Reusing Non-functional Patterns in i* Modeling

Herbet Cunha, Julio Cesar Sampaio do Prado Leite
Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro
Rio de Janeiro, Brasil
hcunha@inf.puc-rio.br, www.inf.puc-rio.br/~julio

***Abstract*—Patterns are an important concept for capturing and reusing Non-Functional Requirements (NFR) knowledge. This paper reports on the use of NFR patterns as a means for reusing cataloged information. In particular we stress its use for intentional models written in i*. This paper focuses on the NFR awareness and details some of its patterns. The integration with i* is done by means of specification patterns, which integrates the cataloged information with the i* model. An example is shown as to argue about the feasibility of our proposal.**

***Index Terms*— NFR, Awareness, Patterns, Requirements Reuse.**

I. Introduction

Awareness is a fundamental requirement for software that needs to adapt itself to some degree. Self-adapting provides to software the ability to deal with changes in the environment in which the software is inserted. The requirement of awareness, in its turn, provides the software the abilities to perceive what is happening in the environment, and "understand" how the environment changes, and how changes in the environment affect its proper functioning.

According to IBM [1], autonomous systems have the ability to manage themselves and perform their tasks through the appropriate choice of actions to be taken according to one or more situations that they perceive in the environment. They are systems that perceive their operating environment, model their behavior in that environment, and take action to change the environment or their behavior.

Several researchers consider the terms autonomous (autonomic computing) and self-adaptive as synonyms [2]. Despite the similarity, Salehie and Tahvildari highlight some differences and consider that the autonomous term refers to a broader context, the handling of all layers of the system architecture (from applications to hardware), while self-adaptation has less coverage - mainly restricted to applications and middleware - and thus falling under the umbrella of autonomic computing. Furthermore, the research on autonomic computing focuses on architecture solutions for automating maintenance tasks while self-adaptive systems generally are more concerned with user requirements, constraints and assumptions of the environment. In this sense, the research on self-adaptive systems has a broader reach, considering the whole process of software development from requirements to operation.

In general, the approach to operationalization, both self-adaptive systems as autonomous systems, goes through some mechanism for monitoring and analyzing the context in which the software operates and to which it must adapt.

Surveying the literature Cunha [3] proposed a Software Interdependence Graph [4] for awareness (Figure 1) which will be the base for cataloged information, which will be detailed by a series of NFR-Patterns [5]. Models in i* will reuse these patterns.

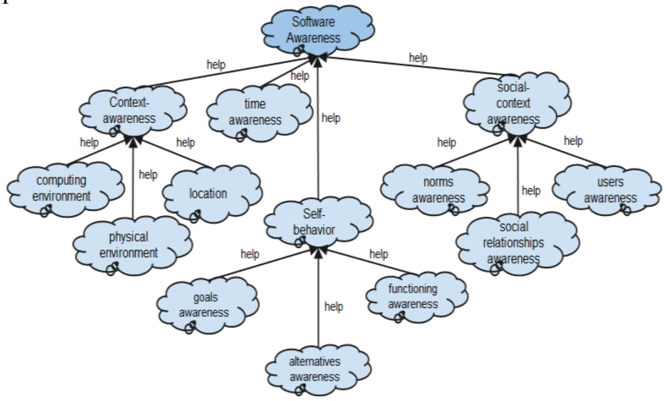


Figure 1 - The Awareness SIG

The paper is organized as follows: an introduction to NFR-patterns, an introduction to i*, a description of how to add awareness abstractions to requirements models, considerations on modeling the awareness requirement, and conclusions.

II. The NFR Patterns Approach

The NFR patterns approach [5] provides a solid basis for treating non-functional requirements in a systematic way, with appropriate representation schemas and rules.

Patterns represent solutions to generic problems, which can be applied to several scenarios. Patterns became popular among software developers when Gamma et al. [6] created the Design Patterns. Many other patterns have followed, a number of which tackled the requirements engineering process in general [7, 8] and NFRs in particular [5, 9, 10].

The NFR pattern approach [5] focuses on the reuse of NFR knowledge [4]. NFR patterns may be specialized to create more specific patterns, composed to build larger patterns, or instantiated to create occurrence patterns using existing patterns as templates. NFR Patterns can be seen as an evolution of the NFR catalogs. Supakkul *et. al* [5] defined four types of patterns for capturing and reusing knowledge of NFRs: (i) Objective Patterns are used to capture the definition of NFRs in terms of specific (soft)goals to be achieved; (ii) Problem Patterns capture knowledge of problems or obstacles to achieving goals; (iii) Alternatives Patterns are used to capture different means, solutions, and requirements mappings; (iv) Selection Patterns allow to choose the best alternative considering their side-effects. In [9], Serrano and Leite presented another type of pattern: the Questions Patterns which helps the refinement of sofgoals towards operationalizations via Alternative Patterns.

Figure 2 and Figure 3 shows two NFR patterns. One is a Question Pattern (Figure 2) and the other is an Alternative Pattern (Figure 3). Those will be reused in building i* models with awareness. In the Question Pattern, some questions were elicited to each type of context-awareness. Answers to these questions will lead to possible operationalization to the NFR, while operationalization alternatives represent ways to *satisfice* the NFR. The answers will guide engineers to choose the most appropriated alternative in each case.

III. i* Framework

In our approach we use the i* (i-star) framework for modelling requirements. i * [11] is well suited for modelling the goal-oriented requirements and also to build agent-oriented models. In i*, the system decomposition is made through actors which can be specialized in software agents. Besides the intentional dependency relationships among actors/agents, an internal model of each actor/agent is drawn from their individual goals, with the possible alternatives to achieving these goals. Moreover, in i* is possible to represent non-functional requirements as first class elements. As such, i* uses the softgoal element and reasons if an acceptability threshold is met. Reasoning about levels of satisficing, makes it possible to consider a choice among alternatives.

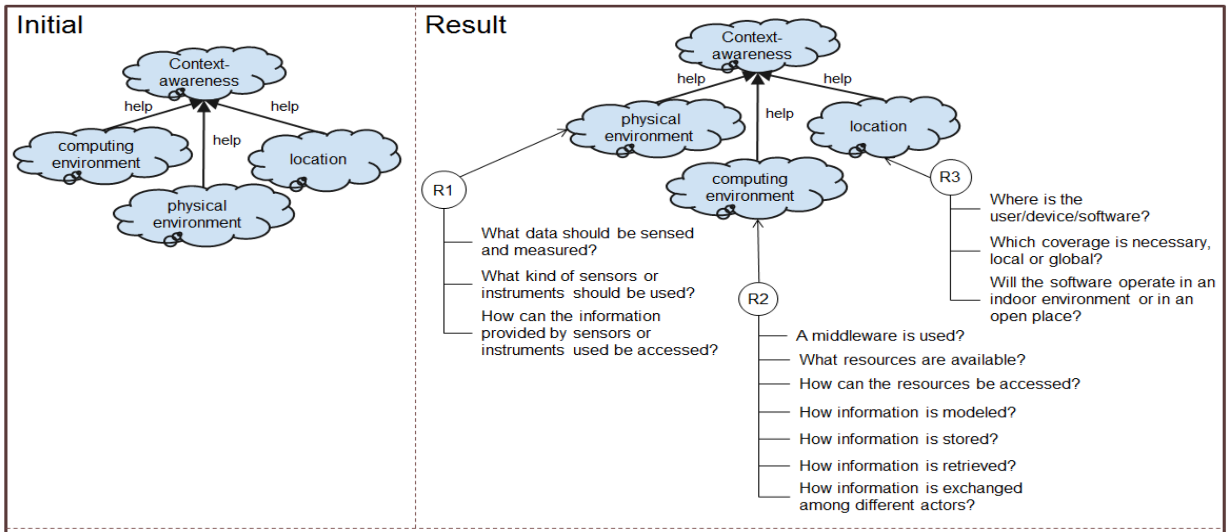
Our choice of i*, as a requirements modeling language for systems with the awareness quality, was bounded by the smooth transition from goal orientation to agent orientation, which is a reasonable choice for operationalization of awareness. However, we could have explored other requirements languages such as KAOS [7].

The i* framework models organizational contexts based on the dependency relationships between the actors and furthers our understanding of the internal reasons of the actors, since they are explicitly expressed, assisting in the choice of alternatives during the software modelling. Actors are active entities that perform actions to achieve goals through the exercise of their skills and knowledge, and may have intentional dependencies on each other. An intentional dependency occurs when the intentional element of this dependency is, somehow, related to some goal of one of the actors. i*, as originally proposed by Yu in [11], uses two models: the Strategic Dependency (SD) model and the Strategic Rationale (SR) model.

The SD model is used to map the network of dependencies between the organizational actors. The SD model is composed by a set of nodes representing the actors (agents, positions or roles) and links that represent dependencies between actors. A dependency is an agreement between two actors, where an actor (the depender) depends on another actor (the dependee) to satisfy a goal, to execute a task, to make a resource available or to satisficy (to reasonably satisfy) a softgoal. Goals, softgoals, tasks and resources that are part of the dependency relationships between actors are called *dependum* and featuring the four possible types of dependencies.

The SR model is a graph, with some types of nodes and links that provide a framework for explicitly express the reasons behind the process. The SR model aims to represent the internal strategies of each actor. The SR model provides an intentional description of the process in terms of the elements, the decisions and choices behind it. The SR model can explicitly represent the detailed understanding of the reasoning (rationale) of the actors in the process. These models are designed to express the process by which: goals are achieved, tasks are developed, resources are made available and softgoals are refined (decomposed) and operationalized. This is done by representing the intentional relationships that are internal to the actors through relationships "means-end" linking process elements and providing an explicit representation of 'why', 'how' and ‘alternatives’.

There are four types of nodes, similar to the types of dependum on an SD model: goal, task, resource, and softgoal. There are two main classes of links: task decomposition and means-end. In task decomposition links a task (whose concept is associated with 'how to do something') is modeled in terms of its decomposition into sub-components, which can be: goals, tasks, resources or softgoals (the four types of nodes). Means-End links indicate an "end", which can be: a goal to be achieved, a task to be performed, a resource to be produced, or a softgoal to be *satisficed* and alternatives "means" to achieve it. The means are usually expressed in the form of tasks, since the notion of task is associated with this ‘how to do something’: in i* graphical notation, an arrowhead pointing the means to the end.


Initial
Context-awareness
help
help
help
computing environment
location
physical environment
Result
Context-awareness
help
help
help
physical environment
location
computing environment
R1
What data should be sensed and measured?
What kind of sensors or instruments should be used?
How can the information provided by sensors or instruments used be accessed?
R2
A middleware is used?
What resources are available?
How can the resources be accessed?
How information is modeled?
How information is stored?
How information is retrieved?
How information is exchanged among different actors?
R3
Where is the user/device/software?
Which coverage is necessary, local or global?
Will the software operate in an indoor environment or in an open place?

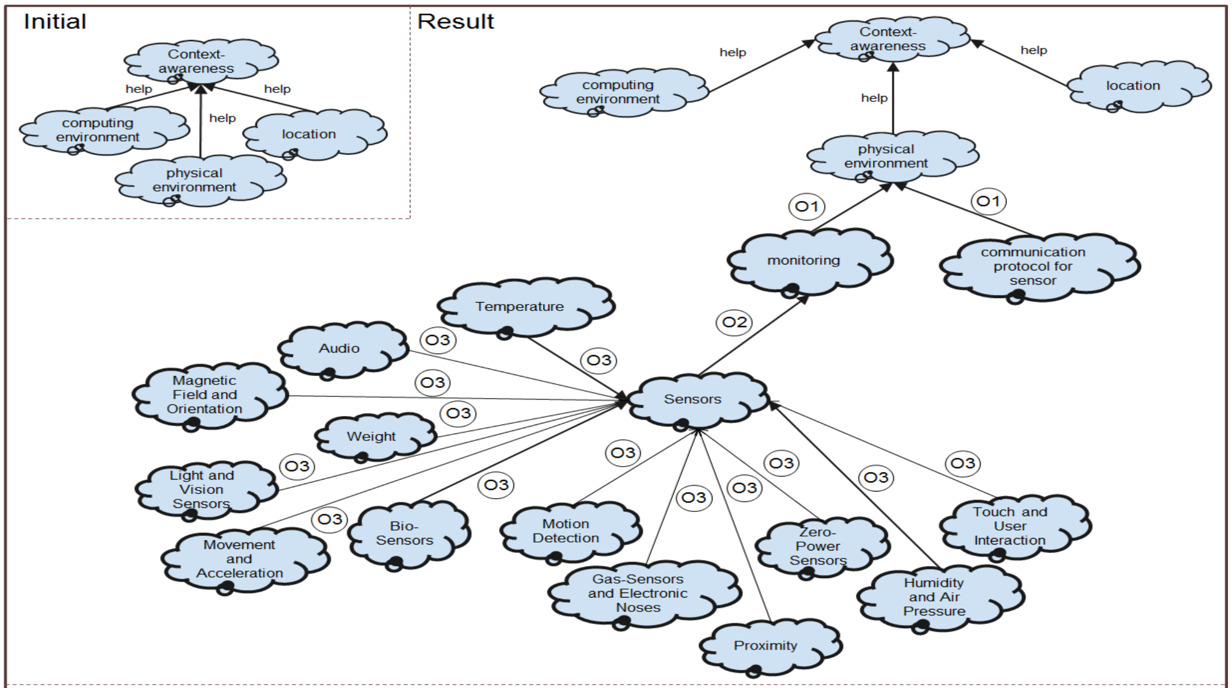
Refinement Rules

R1 - QuestionIdentification (physical enviroment) : {"What data should be sensed and measured?", "What kind of sensors or instruments should be used?", "How can the information provided by sensors or instruments used be accessed?"}

R2 - QuestionIdentification (computing enviroment) : {"A middleware is used?", "What resources are available?", "How can the resources be accessed?", " How information is modeled?", "How information is stored?", "How information is retrieved?", "How information is exchanged among different actors?" }

R3 - QuestionIdentification (location) : {"Where is the user/device/software?", "Which coverage is necessary, local or global?", " Will the software operate in an indoor environment or in an open place?" }

Figure 2 - A Question Pattern for Context-Awareness



Refinement Rules

O1 - OperationalizationAlternatives (physical enviroment) : {monitoring, communication protocol for sensor}

O2 - OperationalizationAlternatives (monitoring) : {Sensors}

O3 - OperationalizationRefinement (Sensors) : { Temperature, Audio, Magnetic Field and Orientation, Weight, Light and Vision Sensors, Movement and Acceleration, Bio-Sensors, Motion Detection, Gas-Sensors and Electronic Noses, Proximity, Zero-Power Sensors, Humidity and Air Pressure, Touch and User Interaction}

Figure 3 – An Alternative Pattern for Context-Awareness

IV. Extending i* with Awareness Modeling Constructs

Our proposal is to add, to i* models, abstractions that help software to perceive the environment with its inherent changes and to relate these new abstractions to other elements in the models that determine the software behavior. These abstractions are situation and context, as defined by Schmidt in [12].

According to Schmidt, a situation is “the state of the real world at a certain moment or during an interval in time at a certain location”. Context is defined as “a mechanism to describe situations by their defining features and group them into one unit” or in other words: “a context is a description of the current situation on an abstract level that can be matched against previously specified situations”. A description can be constituted by “a number of conditions that can be evaluated to true or false, possibly with an assigned certainty”. Still according to Schmidt, the following context properties are essential:

• Each context is anchored in an entity.

• Context awareness is always related to an entity.

An entity could be a place, an artifact, a subject, a device, an application, a different context, or a group of these. In our work, we consider as entities over which the context can be anchored the key elements of i*: actors (including their specializations: agents, roles and positions), intentional elements (goals, softgoals, tasks and resources), dependencies, and other contexts.

In order to represent situations in a given context in i* models, we extended the framework by adding a context element to the original set of intentional elements in the framework. Following, we present more details on this extension.

As our approach is split in the steps of data acquisition and data interpretation, we propose a particular construction for the new added element to i*: decompose the context-awareness element into two subtasks: data acquisition task and data interpretation task. The set with the awareness softgoal (end) along with the context-awareness element (mean) and its respectives subtasks for data acquisition and data interpretation, form a canonical structure known as SRConstruct as designed by Oliveira in [13]. A SRConstruct establishes a strategy for meeting a goal that can be reused in other instances. We consider that a SRConstruct is the core design pattern for i* models, based on the concept of Strategic Dependency Situations [13]. Based on the core pattern we proposed a set of patterns that guides the operationalizing of the awareness requirement in i* models. Figure 4 below shows the SRConstruct for the awareness softgoal, highlighted in blue.

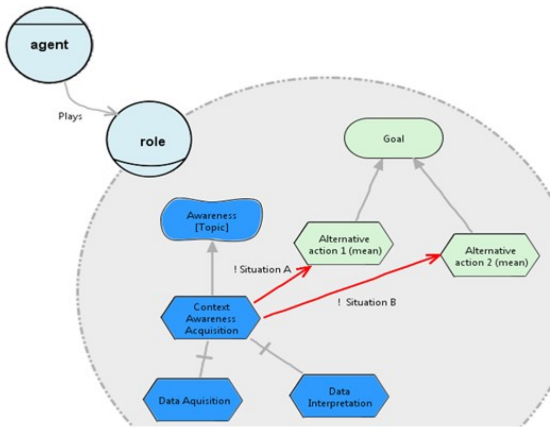


Figure 4 - SRConstruct for the Awareness Requirement

V. Illustration

A key point for engineering the awareness requirement is to define the set of situations that belong to a given context. For some classes of problems, the situations and the impact of context to achieve the goals can be identified directly and explicitly. There are problems that the maintenance of goals (desired states of the world) is achieved by monitoring a few signs in the environment and (re)acting in accordance with the changes. In this class of problems, the corresponding behavior in response to the underlying context situation can be defined in a deterministic way: based on values of monitored variables the agent can decide the best action to take in all cases (all combinations of variables).

For example, consider an intelligent room with an automatic door that opens when people approach the entrance/exit. After a short interval, e.g., 10 seconds, if there is no person near the door, it should be closed. In this example, the system behavior (to open or to close the door) directly depends on the underlying environmental context, captured via presence sensors, to achieve the goal 'automatically operate the door'. The two possible situations in this case are "presence of people" and "absence of people". Additionally, the agent that controls the door may possibly receive alerts from fire prevention system informing the likely presence of fire in the enclosure. In this case, the agent that controls the door has to open it for people who are inside the room can leave safely. Figure 5 shows the i* model extended to this problem. The situations "presence of people" and "absence of people" are mutually exclusive. But the situation "probable presence of fire" is orthogonal to these two and could happen simultaneously with both.

Reuse is instrumented by a specification pattern language (Table 1), which, departing from the core pattern (Figure 4), links the specific alternative pattern (Figure 3) to the case at hand. The pattern language links the catalogue (subtype) to the goal, detailing the suggested operationalization as given by the catalogue.

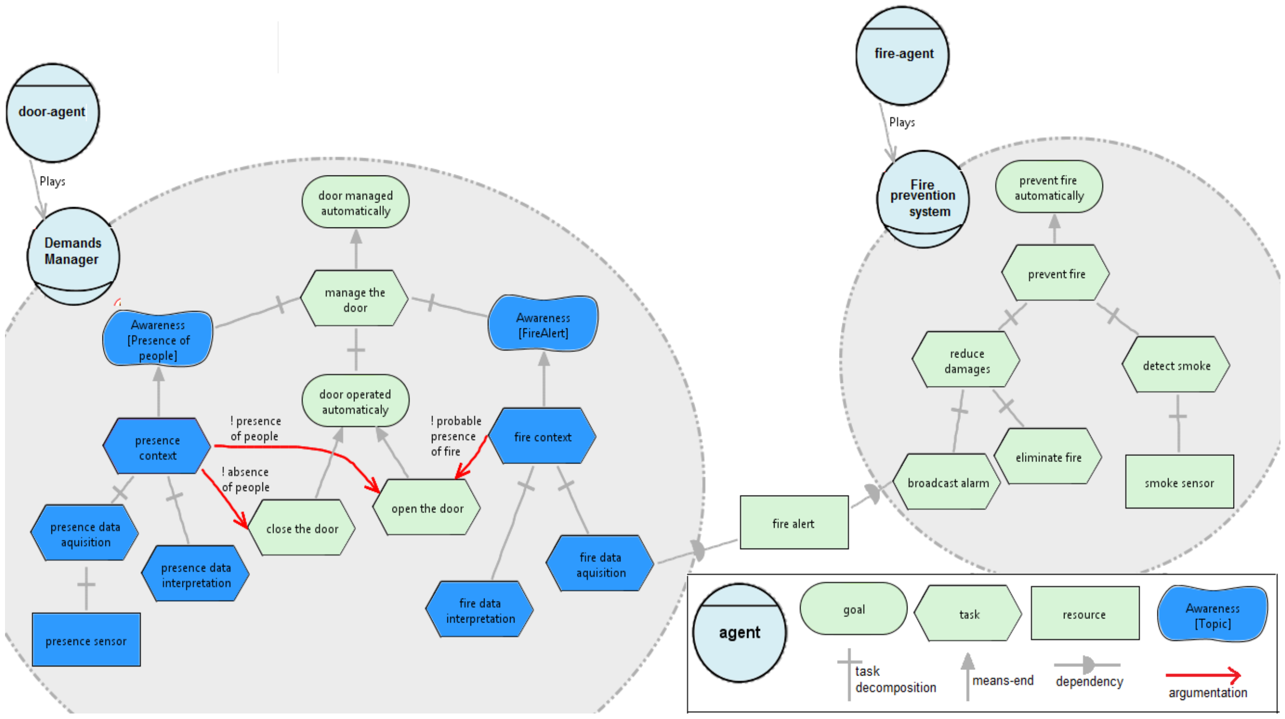


Figure 5 - extended i* model to automatic door system example

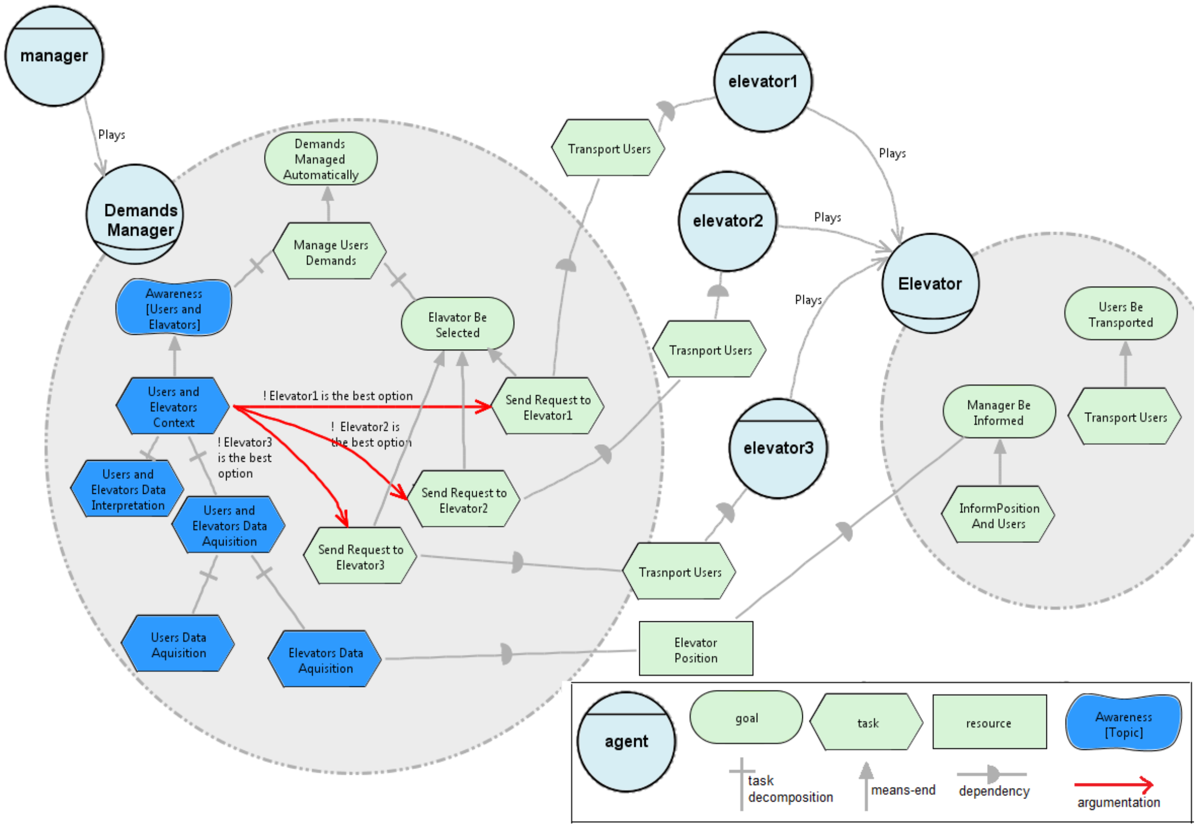


Figure 6 – extended i* model to automatic elevators system

The pattern language for awareness specification captures the aspects we believe that are relevant to an awareness requirement, according to what we discussed in this paper.

Table 1 – The Pattern Language

Awareness requirement name *<The name should follow the rule: Awareness [Topic] >*		
Topic description:	*<Brief description of the topic (problem domain) related to the awareness requirement>*	
Goal:	*<The goal impacted by the context>*	
Awareness subtype:	*<The awareness subtype (from awareness catalog) which is related to this requirement>*	
Suggested operationalization	*<Suggested operationalization to this requirement. Some of them can be found in awareness catalog >*	
Alternative actions:	*<The alternative means to achieve the goal impacted by the context>*	
Entity:	*<The entity in which the context awareness element is anchored (what the context is about)>*	
Source of entity data:	*<The source from where the entity data will be acquired>*	
Context description	*<List of variables that enables the situations identification>*	
Domains of variable *<domain definition section for variables in context description>*		
Variable name	Domain	
Context situations specification *<specification section for the context situations>*		
Situation name	Specification	
Alternative action choice *<specification of relations among situations and alternative actions>*		
Situation	Alternative action	Impact

Table 2, uses the pattern language to describe the specification for the operationalization of the awareness requirement related to Presence of People. It is , an instantiated pattern, with respect to the i* model of Figure 5.

Note that the specification pattern provides a trace to the NFR pattern in use (Awareness Subtype: *Physical environment*), in this case the Alternative Pattern showed in Figure 3, which lists for the monitoring of the physical environment, a series of possible types of sensors. It also has a trace to the goal in question (Goal: *Door managed automatically*).

Consider, now, another example: a building with a system for an automatic control of elevators. A partial intentional model is shown in Figure 6. When requesting the use of an elevator the user enters the destination floor (the current floor where the user is can be easily detected by the system). The system then decides which elevator will serve the user request.

Table 2 – The Specification Pattern (door management)

Awareness[Presence of People]		
Topic description:	The agent should open the door whenever there are people near the door. In order to do that, the agent needs to be aware of the presence of people near the door and act according to his perception.	
Goal:	Demands managed automatically	
Awareness subtype:	Physical environment	
Suggested operationalization	Use of presence sensors	
Alternative actions:	Open the door; Close the door	
Entity:	People using the room.	
Source of entity data:	People presence sensor	
Context Description	{peoplePresence}	
Domains of variable		
Variable name	**Domain**	
peoplePresence	{“PRESENCE”, “ABSENCE“}	
Context situations specification		
Situation name	**Specification**	
PresenceOfPeople	peoplePresence==”PRESENCE”	
AbsenceOfPeople	peoplePresence==”ABSENCE”	
Alternative action choice		
Situation	**Alternative action**	**Impact**
Presence of people	OpenTheDoor	MAKE (strongly positive)
Presence of people	CloseTheDoor	BREAK (strongly negative)
Absence of people	CloseTheDoor	MAKE (strongly positive)
Absence of People	OpenTheDoor	BREAK (strongly negative)

The i* model for multi-agent control system for elevators, Figure 6, has two roles: Demands Manager and Elevator. The manager agent, that plays the role Demands Manager, has a context element named UsersAndElevatorsContext that obtains the user's request, the Elevators positions. Once it has obtained the context information the agent decides to which Elevator it will send the request to transport the user. The agents who play the role Elevator must inform its schedule containing their current floor and the destination floor to which they are going to. If selected to satisfy the user transport request, the agent will receive a request for transport from the manager informing the user current floor and destination floor.

Table 3 – Specification Patterns (automatic elevator)

Awareness[Users and Elevators]	
Topic description:	The agent should be aware of user request (for transport between different floors) and elevators current position and schedule in order to decide which elevator will serve the user request.
Goal:	Users demands be managed automatically
Awareness subtype:	User awareness; Position (elevators)
Suggested operationalization	Use system user interface that allow users to inform their requests, including destination; Communicate with automation elevator system
Alternative actions:	Send the request to elevator1; Send the request to elevator2; Send the request to elevator3;
Entity:	Users and elevators
Source of entity data:	User request devices and elevators
Context Description	{UserRequest, ElevatorSchedule}
Domains of variable	
Variable name	**Domain**
UserRequest	{Trip}
ElevatorSchedule	{Trip}
Trip is a data entity with atributes: number of users (integer); source floor (integer between 1 and 15); destination floor (integer between 1 and 15)	
Context situations specification	
Situation name	**Specification**
Elevator1 is the best option	Elevator1 will travel the minor distance to meet the user request
Elevator2 is the best option	Elevator3 will travel the minor distance to meet the user request
Elevator3 is the best option	Elevator3 will travel the minor distance to meet the user request

Alternative action choice		
Situation	**Alternative action**	**Impact**
Elevator1 is the best option	SendToElevator1	MAKE (strongly positive)
Elevator2 is the best option	SendToElevator2	BREAK (strongly negative)
Elevator3 is the best option	SendToElevator3	MAKE (strongly positive)

Table 3 describes the specification pattern used in the case of Figure 6, in which the awareness requirement related to Users and Elevators is described, making a link to the catalogue regarding the subtype (Awareness subtype: *User awareness; Position (elevators)*). The trace to the goal being object of the awareness is given by the operator Goal (Goal: Demands managed automatically). The suggested operationalization is retrieved from the catalogue and inserted in the i* model.

It is also important to stress the context and the situations mapped in the specification pattern as to provide more detail and improve the description of the rationale of decisions embedded in both cases (Figure 5 and Figure 6).

These examples show the role of the specification patterns, following the pattern language, in providing a direct link from the catalogue to the intentional model in use. In this case the Goal clause of the pattern language is the link referring to the i* language and the Awareness Subtype is the link referring to a specific alternative pattern.

VI. Conclusion

In this paper, we have argued about the importance of the awareness requirement for software that aims to be more independent, in the direction of automaticity. In order to achieve this goal, we approached the problem as quality reuse [14].

We have used the idea of a NFR-Pattern [5] for organizing the awareness catalog in the same way it was used in [9] and in [15], where the NFR in question was regarding the quality of transparency [16].

Since we are proposing to bring the concept of awareness for the requirements process, we reused the information in the NFR-Patterns within the context of a goal oriented requirements modeling language, i*, given the results in deriving smart systems (agents) from intentional models: a well known example is the Tropos approach [20]. Serrano [21] has also showed that even a more directed transformation is possible. As such, we kind of extended i* as to make it awareness driven.

Although we have focused this paper on a specific quality, awareness, we understand that the proposal is applicable to other Non-Functional Requirements (NFR). Earlier work on the Transparency quality has produced similar results [9], but without the guidance of the pattern language (Table 1), which is customized for the specific quality at hand, but could also be instantiated for other Non-Functional Requirement. Future work will design and apply the pattern language to Transparency and other qualities.

Our work is focused on NFR patterns, other work, for instance [7] [22], have dealt with the issue of using patterns in the context of goal oriented requirements engineering. Our work differs from the [23] with respect to the representation of NFR patterns as well as its use in the requirements model.

Future work will also investigate how the NFR patterns could be linked to other requirements languages. Supakkul and Chung [19] had proposed directed links from KAOS and Problem Frames [8] components towards softgoals using contribution links: that is showing how components in any of those languages could be linked to softgoals by providing operationalizations. Since those links are of the type contribution, it is possible to perform satisficing analysis. In our case, our patterns provide the operationalizations, which can be plugged in a given requirements model. Of course, that linking to i* is easier, since softgoals are first class concepts in i*. However, the approach proposed by [19] provides a path to be explored.

As with respect to the awareness quality, we are planning to provide an even better integration of the specification pattern with i*, by augmenting the i* language, as well as to develop prototypes based on i* models. We plan to show that using an awareness driven i*, will be an effective way of implementing of awareness driven software from the requirements, given that requirements models would be transformed in a multi-agent system [3], [20].

ACKNOWLEDGMENT

Leite thanks the project “Lucretius: Foundations for Software Evolution” (http://www.lucretius.eu/) for providing inspiration for the concept of software awareness. Leite also acknowledges the financial support from CNPq and Faperj.

REFERENCES

[1] IBM Autonomic Computing White Paper. An architectural blueprint for autonomic computing. June 2005 - Third Edition.

[2] Souza, V. . Requirements-based Software System Adaptation. PhD Dissertation. International Doctorate School in Information and Communication Technologies. DISI - University of Trento. 2012.

[3] CUNHA, H. . Desenvolvimento de software consciente com base em requisites. Ph.D. Dissertation. Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, RJ, Brazil. 2014.

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

[5] SUPAKKUL S., Hill T, Chung L, Tun T. T., and Leite J.C.S.P. , "An NFR Pattern Approach to Dealing with NFRs," 18th IEEE Intl. Requirements Engineering Conf., Sydney, Australia, 2010.

[6] GAMMA, E.; Helm, R.; Johnson, R.; Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA. (1995).

[7] DARIMONT R. and LAMSWEERDE A. Formal refinement patterns for goal-driven requirements elaboration. SIGSOFT Softw. Eng. Notes 21. 1996.

[8] JACKSON, M.. Problem frames: analyzing and structuring software development problems , Addison-Wesley . 2001.

[9] SERRANO, M.; LEITE, J. C. S. P. , Capturing Transparency-Related Requirements Patterns through Argumentation. In: 1st International Workshop on Requirements Patterns (RePa 2011), 2011, Trento. 1st International Workshop on Requirements Patterns, 2011.

[10] CHUNG L., Supakkul, S. Capturing and Reusing Functional and Non-functional Requirements Knowledge: A Goal-Object Pattern Approach in the IEEE International Conference on Information Reuse and Integration. 2006.

[11] YU E. Modelling Strategic Relationships for Process Reengineering. Ph.D. Dissertation. University of Toronto, Toronto, Ont., Canada. 1996.

[12] SCHMIDT, A. Ubiquitous Computing - Computing in Context. PhD dissertation, Lancaster University. 2002.

[13] OLIVEIRA, A. P., Cysneiros, L.M., Defining Strategic Dependency Situations in Requirements Elicitation. Anais do WER06 - Workshop em Engenharia de Requisitos, Rio de Janeiro, RJ, Brasil, Julho 13-14, 2006, pp 12-23.

[14] LEITE, J.; Yu, Y.; Liu, L.; Yu, E. & Mylopoulos, J. Quality-Based Software Reuse. In: Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005, 2005, Porto, Portugal.

[15] CUNHA, H.; Sampaio do Prado Leite, J.C.; Duboc, L.; Werneck, V., "The challenges of representing transparency as patterns," Requirements Patterns (RePa), 2013 IEEE Third International Workshop on , vol., no., pp.25,30, 15-15 July 2013.

[16] LEITE, J. C. S. P. ; Cappelli, C. . Software Transparency. Business & Information Systems Engineering, p. 4, 2010.

[17] van LAMSWEERDE, A. (2000, June). Requirements engineering in the year 00: A research perspective. In Proceedings of the 22nd international conference on Software engineering (pp. 5-19). ACM.

[18] van LAMSWEERDE, A. "Requirements engineering: from system goals to UML models to software specifications." (2009).

[19] SUPAKKUL, S., and Chung L. "The RE-Tools: A multi-notational requirements modeling toolkit." Requirements Engineering Conference (RE), 2012 20th IEEE International. IEEE, 2012.

[20] CASTRO, J., Kolp, M., Mylopoulos, J.: A Requirements-Driven Development Methodology. Seminal Contributions to Information Systems Engineering 2013: 265-280.

[21] SERRANO, M., Leite, J.C.S.P.: Development of Agent-Driven Systems: from i* Architectural Models to Intentional Agents Code. iStar 2011: 55-60.

[22] KOLP, M.,Giorgini, P., Mylopoulos, J.,Organizational Patterns for Early Requirements Analysis. In Advanced Information Systems Engineering, LNCS 2681, Springer,2003, 617-632.

[23] GROSS, D., Yu, E., From Non-Functional Requirements to Design through Patterns, Requirements Engineering,V. 6, N. 1, Springer-Verlag, 2001, 18-36.