Using i* to identify candidate aspects

Eva Spies[†], Julia Rüger[†] and Ana Moreira[‡]
[†] Universität Koblenz-Landau, Postfach 201 602, 56016 Koblenz GERMANY
[‡]Dept. Informática, FCT, Universidade Nova de Lisboa, 2829-516 Caparica, PORTUGAL
{evaspies, jrueger}@uni-koblenz.de, amm@di.fct.unl.pt

Abstract

Aspect-orientation has been capturing researcher's attention for the last few years. We have seen the birth of several aspect-oriented programming techniques and also the propagation of the aspect concept to the earlier stages of the software development process, such as requirements analysis and design. Our long-term goal is to extend aspect-orientation to the business modelling activity. This paper discusses our first results by using the i*[1] technique to assist us with the concern elicitation process.

1. Introduction

The main idea of separation of concerns is to focus ones attention only on one certain issue at a time. Separation of concerns aims at identifying and modularizing those parts of software that are relevant to a particular concept, goal or purpose. Traditional approaches to software development, such as object-oriented and structured methods, have been created with this principle in mind. However, they are unable to handle broadly scoped requirements and constraints, also known as non-functional requirements. Non-functional requirements are global properties of a system and usually refer to quality of service. Recent approaches achieve separation of concerns by integrating functional and non-functional requirements [Dardenne 1993, Yu 1995a]. Nevertheless, they do not consider the crosscutting nature of some of those concerns.

Examples of crosscutting concerns are security, fault tolerance and usability. The main goal of aspect-orientation is to promote modularization by offering mechanisms that permit the encapsulation of crosscutting concerns in separate modules, known as *aspects*. Aspects make the specifications and the code more general, so that they can be reused in several other cases. Aspect-oriented software development (AOSD) aims at providing means for their systematic identification, separation, representation and composition [Elrad 2001, AOSD].

For the last few years we have seen the appearance of several aspect-oriented requirements analysis and design approaches. Our paper focuses on aspects on the early-requirements activity (as opposed to late-requirements, according to the common requirements engineering classification activities [Mylopoulos 1999]). One of the problems pointed to the Early Aspects (www.early-aspect.net) approaches (e.g. [Moreira 2002, Rashid 2003, Brito 2004, Baniassad 2004]) is the lack of an elicitation process, since are late-requirements techniques. Our work is a first step towards solving this problem. We based ourselves on the early-requirements technique i* [Yu 1995], as a guide to the identification of the main candidate aspects and to integrate the results within the requirements engineering model proposed in [Brito 2004]. The goal is to propose a set of guidelines to help identifying concerns and describe each one using the template proposed in [Brito 2004].

This paper is organized as follows. Section 2 sets the foundation for this paper, by discussing some work on early aspects and giving an overview of i*. So that the main section of this paper, Section 4, can be better understood, we first apply i* to a case study, in Section 3. By doing so the reader can have a better feeling of the main models proposed by i*, from where our guidelines to derive concerns, and ultimately candidate aspects, will be extracted in Section 4. Finally, Section 5 concludes this paper and points directions for further work.

2. Background

2.1 Early aspects

There are three main steps in the aspect-oriented software development. First, we need to identify crosscutting concerns, and therefore be able to structure the requirements by decomposing the problem into concerns. Next, we need to represent (specify or implement, depending on the level of abstraction) each concern. Finally, all the concerns, crosscutting and non-crosscutting, need to be composed to obtain the final system. The composition process is also known as *weaving*. The composition is guided through composition rules. A composition rule defines the way, in which a crosscutting concern affects other concerns. The composition rule can appear inside the crosscutting concern, specifying how other concerns are affected by this crosscutting concern (just like what happens in AspectJ); it can appear inside a non-crosscutting concern, specifying the way crosscutting concerns affect that concern; finally, it can appear in a separated module.

Our paper focuses on aspects at the early stage of the software development life cycle. We call them "candidate aspects", as at this early stage we still do not know if they will be handled as aspects during later stages of the software development process [Rashid 2003].

Our work is based on the results presented in [Brito 2004], in particular a template they propose to define concerns, crosscutting or non-crosscutting. The template is presented in Table 1.

Table 1: Template to describe a concern

Name	The name of the concern.
Source	Source of information, e.g. stakeholders, documents, domain, catalogues and business process.

[1] Should be read "eye-star".

Stakeholders	User that needs the concern in order to accomplish their job.
Description	Short description of the intended behaviour of the concern.
List of Responsibilities	
R_i	List of what the concern must perform; knowledge or proprieties the concern must offer.
List of Contributions	
C_i	List of concerns that contribute or affect this concern. This contribution can be positive (+) or negative (-)
List of Priorities	
Stakeholder $_i$	Expresses the importance of the concern for a given stakeholders. It can take the values: Very Important, Important, Medium, Low and Very Low.
List of required concerns	
RC_i	List of concerns needed or requested by the concern being described

A template needs to be filled in for each concern. In order to accomplish this they need to identify concerns, specify concerns and identify crosscutting concerns. In the first task the rows name, source, stakeholders and description can be completed. The second task, specifying concerns, is divided into: applying the approach that better specifies each concern (classification row) and identifying contributions between concerns so that conflicts can be detected. Conflicts detected are solved by attributing priorities to conflicting concerns (priority row). The last task composes concerns by first identifying those that are crosscutting, which helps filling in the required concerns and description rows. A concern is crosscutting if it is needed by two or more concerns.

2.1 i* framework

Based on the NFR-Framework [Chung 2000], i* provides understanding of the reasons ("why") which underlie system requirements, focused on strategic actor relationships [Yu 1995a]. It is visualized in two main models, the Strategic Dependency Model (SDM) and the Strategic Rational Model (SRM). The SDM gives an overview of the systems environment meanwhile the SRM illustrates the internal behaviour of the actors.

Strategic Dependency Model

This model presents the systems' goals and what has to be done to reach them. It captures the motivation and the rationality of activities, which are carried out by the main actors of the system. The main elements are Actors, Dependencies and Strength.

Actors are active entities of the system that have to interact with each other to make the system work (see Figure 1). Yu states that actors can be differentiated into three items, the sub-units of a complex actor [Yu, 1997a]. Those are role, agent and position. An agent represents a person or a system; it is the most important type of actor and therefore the only one used in this paper.

Fig. 1: Notation for actor

Dependencies are defined among actors and are the most important issues in the SDM [Yu 1995a]. They are intentional relationships that deal with desires, commitments and expectations between actors. Those dependencies are used to differentiate among four kinds of relationships (see Figure 2): goal-dependency, task-dependency, resource-dependency and softgoal-dependency. In a dependency, one actor, the depender, depends on another actor, the dependee, for a certain concern, the dependum (a goal, a task, a resource or a softgoal).

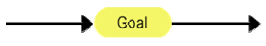

Fig 2.a: Goal-dependency

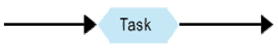

Fig. 2.b: Task-dependency

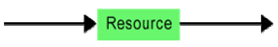

Fig. 2.c: Resource-dependency

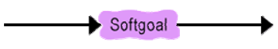

Fig. 2.d: Softgoal-dependency

In a goal-dependency (Figure 2.a), the dependum is expressed as an assertional statement. The dependee has to do whatever is necessary and possible, to achieve the goal however he is free how he will achieve it. On the one hand, a depender does not care how the dependee solves the goal he also does not have to have knowledge to achieve this goal. It is the outcome that matters. On the other hand, a depender is dependent if the dependee fails.

In a task-dependency (Figure 2.b), the dependum is an activity that has to be carried out. A task-dependency specifies how the dependee should perform the task, but not why. In a SDM there are no steps shown which are required to perform the task. The depender has control over how the task is performed; he is able to have a task performed without engaging personally, but is vulnerable if the task fails. Even though the dependee is controlled, he still has its freedom of action within this constrains.

In a resource-dependency (Figure 2.c) the depender depends for an availability of an entity from the dependee. This entity could be physical or informational. There are no decisions, or issues to be addressed; it is the resource of a deliberation action process. The resource can be used by the depender.

Finally, a softgoal-dependency (Figure 2.d) is associated with the notion of non-functional requirements. The functionality is similar to the goal-dependency. The dependee should perform a task which encounters a softgoal. Different to goals is that the conditions to be attained are elaborated as the task is performed and that there are no clear-cut criteria for their satisfaction. The depender makes the last decision, with the benefit of the know how of the dependee. A Softgoal allows the SDM to deal with many usual informal concepts.

The dependencies indicate the control in the relationship between two actors regarding the dependum. They characterize how decisions fall on either side of the dependency, and which side will handle problems if they arise.

The *strength* is an addition to give a dependency an importance rank. It can be marked independently on both sides of the actors with a symbol. There are three different kinds of strength: open (uncommitted), committed and critical. (See Figure 3 for an example.)

Strength "open" uses the symbol "O"; if this sign appears on the side of the depender, it means that if the dependency fails the depender is affected but not too badly; on the other hand, if it appears on the side of the dependee it signifies that the dependee is able to achieve the goal/perform the task/furnish the resource, but he has no commitment with this relationship.

Strength "committed" has no special symbol associated. This means that whenever there is no strength marked the dependency is committed. If there is no symbol on the dependers' side it means that if the dependency fails the depender is affected badly. No symbol on the dependees' side signifies that the dependee will do his best to gain the goal/task/resource.

Strength "critical" has the symbol "X"; it is marked on the side of the depender to signify that a goal of the depender could not be achieved if the dependency fails. At the side of the dependee the symbol means that there is a need of guarantee the success of goal/task/resource to have a critical dependency.

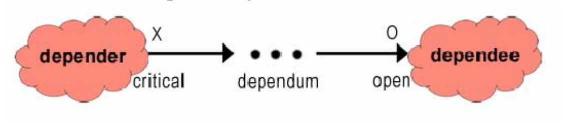

Fig. 3: Example of a dependency with two strengths

Strategic Rational Model

The SRM provides a way of modelling stakeholder's interests and how they might be met. At the first look, this model is similar to the SDM, since it also contains dependencies (goal-, task-, resource-, softgoal-dependency). The main difference is that there is no overview of the systems' environment, instead, it is a closer view at the important actors of the future system. The SRM is at a more detailed level than the SDM. It shows what happens "inside" the actors, to model internal intentional relationships. This means that one needs to create a separate SRM for each actor of the system. Some dependencies inside the Strategic Rational Model are connected to external dependencies of the SDM. Its main elements are dependencies, task-decomposition links, means-end links and contributions.

Instead of defining dependencies between actors, as in the SDM, the SRM links the dependencies into a tree. The tree of dependencies shows what the particular actor does by himself. *Task-decomposition links* and *means-end links* hold the dependencies together (see Figure 3).

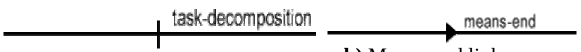

Fig. 4. a) Task-decomposition link **b)** Means-end link

A task-decomposition link (Figure 4.a) shows that a task is divided into sub-elements. This leads to a hierarchy that describes what should be done to carry out a certain task. Only if all the sub-elements are achievable the mother-element is achievable, therefore this is an AND relationship.

Means-ends links (Figure 4.b) are links between internal relationships of intentional elements (goal, task, resource, softgoal), but only inside the SRM; they don't exist in the SDM. They should help to understand the "why's" you could ask for in some tasks/pursue a goal/need a resource/want a softgoal. Means-end relationships suggest that there can be other means for achieving the same goal. This means that we can handle those relationships as OR relationships. "The means-end links for softgoals, however, require more differentiation because there can be various types of contributions leading to a judgment of whether the softgoal is sufficiently met" [Yu, 1997].

In addition, means-end links have *contributions*, which give assessments to the dependencies. They can be positive (help, make, some+) or negative (hurt, break, some-). According to Yu, "the SRM, though conducive to systematic reasoning and decision making, still relies on human designers to make decisions and judgements" [Yu, 1995b].

3. Applying i* to a case study

The example we have chosen to illustrate the use of i* and to explore the connection of this framework with aspects is an integrated car key system that is able to help its owner to find his/her car in a car parking. We call this key YkeyK (meaning "Your Key Knows").

3.1 YkeyK requirements

A YkeyK is probably what we all wished to have any time we forgot where we parked our car. Imagine you park your car in a multi-storey car parking, and, after a few hours you forget where you left your car. Wouldn't it be nice if somebody or some device would show you the way? YkeyK is that device. It is the key of the car equipment with special functionalities that leads you straight through the shortest and quickest way to the parking space where your car is.

There are requirements to the car and the car park, which are needed to make the guiding system work. First, the car has to be equipped with a "car location system"[2]. Second, the car park has to cooperate with the "car location system" by having the necessary equipment and data for the search. This data consists of information about the topology and environment, the pricelist (cost per hour) as well as the entry time (hour and minute) of the car.

Figures 5, 6 and 7 give a general view of the whole context in which the YkeyK functions.

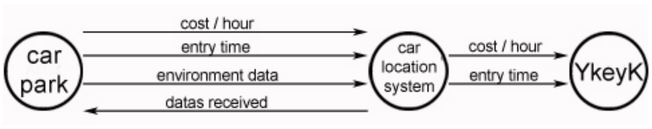

Fig. 5: Data transfer at the entry gate

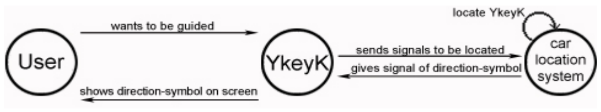
User
wants to be guided
shows direction-symbol on screen
YkeyK
sends signals to be located
gives signal of direction-symbol
locate YkeyK
car location system

Fig. 6: Search mode

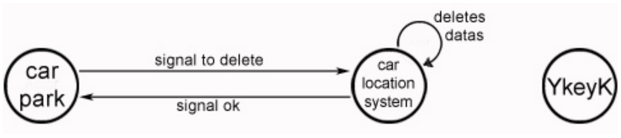

Fig. 7: Data transfer at the exit gate

Figure 5 illustrates what happens when the driver is standing with his/her car in front of the barrier next to the entrance machine of the car park. This equipment has to send the information in the database to the car location

[2] The "car location system" is an imaginary system, which has to be able to cooperate with the YkeyK and is supposed to locate it. It is not our main system; therefore we will not specify it.

system, which acknowledges the transfer. After this, the machine produces a ticket that the user has to collect so the barrier opens.

The YkeyK can be in two modes, the stop mode and the search mode. To change between those modes, the user has to press the button “search / stop”. In stop-mode the YkeyK has no signal connection to the “car location system“, it only shows the current time and date on the screen. To get into search-mode (see Figure 6) the user presses the button on his key, when s/he wants to be guided. The YkeyK sends a request to the car location system initiating the search. This request is received by the car location system which starts to locate the YkeyK and sends back signals of direction-symbols. The YkeyK transforms those signals into direction symbols and shows them on the screen for the user to read. Those symbols are “go up”, “go down”, “go straight”, “go right”, “go left” and “go back”. During search-mode, the YkeyK has always to send signals, so the car location system can locate it and correct the direction. If the user doesn’t follow the shown direction the “car location system” has to react quickly and generate a new way.

Figure 7 describes what happens when the driver wants to drive out of the parking and is standing in front of the barrier, next to the exit machine. The user has to put her/his paid ticket into the machine, which requests the car locations system to delete the database. (To pay the ticket, there is another machine, similar to those in any car park.) After deleting the database, the car location system acknowledges the cancellation and the barrier opens.

3.2 Building the SDM and the SRM

Figure 8 depicts the SDM for the YkeyK whole operational environment. At this level of abstraction, the SDM is composed of a set of actors and the dependencies between those actors. One actor (the YkeyK) is our system while the other represents the stakeholders (people or other systems) that the YkeyK needs to interact with. As explained in Section 2.2, the dependencies relate actors through goals, tasks, resources or softgoals. In particular, here we present the systems’ goals and what has to be done to reach them. The SDM captures the motivation and the rationality of activities, which are carried out by the main actors of the system.

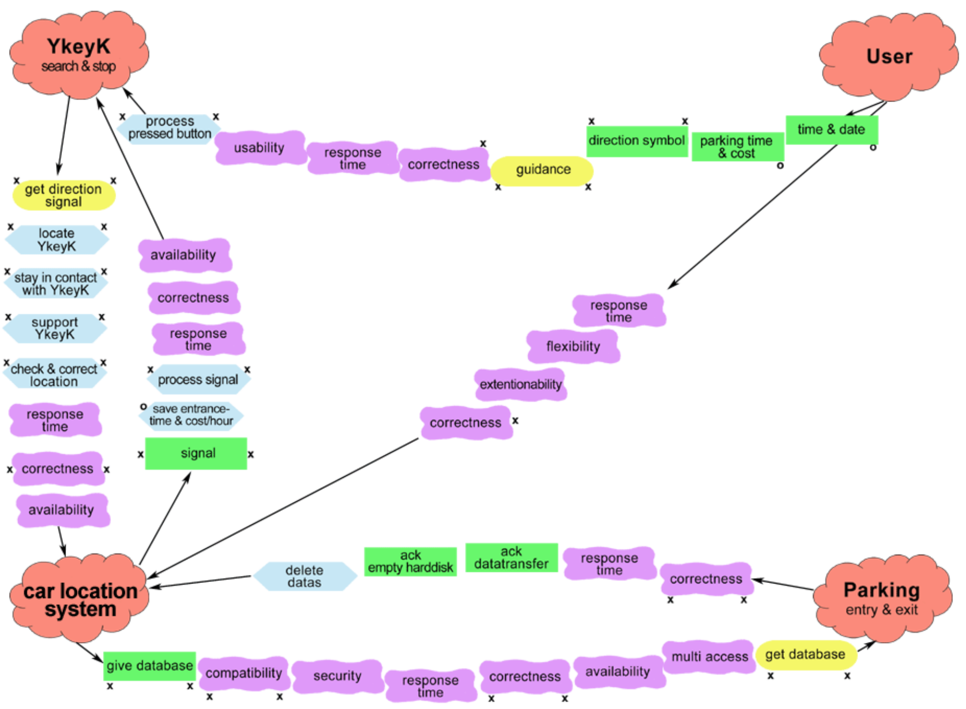

Fig. 8: SDM for the YKeyK system

In our case the actors are “User”, as a human being, “YkeyK”, our future system, “car location system”, the equipped car and “Parking”, as a fixed well-equipped setting. The main part of the system is the YkeyK therefore we need to go deeper into this actor and see what are its main actors and dependencies. When trying to build the SRM we realized that this was still a complex system. For this reason we decided to build a lower level SDM for that actor (see Figure 9). To reach the users’ goal, to be guided, we identified four actors “stop mode”, “search mode”, “clock” and “display”. In “stop mode” the YkeyK does not send signals but it can receive them. The main mode is probably “search mode”. Here the sending, receiving and also processing, of signals take place. The “clock”, in a deeper level SDM, generates the cost the parking time. On the “display” the generated data, such as time, date, direction symbols, parking time and cost, is presented to the user. In this circle of actors, all the interrelationships of the SDM from the upper level (see Figure 8) are contained.

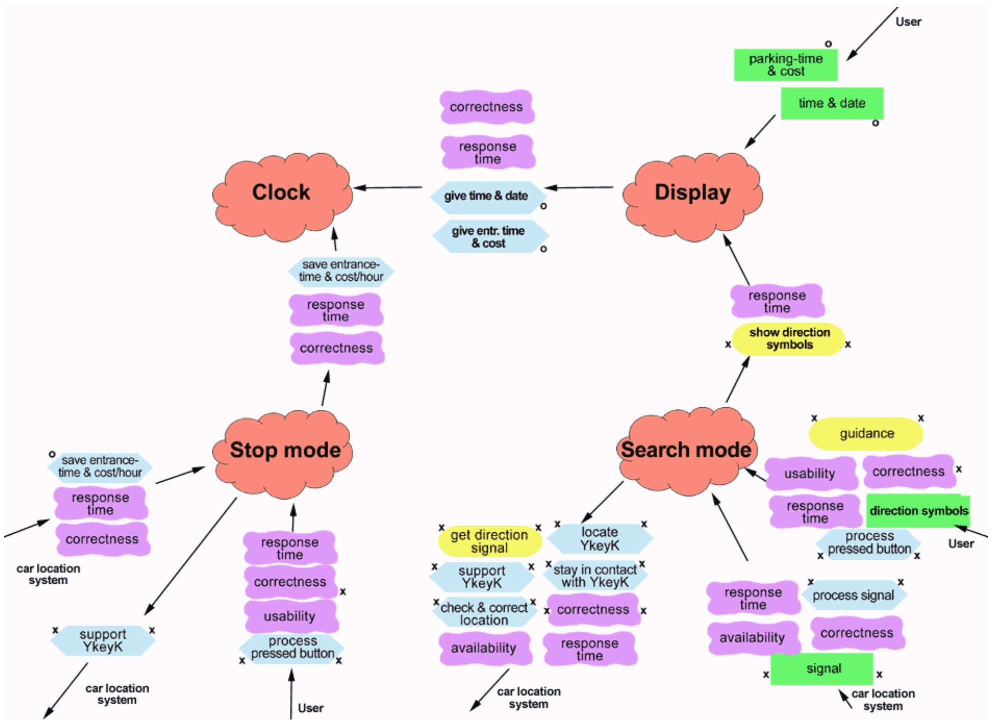

Fig. 9: SDM for the YKeyK actor

Building lower level abstraction SDMs can be thought of as building lower level abstraction data flow diagrams; this means that we need to guarantee consistency between levels of abstraction, by considering that all the dependencies "entering" or "leaving" an actor need to be handled in the SDM that describes that actor. So, the new actors have interrelationships with the actors of the global SDM.Figure 10 illustrates the strategic rational model for our case study. In this paper, we only show the SRM to the actor "search mode", as it is the most important element in our system. This SRM is the model represented inside the dotted circle. The dependencies around this circle have been generated in strategic dependency model.

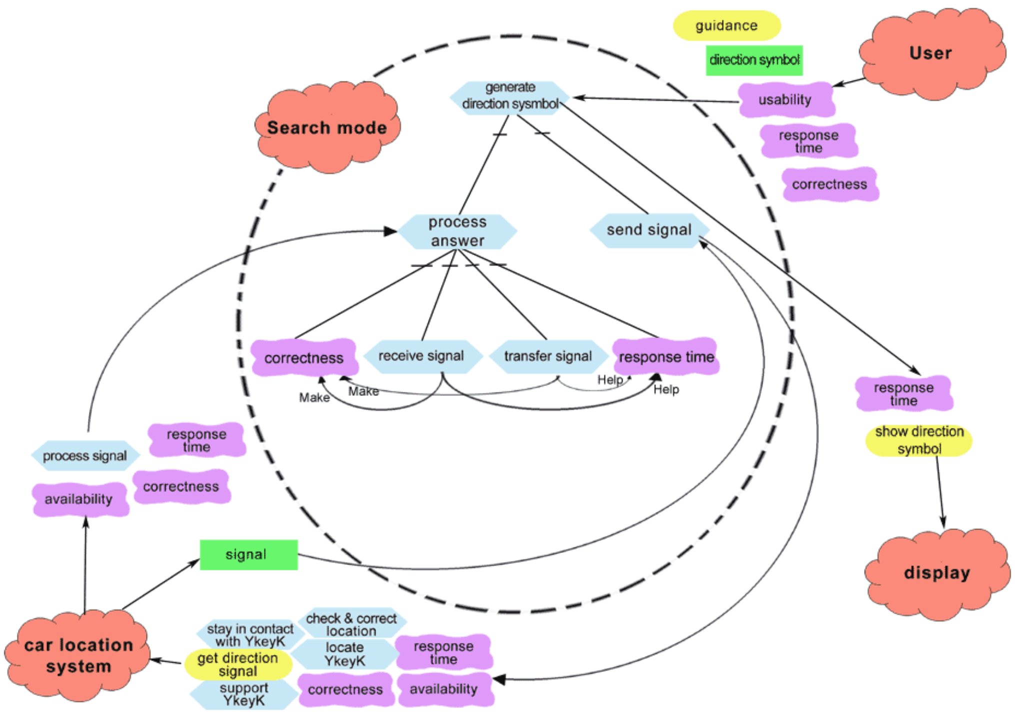

Fig. 10: SRM for Search Model

4. Deriving candidate aspects from i*

In this section we show how we can relate the SDM and SRM elements to identify concerns and, from there, candidate aspects. Based on [Brito 2004] we will propose a set of guidelines to help completing a template out of a SDM and a SRM. Finally, we will apply these guidelines to the YKeyK case study.

4.1 Guidelines

In order to gather all the information needed for the template we need to:

- Build a template for each dependency. This means that a dependency is a candidate concern.
- Define a mapping for getting from the SDM and SRM the information for the template.
- Propose a set of keywords to be used in order to simplify and to disambiguate the description of dependencies.
- Propose a mapping between i* contributions and template's contributions.
- Propose a mapping between i* strengths and the template's priorities.

Each of these points is explained as we discuss the various entries in the template. The *Name* row is filled with the name of a dependency. The *Source* row describes where the information of the dependency comes from. Those sources can be stakeholders, documents, a domain, catalogues or a business process. The *Stakeholders* row can be read out of the models.

In SDM the stakeholders are the two actors connected by the dependency under study. In case of the SRM there exists only one stakeholder, which is the actor being described by the model.

The *Description* row describes the dependency in SDM one has to use certain keywords to identify the dependency, which may be a goal, a softgoal, a task or a resource. For a goal, the depender wants the dependee to achieve "the goal" for it. In a task dependency, the dependee has to perform "the task" for the depender. In a resource dependency, the depender asks for "the resource" to receive from the dependee. The depender wishes "the softgoal" for the dependency performed by the dependee.

In a SRM, there is only one actor, the depender, not the dependee. So we need different keywords: for the goal, the actor wants to achieve the goal. The actor has to perform "the task". For the resource dependency, the actor has to offer "the resource". For the softgoal, the actor should perform "the softgoal". The degree of strength of "should" will vary according to the contributions.

The *List of Responsibilities* row is a little more difficult to fill, since there is no accurate rule to read it from an SDM model. Here we have to think about which other dependencies are related to this recent one. Those are the dependencies which the current dependency has to perform; this could be knowledge or proprieties that the dependency must offer.

The *List of Contributions* row can only be filled based on the information on the SRM. The template proposes that a positive (+) or negative (-) contribution should be specified. In the SRM contributions are "help", "make", "some+", "hurt", "break" and "some-". We map "help", "make" and "some+" to positive (+). The others are mapped to negative (-). If a SRM does not exist, and the only available information is that of a SDM the contribution should take the value "<none>".

The *List of Priorities* row can be easily filled in based on the information available in an SDM. Each dependency shows how important it is, by means of the strength symbol. The position of the symbol indicates to which actor the priority refers to. The template provides five kinds of priority, while the model uses only three. "X" stands for "very important", no symbol (committed) refers to "important" and "medium", and "O" stands for "low" and "very low".

The *List of Required concerns* row (notice that in this case, the concerns are dependencies) requires a little more work. If the dependency under study is a softgoal, this row cannot be filled. So one only writes "<none>". For the other dependencies one has to find the matching softgoals, which have to be found by closely inspecting the requirements of each dependency.

4.2 Applying the guidelines to the case study

Applying the rules and mappings discussed in the previous subsection to our case study helps us identifying a reasonable set of concerns. Tables 2 and 3 show two templates completed based on the information available on the SDMs and SRMs built for our case study. While Table 2 describes the template for the goal "guidance", Table 3 describes the template for the "response time" softgoal.

Table 3: Template for Guidance goal-dependency

Name	Guidance
Source	Stakeholders
Stakeholders	User, YkeyK
Description	The User wants the YkeyK to achieve "guidance".
List of Responsibilities	
R1	User presses button to be guided.
R2	YkeyK shows direction symbols on the screen.
List of Contributions	
<none>	
List of Priorities	
User	Very important
YkeyK	Very important
List of Required Dependencies	
RD1	Correctness
RD2	Response time
RD3	Usability

Table 4: Template for Response Time softgoal-dependency

Name	Response time
Source	Stakeholders, catalogues
Stakeholders	Search mode
Description	The search mode should perform "response time".
List of Responsibilities	
R1	The search mode should react in time when it transfers signal.
R2	The search mode should react in time when it receives signal.
R3	The search mode should react in

	time when it processes answer.
R4	The search mode should react in time when it generates direction symbol.
List of Contributions	
correctness	-
List of Priorities	
<none>	
List of Required Dependencies	
<none>	

Given that the resulting set of concerns was obtained based on the results of a business analysis, we have a comfortable starting point to now start our late-requirements analysis.

As the development process progresses, new concerns will be identified, not only during requirements engineering, but also during architecture modelling and design.

The candidate aspects are the concerns that cut across more than one concerns, i.e. those that are required by several other concerns. By analyzing all the templates obtained for our system, we found the following crosscutting concerns: Response Time, Usability, Correctness and Availability.

5. Conclusion and future work

One of the problems that are pointed to the existing aspect-oriented requirements engineering approaches is the difficulty on identifying concerns, since those approaches lack an elicitation process.

Our work is a first step towards solving this problem. We based ourselves on the i*, a business modelling technique, developed by Yu at University of Toronto [Yu 1995a]. The main result of our work is to offer a list of the candidate aspects each one specified according to the template proposed in [Brito 2004]].

The paper started with an overview on aspect-orientation and then presented an introduction to business modelling through i*. We discussed some guidelines and mappings that should be followed to describe a concern using the template, and followed by applying those to a case study.

Our plan for the future is to formally define the mappings and rules, test them with more cases studies and to integrate our views further with other aspect-oriented requirements engineering approaches. On a complementary line of work, we are also aiming at extending business modeling with aspects. All softgoals from one kind, for example correctness, are able to cut across each other. We need to investigate this further, as well as the interactions between a softgoal and other kinds of softgoals.

References

[AOSD] www.aosd.net

[Baniassad 2004] E. Baniassad and S. Clarke. Theme: An approach for aspect-oriented analysis and design. In 26th Int'l Conf. Software Engineering (ICSE), (Edinburgh, Scotland). IEEE (To Appear).

[Brito 2004] I. Brito, A. Moreira. Integrating the NFR framework in a RE model, Workshop on Early-Aspects at the Third International Conference on Aspect-Oriented Software Development, Lancaster, UK, http://trese.cs.utwente.nl/workshops/early-aspects-2004/workshop_papers.htm, 2004.

[Chung 2000] L. Chung, B.A. Nixon, E. Yu and J. Mylopoulos. Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers, 2000.

[Dardenne 1993] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. Science of Computer Programming, 20(1-2):3–50, 1993.

[Elrad 2001] T. Elrad, R. E. Filman, A. Bader, Guest Editors. Aspect-oriented programming. High-Performance Java Communications of the ACM Aspect-oriented Programming, October 2001-Volume 44 Number 10

[Moreira 2002] A. Moreira, J. Araújo, I. Brito. "Crosscutting Quality Attributes for Requirements Engineering", Software Engineering and Knowledge Engineering Conference (SEKE), Ischia, Italy, 15-19 July 2002.

[Mylopoulos 1999] J. Mylopoulos. *Requirements-Driven Information System Engineering.* Workshop on Agent-Based Information Systems. CAiSE'99. Heidelberg, Germany 1999.

[Rashid 2003] A. Rashid, A. Moreira, J. Araújo. "Modularisation and Composition of Aspectual Requirements", AOSD 2003, Boston, USA, 17-21 March, 2003.

[Yu 1995a] E. Yu. Modelling Strategic Relationships for Process Reengineering, Ph.D. thesis, also Tech. Report DKBS-TR-94-6, Dept. of Computer Science, University of Toronto, 1995.

[Yu 1995b] E. Yu. Models of Supporting the Redesign of Organizational, Work. Proc. Conf. on Organizational Computing Systems (COOCS'95), Milpitas, California, August 1995, pp. 225-236.

[Yu 1997] E. Yu. Strategic Modelling for Enterprise Integration, University of Toronto web page.