules are defined using XML. They are a list of constraint actions and operators, which are used to specify how an aspectual scenario influences or constrains the behaviors of a set of non-functional requirements. The proposed way to relate concerns to requirements is by using a matrix as concerts include constraint requirements. The work also presents a way to do conflict resolution which our work does not. However, our role-based approach provides a more systematic and flexible way for modeling aspects, as while our approach relies the composition on bindings theirs relies on name merging which is more restrictive.

Table 5.4 compares our approach to other approaches incorporating aspects into goal-oriented requirements engineering.

[1] http://www.cs.toronto.edu/km/ome/
[2] http://sourceforge.net/projects/openome
[3] http://www.lsi.upc.es/~ggrau/JPRIM/
[4] http://www.utdallas.edu/~supakkul/tools/softgoal-profile/softgoal-profile.html
[5] This was an online tool that could not be found anymore authored by Anton, Liang and Rodenstein

Table 5.4 – Comparison between our approach and AGORE approaches

	AspectKAOS	I* with Aspects	AoGRL	VGraph
Obstacles Modeling	Yes	Not explicitly	Not explicitly	No
Conflict Modeling	No	Yes	Yes	Yes
Tool Support	Yes*	Yes	Yes	Yes
Modularization	Yes	Yes	Yes	Yes
Cross-cutting Concerns Modeling	Yes	Yes	Yes	Yes
Explicit Relationships	Yes	Yes	Yes	Yes
Composition Mechanism	Yes	Yes	Yes	Yes

* tool to identify aspects.

As i* is the basis for "i* with Aspects", the features available to i* are also available to i* with Aspects. Obstacles are not explicitly modeled. With the use of contribution links, i* models conflicts and as such, with "i* with aspects" it is also possible to model.

In AoGRL obstacles are not modeled explicitly, although it can model conflicts with the use of contribution links. In VGraph obstacle modeling is not supported. In both of these approaches, the use of "Goal Analysis Tool"[6] detects conflicts. On the other hand, AoGRL can take advantage of all the tools stated earlier on section 5.2.2.

As all of these approaches integrate aspects, they do provide modeling of cross-cutting concerns. This feature also helps modularization. In AspectKAOS modularization is improved by the use of aspects. In i* with aspects a way to implement modularization is given in [26]. In [30] an approach is built which allows for modularization support in AoGRL.

[6] http://www.ceismc.gatech.edu/MM_tools/GAT.html

By supporting modeling of cross-cutting concerns, all of these approaches should implement some kind of composition. In fact they do, as there is a real need to tell "how" and "where" cross-cutting concerns affect other concerns in the system.

One feature in KAOS we thought should be improved was the definition of relationships. By being implicit, they depend on user interpretation. By adding explicit relationships between aspects, and obstacles and goals, we avoid being dependent on user interpretation. i* and AoGRL support explicit relationships in contribution links. Explicit relationships are present also in consistency graph and correlations.

5.3 SUMMARY

This was a reworked solution to use KAOS. This solution originally was modeled using UML, which we just “tossed away” and modeled the same system using Aspect KAOS. This was used to demonstrate the advantages of Aspect KAOS over traditional methods as UML.

The composition language was useful because it was both textual and graphical, which made it became simple and non-intrusive.

In order to bind all the roles, in our model, a language would have to be built. Fortunately, as our composition is textual and graphical, the impact of such change was not a problem.

CHAPTER 6

CONCLUSIONS

Here we presented an approach that integrates aspects to KAOS, a well-known goal-oriented approach used in requirements engineering. The idea was to enhance the KAOS goal models by identifying modularizing representing and composing crosscutting goals, which we can aspect goals.

Next we discuss not only the main contributions of the work but also its limitations. Then we suggest some future work and give some final remarks.

6.1 CONTRIBUTIONS

From the existing works aiming at integrating KAOS and aspects, there is still no relevant work. In general, when integrating goals and aspects, it is done in such a way and with a notation that is not as easy as it should be, mainly when it comes to the composition part, which can become complex rapidly.

Our approach provides an extensive graphical guidance and it recurs only to textual constructs when it is really needed. The textual constructs are only needed when binding the roles so it is not a big problem. Actually roles in software engineering are something widely used, and as such, picking the notation and the textual constructs should not be very difficult even for beginners.

Finally, we provided some useful extensions to the original KAOS, by defining more specific kinds of relationships between goals and introducing the interruption goal.

6.2 LIMITATIONS

Actually there is limited tool support for our solution. The only tool that was built only simplifies the building of the aspect table. A tool that could model our entire notation would be helpful. Conflict reasoning considering aspects is another limitation of our work, as it is not currently supported. Also, the composition mechanism may imply that in very complex systems too many bindings will be needed. To solve that we may reuse instantiations kept in an "instantiation library" or "instantiation catalogue".

6.3 FUTURE WORK

One point that should be developed is to extend our approach to other models that are included in KAOS. Therefore, our approach should be extended to object model, operation model and responsibility model. Our approach should also be extended so it would be possible to model conflicts and incorporate aspects into it.

Also, when developing goal models, conflicts do exist. A conflict resolution technique addressing aspect interaction must be developed. Furthermore, it is possible to extend our work to specify product lines where variability could be modeled as aspects.

Another way future work can improve ours, by taking a complete evaluation of the cohesion and coupling of our approach against others. In this sense, it would be necessary to define concrete metrics.

Finally, a more complete tool support could be provided and more real case studies could be applied to the approach for validation purposes.

6.4 FINAL REMARKS

The work presented here is intended to help practitioners with the simplification of KAOS models. This is done with the help of graphical elements, as it is easier both for current practitioners of KAOS as for newcomers to acquire knowledge based in visual elements, than in textual elements.

By using AspectKAOS, the final model is simplified as the density of the goal model is smaller than when using KAOS, as the aspects are incorporated on a different model (aspectual model) separating them from the goal model.

Also, by specifying more specific kinds of relationships between aspects and between obstacles and goals, it helps to clarify how elements interact with each other.

BIBLIOGRAPHY

[1] A. Anton, “Goal-Based Requirements Analysis”, Proceedings of the 2nd IEEE International Conference on Requirements Engineering, April 1996, Colorado Springs, USA, pp. 136-144.

[2] A. Dardenne, A. Lamsweerde, S. Fickas, “Goal-Directed Requirements Acquisition”, Science of Computer Programming, April 1993, Volume 20(1-2), Elsevier North-Holland, Inc, pp. 3-50.

[3] A. Finkelsetin, J. Kramer, B. Nuseibeh, L. Finkelstein, M. Goedicke, “Viewpoints: A Framework for Integrating Multiple Perspectives in System Development", International Journal of Software Engineering and Knowledge Engineering, March 1992, Volume 2(1), World Scientific Publishing Co, pp. 31-58.

[4] A. Lamsweerde, "Divergent Views in Goal-Driven Requirements Engineering", Proceedings of the Workshop on Viewpoints in Software Development, October 1996, San Francisco, USA, pp. 252-256.

[5] A. Lamsweerde, “Elaborating Security Requirements by Construction of Intentional Anti-Models”, Proceedings of the 26th International Conference on Software Engineering, May 2004, Edinburgh, UK, pp. 148-157.

[6] A. Lamsweerde, “Goal-Oriented Requirements Engineering: A Guided Tour”, Proceedings of the 5th IEEE International Symposium on Requirements Engineering, August 2001, Toronto, Canada, pp. 249-263.

[7] A. Lamsweerde, “KAOS Tutorial”, Cediti, September 5, 2003, http://www.objectiver.com/fileadmin/download/documents/KaosTutorial.pdf.

[8] A. Lamsweerde, A. Dardenne, B. Delcourt, F. Dubisy, “The KAOS Project: Knowledge Acquisition in Automated Specification of Software”, Proceedings of AAAI Spring Symposium Series, March 1991, Stanford University, USA, pp. 59-62.

[9] A. Lamsweerde, E. Letier. “From Object Orientation to Goal Orientation: A Paradigm Shift for Requirements Engineering”, Radical Innovations of Software & Systems Engineering, Post-Workshop Proceedings of the Monterey 2002 Workshop, October 2002, Venice, Italy.

[10] A. Lamsweerde, E. Letier, "Handling Obstacles in Goal-Oriented Software Engineering", IEEE Transactions on Software Engineering, October 2000, Volume 26(10), Editora, pp. 978-1005.

[11] A. Lamsweerde, E. Letier, “Integrating Obstacles in Goal-Driven Requirements Engineering”, Proceedings of the 20th International Conference on Software Engineering, April 1998, Kyoto, Japan, pp. 53-62.

[12] A. Lamsweerde, E. Letier, R. Darimont, "Managing Conflicts in Goal-Driven Requirements Engineering", IEEE Transactions on Software Engineering, November 1998, Volume 24(11), pp. 908-926.

[13] A. Rashid, A. Moreira, J. Araújo, “Modularisation and Composition of Aspectual Requirements”, Proceedings of the 2nd International Conference on Aspect-Oriented Software Development, March 2003, Boston, Massachusetts, pp. 11-20.

[14] B. Nixon, "Dealing with Performance Requirements During the Development of Information Systems", Proceedings of IEEE International Symposium on Requirements Engineering, January 1993, San Diego, USA, pp. 42-49.

[15] C. Potts, “Using Schematic Scenarios to Understand User Needs”, Proceedings of the 1st Conference on Designing Interactive Systems, August 1995, Ann Arbor, USA, pp. 247-256.

[16] D. Kim, R. France, S. Ghosh, E. Song, “A UML-Based Metamodeling Language to Specify Design Patterns”, Workshop in Software Model Engineering, October 2003, San Francisco, USA.

[17] D. Stein, S. Hanenberg, R. Unland, "Modeling Pointcuts", Workshop Proceedings of Early Aspects 2004, March 2004, Lancaster, UK, pp. 108-115.

[18] E. Amoroso, "Fundamentals of Computer Security Technology", Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1994.

[19] E. Baniassad and S. Clarke, “Finding Aspects in Requirements with Theme/Doc”, Workshop Proceedings of Early Aspects 2004, March 2004, Lancaster, UK, pp. 16-23.

[20] E. Letier, A. Lamsweerde, "Agent-Based Tactics for Goal-Oriented Requirements Elaboration", Proceedings of the 24th International Conference on Software Engineering, May 2002, Orlando, USA, pp. 83-93.

[21] E. Yu, “Modeling Organizations for Information Systems Requirements Engineering”, Proceedings of the 1st IEEE Symposium on Requirements Engineering, January 1993, San Diego, California, pp. 34-41.

[22] E. Yu, “Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering”, Proceedings of the 3rd IEEE International Symposium on Requirements Engineering, January 1997, Annapolis, USA, pp. 226-235.

[23] E. Yu, J. Mylopoulos, “Understanding “Why” in Software Process Modeling, Analysis and Design”, Proceedings of the 16th International Conference on Software Engineering, May 1994, Sorrento, Italy, pp. 159-168.

[24] F. Alencar, C. Silva, A. Moreira, J. Araújo, J. Castro, “Identifying Candidate Aspects with I-star Approach”, Workshop on Early Aspects in 5th International Conference on Aspect-Oriented Software Development, March 2006, Bonn, Germany.

[25] F. Alencar, C. Silva, A.Moreira, J. Araújo, J. Castro, “Improving the Understandability of I* Models”, 10th International Conference on Enterprise Information Systems, June 2008, Barcelona, Spain.

[26] F. Alencar, J. Castro, A. Moreira, J. Araújo, C. Monteiro, R. Ramos, J. Mylopoulos, "Simplifying i* Models", 17th International Workshop on Agent-Oriented Information Systems, June 2007, Trondheim, Norway.

[27] F. Alencar, J. Castro, A. Moreira, J. Araújo, C. Silva, R. Ramos, J. Mylopoulos, "Integration of Aspects with i* Models", 8th International Bi-Conference Workshop on Agent-Oriented Information Systems, May 2006, Hakotade, Japan.

[28] F. Vieira, I. Sofia Brito, A. Moreira “Using Multi-criteria Analysis to Handle Conflicts During Composition”, Workshop on Early Aspects in 5th International Conference on Aspect-Oriented Software Development, March 2006, Bonn, Germany.

[29] G. Kotonya, I. Sommerville, “Requirements Engineering with Viewpoints”, BCS/IEEE Software Engineering Journal, January 1996, Volume 11(1), pp.5-18.

[30] G. Mussbacher, D. Amyot, J. Araújo, A. Moreira, M. Weiss, "Visualizing Aspect-Oriented Goal Models with AoGRL", 2nd International Workshop on Requirements Engineering Visualization, October 2007, New Delhi, India.

[31] H. Mili, A. Elkharraz, H.Mcheick, "Concerned about Separation", Workshop Proceedings of Early Aspects 2004, March 2004, Lancaster, UK, pp. 76-86.

[32] I. Brito, A. Moreira, "Towards an Integrated Approach for Aspectual Requirements", 14th IEEE International Requirements Engineering Conference, September 2006, Minneapolis, USA, pp. 341-342.

[33] I. Brito, A. Moreira, "Integrating the NFR framework in a RE model", Workshop Proceedings of Early Aspects 2004, March 2004, Lancaster, UK, pp. 28-34.

[34] I. Groher, T. Baumgarth, "Aspect-Oriented from Design to Code", Workshop Proceedings of Early Aspects 2004, March 2004, Lancaster, UK, pp. 63-69.

[35] I. Jacobson, "Use Cases and Aspects – Working Seamlessly Together", Journal of Object Technology, July-August 2003, Volume 2(4), pp. 7-28.

[36] I. Sommerville, "Software Engineering, 8th edition", Pearson Education Limited, Edinburgh Gate, Harlow, England, 2007.

[37] J. Araújo, J. Whittle, K. Kim, "Modeling and Composing Scenario-Based Requirements with Aspects", The 12th IEEE International Requirements Engineering Conference, September 2004, Kyoto, Japan.

[38] J. Mylopoulos, L. Chung, B. Nixon, "Representing and Using Non-Functional Requirements: A Process-Oriented Approach". IEEE Transactions on Software Engineering, June 1992, Volume 18(6), pp. 483-497.

[39] J. Warmer, A. Kleppe, "The Object Contraint Language: Precise Modeling with UML", Addison-Wesley, Edinburgh Gate, Harlow, England, 1999.

[40] J. Whittle, J. Araújo, “Scenario Modelling with Aspects”, IEE Proc. Softw., Volume 151(4), August 2004.

[41] K. Graversen, K. Østerbye, “Implementation of a Role Language for Object-Specific Dynamic Separation of Concerns”, 2nd International Conference on Aspect-Oriented Software Development, March 2003, Boston, Massachusetts.

[42] L. Liu, E. Yu, “From Requirements to Architectural Design - Using Goals and Scenarios”. ICSE-2001 Workshop: From Software Requirements to Architectures, May 2001, Toronto, Canada.

[43] L. Liu, E. Yu, J. Mylopoulos, “Security and Privacy Requirements Analysis within a Social Setting”. Proceedings of the 11th International Conference on Requirements Engineering, September 2003, Monterey, USA, pp. 151-161.

[44] M. Jackson, "Problems, Subproblems and Concerns", Workshop Proceedings of Early Aspects 2004, March 2004, Lancaster, UK, pp. 24-28.

[45] Object Management Group, “Unified Modeling Language Specification version 2.1.2”, Infrastructure specification, November 2007, available at http://www.omg.org/spec/UML/2.1.2/ (accessed June 2008).

[46] P. Zave, “Classification of Research Efforts in Requirements Engineering”, ACM Computing Surveys, December 1997, Volume 29(4), pp. 315-321.

[47] R. Darimont, A. Lamsweerde, "Formal Refinement Patterns for Goal-Driven Requirements Elaboration", Proceedings of the 4th Symposium on the Foundations of Software Engineering, October 1996, San Francisco, USA, pp. 179-190.

[48] R. France, D. Kim, S. Ghosh, E. Song, “A UML Based Pattern Specification Technique”, IEEE Transactions on Software Engineering, March 2004, Volume 30(3), pp. 193-206.

[49] R. Laney, L. Barroca, M. Jackson, B. Nuseibeh, "Composing Requirements Using Problem Frames", Proceedings of the 12th IEEE International Requirements Engineering Conference, September 2004, Kyoto, Japan, pp. 122-131.

[50] R. Thayer, M. Dorfman, “Software Requirements Engineering, Second Edition”, Los Alamitos, Calif.: IEEE Computer Society Press, 1977.

[51] S. Somé, "An Environment for Use Cases based Requirements Engineering", Proceedings of the 12th IEEE International Conference on Requirements Engineering, September 2004, Kyoto, Japan, pp. 364-365.

[52] V. Santander, J. Castro, “Deriving Use Cases from Organizational Modeling”, Proceedings of IEEE Joint International Conference on Requirements Engineering, September 2002, Essen, Germany, pp.32-39.