



Universidade Federal de Pernambuco

Centro de Informática

# **STREAM: A Systematic Process to Derive Architectural Models from Requirements Models**

**Marcia Jacyntha Nunes Rodrigues Lucena**

Recife, February

2010

**Marcia Jacyntha Nunes Rodrigues Lucena**

**STREAM: A Systematic Process to Derive Architectural Models  
from Requirements Models**

**Doctoral Thesis**

*Thesis presented to the Graduate Program in  
Computer Science of the Universidade Federal de  
Pernambuco in partial fulfillment of the  
requirements for the degree of Doctor of Science.*

Advisor: Jaelson Freire Brelaz de Castro

Co-Advisor: Carla Taciana Lima Lourenço  
Silva Schuenemann

Recife, February

2010

Á minha caríssima família:  
Leonardo, Daniel, Beatriz e meus pais, Rodrigues e Zélia

# Abstract

Requirements engineering and architectural design are key activities for successful development of software systems. Both activities are strongly intertwined and interrelated, but many steps toward generating architecture models from requirements models are driven by intuition and architectural knowledge. Thus, systematic approaches that integrate requirements engineering and architectural design activities are needed. This thesis presents a systematic approach based on model transformations to generate architectural models from requirements models. The source and target languages are respectively the i\* modeling language and Acme architectural description language (ADL). Non-functional requirements are used to select among architectural solutions and determine the architectural patterns that are applied. Two real case studies are used to show the feasibility of our approach: a web-based recommendation system and a web-based information system.

# Table of Contents

|          |                                              |           |
|----------|----------------------------------------------|-----------|
| <i>1</i> | <i>Introduction.....</i>                     | <i>1</i>  |
| 1.1      | Context .....                                | 2         |
| 1.2      | Problem .....                                | 4         |
| 1.3      | Objectives.....                              | 5         |
| 1.4      | The Proposed Solution.....                   | 6         |
| 1.5      | Contributions.....                           | 7         |
| 1.6      | Methodology.....                             | 8         |
| 1.7      | Organization of Chapters .....               | 10        |
| <i>2</i> | <i>Source and Target Models .....</i>        | <i>11</i> |
| 2.1      | Requirements Process – a general view .....  | 12        |
| 2.2      | Goal Oriented Requirements Engineering ..... | 13        |
| 2.3      | I* Requirements Modeling Technique.....      | 15        |
| 2.4      | Software Architecture.....                   | 24        |
| 2.5      | Acme.....                                    | 25        |
| 2.6      | Summary.....                                 | 29        |
| <i>3</i> | <i>STREAM Approach.....</i>                  | <i>30</i> |
| 3.1      | Introduction.....                            | 31        |
| 3.2      | Running Example .....                        | 32        |
| 3.3      | Prepare Requirements Models .....            | 36        |
| 3.4      | Generate Architectural Solutions .....       | 55        |
| 3.5      | Select an Architectural Solution.....        | 59        |
| 3.6      | Refine Detailed Architecture .....           | 60        |
| 3.7      | Related Work .....                           | 70        |
| 3.8      | Conclusion.....                              | 72        |
| <i>4</i> | <i>Running Example.....</i>                  | <i>74</i> |

|          |                                                             |                   |
|----------|-------------------------------------------------------------|-------------------|
| 4.1      | Introduction.....                                           | 75                |
| 4.2      | BTW Project .....                                           | 75                |
| 4.3      | Prepare Requirements Models .....                           | 85                |
| 4.4      | Generate Architectural Solutions .....                      | 98                |
| 4.5      | Select Architectural Solutions .....                        | 100               |
| 4.6      | Refine Architecture .....                                   | 102               |
| 4.7      | BTW architecture of SCORE project.....                      | 104               |
| 4.8      | Evaluating Architectural Models .....                       | 105               |
| 4.9      | Conclusions .....                                           | 109               |
| <b>5</b> | <b><i>Conclusions .....</i></b>                             | <b><i>110</i></b> |
| 5.1      | Introduction.....                                           | 111               |
| 5.2      | Future Works .....                                          | 111               |
| 5.3      | Summary of the publications .....                           | 112               |
|          | <b><i>References.....</i></b>                               | <b><i>114</i></b> |
| <b>A</b> | <b><i>Appendix.....</i></b>                                 | <b><i>122</i></b> |
| A.1      | Textual representation of the Medi@ System (Chapter 3)..... | 123               |
| A.2      | Textual representation of the BTW System (Chapter 4).....   | 126               |

# List of Figures

|                                                                                                                           |    |
|---------------------------------------------------------------------------------------------------------------------------|----|
| Figure 2-1 Dimensions of Problem ([Lamsweerde, 2009]).....                                                                | 13 |
| Figure 2-2 SD model.....                                                                                                  | 16 |
| Figure 2-3 Use of i* means-end relationships.....                                                                         | 17 |
| Figure 2-4 Use of i* task-decomposition relationship.....                                                                 | 18 |
| Figure 2-5 Example of SR model.....                                                                                       | 19 |
| Figure 2-6 Example of SR model with expanded actors (A) and the meaning of dependency between internal elements (B) ..... | 20 |
| Figure 2-7 i* Meta-model.....                                                                                             | 22 |
| Figure 2-9 Graphical Representation of Structural Elements of Acme .....                                                  | 27 |
| Figure 2-10 Example for the textual and graphical representation of Acme.....                                             | 27 |
| Figure 2-11 Acme meta-model.....                                                                                          | 28 |
| Figure 3-1 Overview of STREAM Process.....                                                                                | 32 |
| Figure 3-2 SD i* Model to Media Shop.....                                                                                 | 33 |
| Figure 3-3 Medi@ actor in SD i* Model to Media Shop.....                                                                  | 34 |
| Figure 3-4 Medi@ SR Model .....                                                                                           | 35 |
| Figure 3-5 Prepare Requirements Work-definition and its activities flow.....                                              | 38 |
| Figure 3-6 SR model of Medi@ with highlighted Internal Elements .....                                                     | 40 |
| Figure 3-7 i* SR model after applied transformation rules.....                                                            | 49 |
| Figure 3-8 Modular SR i* model with MediaShop actor .....                                                                 | 54 |
| Figure 3-9 Mapping a generic dependency between i* actors to Acme .....                                                   | 55 |
| Figure 3-10 Mapping a goal dependency to Acme.....                                                                        | 56 |
| Figure 3-11 Mapping a task dependency to Acme.....                                                                        | 57 |
| Figure 3-12 Mapping a resource dependency to Acme.....                                                                    | 57 |
| Figure 3-13 Mapping a softgoal dependency to Acme.....                                                                    | 58 |
| Figure 3-14 Architectural Solution with component using representations .....                                             | 59 |
| Figure 3-16 Activities Flow of Derive Detailed Architecture Work-definition.....                                          | 60 |
| Figure 3-18 MVC pattern represented in Acme.....                                                                          | 65 |
| Figure 3-19 Media after applying MVC pattern .....                                                                        | 66 |
| Figure 3-20 Architectural Solution for Medi@ after application of maintain availability refinement pattern .....          | 68 |
| Figure 4-1 BTW system early requirement SD Diagram.....                                                                   | 77 |

|                                                                                   |     |
|-----------------------------------------------------------------------------------|-----|
| Figure 4-2 SR Diagram for BTW system early requirement.....                       | 79  |
| Figure 4-3 Late Requirements BTW SD Model.....                                    | 81  |
| Figure 4-4 Late Requirements SR Model of BTW System.....                          | 84  |
| Figure 4-5 First activity of the STREAM approach.....                             | 85  |
| Figure 4-6 First sub-activity of the Prepare Requirements activity.....           | 86  |
| Figure 4-7 SR model of BTW with highlighted Internal Elements.....                | 87  |
| Figure 4-8 Second sub-activity of the Prepare Requirements activity.....          | 88  |
| Figure 4-9 Moving Sub-graph B – Map be Handled Goal .....                         | 90  |
| Figure 4-10 Applying TR1 and TR3 after moving sub-graph B .....                   | 91  |
| Figure 4-11 Moving Subgraph C – Information be Publisher in Map.....              | 92  |
| Figure 4-12 Applying TR1-v1 and TR3 after moving sub-graph C.....                 | 93  |
| Figure 4-13 Moving Subgraph A – User Access be Controled Goal.....                | 94  |
| Figure 4-14 - Applying TR3 after moving sub-graph A .....                         | 95  |
| Figure 4-15 Third sub-activity of the Prepare Requirements activity.....          | 96  |
| Figure 4-16 Second activity of the STREAM approach.....                           | 98  |
| Figure 4-17 BTW Architectural model produced by the third activity of STREAM..... | 100 |
| Figure 4-18 Third activity of STREAM.....                                         | 101 |
| Figure 4-19 Selected Architectural Solution for the BTW System.....               | 102 |
| Figure 4-20 Forth Activity of STREAM.....                                         | 103 |
| Figure 4-21 BTW architecture after applying the Layer architectural pattern ..... | 104 |
| Figure 4-22 BTW architecture of SCORE project.....                                | 105 |

# List of Tables

|                                                                                               |     |
|-----------------------------------------------------------------------------------------------|-----|
| Table 2-1 Contribution link labels [Grau, 2008].....                                          | 18  |
| Table 2-2 Valid situations outgoing dependency link and incoming dependency .....             | 21  |
| Table 2-3 OCL expressions to i* meta-model .....                                              | 23  |
| Table 3-3 Transformation Rule to move elements in a task-decomposition link (TR1).....        | 42  |
| Table 3-4 Transformation Rule to move goals in a Task-decomposition link (TR1v1) .....        | 43  |
| Table 3-5 Transformation Rule to move alternatives (means) in a means-end link (TR2)..        | 44  |
| Table 3-6 Transformation Rule to move a goal (means) in a means-end link (TR2-v1).....        | 44  |
| Table 3-7 Transformation Rule to move a contribution link crossing the actor's boundary ..... | 45  |
| Table 3-8 Transformation Rule to move shared elements .....                                   | 46  |
| Table 3-9 Transformation Rule to change the name of New Actor .....                           | 47  |
| Table 3-1 Metrics from Sant'Anna's Framework [Santa'Anna et al, 2003] .....                   | 51  |
| Table 3-2 Correlation between i* elements and concepts of Sant'Anna's metrics framework ..... | 51  |
| Table 3-10 Coupling and cohesion metrics.....                                                 | 52  |
| Table 3-11 NFR vs. NFR based on [Reza, 2005] .....                                            | 62  |
| Table 3-12 Maintain Availability refinement patter.....                                       | 67  |
| Table 3-13 <i>Maintain Accurate Data</i> refinement pattern.....                              | 68  |
| Table 3-14 <i>Maintain FaultTolerantCommunication refinement pattern</i> .....                | 69  |
| Table 3-15 Maintain Interoperability refinement pattern .....                                 | 70  |
| Table 4-1 Summary of modularity metrics.....                                                  | 97  |
| Table 4-2 Results of the application of modularity metrics to BTW i* models .....             | 98  |
| Table 4-3 Suite of modularity architectural Metrics [Sant'Anna et al, 2007].....              | 106 |
| Table 4-4 Results after applying architectural metrics .....                                  | 107 |

# List of Acronyms

|        |                                                                              |
|--------|------------------------------------------------------------------------------|
| ADL    | Architectural Description Language                                           |
| GORE   | Goal Oriented Requirements Engineering                                       |
| NFR    | Non-Functional Requirements                                                  |
| OCL    | Object Constraint Language                                                   |
| OMG    | Object Modeling Group                                                        |
| SD     | Strategical Dependency                                                       |
| SPEM   | Software Process Engineering Metamodel                                       |
| SIRA   | Systematic Integration between Requirements and Architecture                 |
| SR     | Strategical Rationale                                                        |
| STREAM | Strategy for Transition between Requirements models and Architectural Models |

# 1 Introduction

---

This chapter presents the main motivations of this thesis. Afterwards, it describes the current stage of the relationship between requirements and architecture. The following sections point to the work scope, the objectives of this thesis, the methodology and the proposed approach, as well as the document structure.

---

## **1.1 Context**

Requirements Engineering (RE) plays an important role in the software development process. RE involves activities that support understanding and managing changes in requirements during the software development process. Evidences show that requirements errors, such as misunderstandings and omissions, are more expensive to repair in project activities [Boehm, 2007]. Currently, in Model-Driven Development (MDD) approaches [Kleppe, 2003], the importance of the requirements modeling is even greater, because the requirements models are the starting point of the transformation process.

In the context of Model Driven Development (MDD), software development is done by successive transformations on models. These transformations enable a smooth and systematic transition between the initial and final model. This strategy minimizes the gaps between models and reduces the chances that the final solution does not satisfy the initial specification.

As requirements engineering is considered as an early activity in the development process, software architecture is the initial activity for developing the solution. The software architecture defines the structure from which the system will be developed. Other project activities for the solution, such as detailed design and implementation, depend on the architectural design. Although software architecture is something that can be modified, the cost of late changes increases over the development life cycle [Shaw et al., 1996]. Decisions taken to define the architecture of a system have great influence on the quality of the final product [Taylor, 2009]. Factors that influence decisions regarding the definition of the architecture are usually present in the requirements document. Besides the functional requirements, there are non-functional requirements that have a great impact on the architecture. Thus, the requirements directly influence the generation of the architecture. The generation of the architecture will depend on how the transition between requirements and architecture will take place.

Many efforts have been made to understand the interaction between requirements engineering and architectural design activities. The two STRAW (Workshop From Software Requirements to Architectures) workshops [Castro, 2001] [Berry, 2003] and the series of SHARK (Workshop on SHaring and Reusing architectural Knowledge) workshops [Lago, 2009] [Avgeriou, 2008] are some of the events dedicated to this research

area. In SHARK workshop, the focus is related to understanding the reasoning of architecture development, that is, to capture the whys and reasons of the chosen architectural solution. For instance, an intermediate model can be introduced to bridge the gap between requirements-related concepts and the architecture-related ones [Grunbacher, 2001]. Requirements-architecture links can also be studied in the problem frame setting; architectural constraints are then elements of the problem domain that can drive problem decomposition and recomposition [Rapanotti, 2002].

In this thesis, we adopt the goal-oriented requirements engineering (GORE) [Lamsweerde, 2001]. GORE refers to the use of goals for requirements elicitation, analysis, negotiation, assignment, documentation and evolution [Lamsweerde, 2009]. A goal is an aim to be achieved by the system under consideration. GORE approaches clarify the goals of the system, thus allowing reasoning and analysis of alternative models of requirements, enabling the architects to deepen their understanding of the problem [Lamsweerde, 2003]. One of the goal-oriented approaches widely used in academy and industry is the *i\** framework [Yu, 1995]. It captures the social and intentional relationships in the system organizational environment, as well as the functionalities and quality attributes of the system. In this thesis, requirements specification relies on *i\** models.

For the descriptions of software architecture, we rely on Acme, a general-purpose architectural description language (ADL) that provides a simple structural framework for architectural design [Garlan et al, 2000]. Acme uses an annotation mechanism to support semantic descriptions. Besides, Acme is an interchange language that allows translating architectural models into other ADLs, enabling generated architectural models to be described in these languages. ADLs are the conventional architectural forms of showing how architecture design deals with both functional and non-functional requirements [Garlan et al, 1997].

Architectural patterns are a key concept in software architecture, since they offer well-established solutions to architectural problems, help to document the architectural design decisions, promote communication between stakeholders through a common vocabulary, and describe the non-functional requirements of a software system as requirements to be achieved [Buschmann, 1996] [Avgeriou, 2005].

We adopt the use of architectural patterns based on [Buschmann, 1996], since they are considered as problem-solution pairs that occur in a given context and are affected by it. Further, a pattern does not only document how a solution solves a problem but also why

it is solved, that is the rationale behind this particular solution. The description of architectural patterns is based on the context-problem-solution triplet and may be further elaborated with details, especially focusing on the rationale behind a solution.

Furthermore, the architectural refinement patterns used in this work definition is based on Lamsweerde's patterns [Lamsweerde, 2003][Lamsweerde, 2009]. The key principle of architectural refinement is to inject non-functional requirements from goal model within component interactions (connector refinement) or within single component (component refinement) [Lamsweerde, 2003].

## **1.2 Problem**

An appropriate understanding of the problem is a pre-requisite for the achievement of an adequate solution. In software development, Requirements Engineering seeks understanding, analyzing, reasoning and documenting requirements of software systems. Architecture, design and implementation intend to meet these requirements through the documentation, development and implementation of artifacts related to these systems.

Many researchers associate the requirements with the problem-space and the architecture with the solution-space [Deboer, 2008]. However, there is a semantic gap between requirements and system architecture, which consists of conceptual differences between what to do (requirements) in opposition to how to do (architecture, design and coding). From the challenges that appear when developers try to match requirements and architecture [Grünbacher et al, 2004], we highlight:

(i) Requirements are often captured informally in natural language while architecture is specified on a semi-formal way;

(ii) Properties of the software described as non-functional requirements are difficult to specify in architectural models;

(iii) The iterative and concurrent evolution of requirements and architectural models are based on incomplete requirements. Besides this, some requirements can only be understood after modeling the architectural design;

(iv) Mapping requirements to architecture and the consistent maintenance of these artifacts are difficult tasks. Besides, a simple requirement may address various architectural elements and a simple architectural element may have nontrivial links to several requirements.

Thus, these issues related to the transition between requirements and architecture, make it difficult to build a bridge between them. Although there are many methods, processes and notations specifically tailored to requirements [Kotonya, 1998] and architecture [Taylor, 2009], it is unusual to have a systematic approach that uses the analysis and understanding of the problem and produces a suitable solution to meet satisfactorily the requirements. Despite the fact that there are differences between these activities, the techniques and methods used for software development should include a systematic way of dealing with the relationship between requirements models and architectural models.

What is the point of building rich, consistent, and complete requirements models if the transition to the architecture is developed on an ad hoc basis, without paying attention to the relationship between the consumed and produced models? Moreover, some information present in the requirements specification, such as the non-functional requirements, can be lost in the transition to the architectural design, since with an ad hoc method it is more difficult to verify which requirements are present in the architectural design. It is important to ensure that the requirements information is not lost during the development process.

### **1.3 Objectives**

The aim of this thesis is reduce the gap between requirements and architecture activities, specifically between i\* requirements models and Acme architectural models. To achieve this general objective, the specific objectives are the following:

1. Develop a systematic approach to integrate requirements engineering and architectural design activities based on model transformations;
2. Starting with an i\* model describing the requirements, gradually transform the initial specification into an architectural description in Acme;
3. The strategy should take into account both functional and non-functional requirements;
4. Face three of the challenges describe above, (i), (ii), and (iv).
5. Use of modularly metrics to evaluate the intermediary models produced during models transformation.
6. Compare architectural solutions produced by our approach to ad-hoc architectural solutions using modularity metrics.

## 1.4 The Proposed Solution

With the widely use of iterative and incremental software development process as de facto standard, a strong integration between requirements specification and architectural design activities can efficiently promote traceability and the propagation of changes between requirements models and architectural models [Silva et al., 2007]. In this context, we can highlight the twin peaks model [Nuseibeh, 2001], which emphasizes the co-development of requirements and architectures in an incremental way.

Since there is a close relation between architectural design and requirements specification [Deboer et al., 2008], the Model-driven Development (MDD) [Kleppe, 2003] can be an effective way to generate architectural models from requirements models, by using model transformations rules which can specify accurately the correlation between requirements and architectural models.

This thesis proposes STREAM (Strategy for Transition between Requirements models and Architectural Models). STREAM defines a systematic strategy to integrate requirements and architecture based on MDD that includes four steps: (i) preprocessing requirements models to increase their modularity and make it easier the transition to architectural models; (ii) deriving architectural solutions from requirements model to; (iii) choosing an architectural solution based on non-functional requirements, and (iv) choosing and applying architectural patterns to refine architectural solutions. Thus, to perform the steps (i) and (ii), respectively, horizontal and vertical model transformation rules are proposed.

The objective of this approach is generating architectural model from requirements model based on the MDD approach. To reach this objective, we had to:

- (i) Define an metamodel for  $i^*$  [Lucena et al, 2008] and a metamodel for Acme;
- (ii) Define a set of horizontal rules and a process to transform  $i^*$  models in modular  $i^*$  models [Lucena et al, 2009a][Lucena et al, 2009b];
- (iii) Define a set of vertical rules and a process to map  $i^*$  models to Acme architectural models [Lucena et al, 2009c];
- (iv) Define a set of metrics to evaluate the degree of modularity and complexity of  $i^*$  models;

(v) Use non-functional requirements to choose one architectural solution among alternatives;

(vi) Use non-functional requirements to choose among architectural patterns to refine the chosen architectural solution.

We run some examples to demonstrate our approach. We have selected two case studies from different application domains and with different degrees of complexity: an e-commerce system [Castro et. al., 2002] and a web-based recommendation system [Pimentel et al., 2009].

## **1.5 Contributions**

The use of goal-oriented approaches and, in particular, the  $i^*$  framework, brings some benefits to our approach. We can highlight that with  $i^*$  models (i) there is a better traceability between early (organizational) and final (system) requirements; (ii) we can describe in the same model both functional and non-functional requirements; (iii) we can describe the rationale of how to achieve the non-functional requirements. Further, considering the challenges presented previously in the section 1.2, our approach contributes to address the following issues:

- i. Lack of formalism in requirement description: in opposite to describe the requirements in natural language, in our approach we use the  $i^*$  language, which has a well-defined notation and semantics [Yu, 1995][Yu et al, 2008]. Thus, the initial model of requirements will have less inconsistencies and ambiguities compared with a natural language.
- ii. Difficulty in describing non-functional requirements in architectural models: a process that systematically derives architectural model from requirements model make the role of non-functional requirements clearer in architectural design phase;
- iii. Incompleteness of requirements: In goal-oriented approaches, the goals of the system are understood and documented in early phases. Later they are refined into elements that are more concrete and more dependent on the system. Note that the  $i^*$  requirements models are based on goals that are less susceptible to change than requirements model based on functionalities.

- iv. Mapping between requirements and architectural elements: increasing the modularity of i\* models, based on the domain dependency criteria, could reduce the number of relationships one-to-one between requirements elements and architectural elements. In fact, a group of requirements elements in i\* models is mapped to one architectural element.

Besides the benefits of using a goal-oriented approach, we highlight other benefits brought by using an MDD approach:

- (i) Both requirements and architecture will be captured in a semi-formal way using models to be transformed;
- (ii) Some non-functional requirements may be traced from requirements to architectural models;
- (iii) Propagation of changes from architectural model to requirements model will help to have a better knowledge of requirements;
- (iv) The consistency between requirements models and architectural descriptions is improved.

## **1.6 Methodology**

Research in Software Engineering may be conducted through four different methods: Scientific, Engineering, Experimental and Analytical [Travassos, G. et al 2002]. This thesis follows the engineering method, which is based on the observation of existing solutions, the identification of problems in these solutions and suggestions of approaches to improve the analyzed solutions.

Initially, a study on languages used by Requirements Engineering and Software Architecture was performed. In particular, requirements modeling languages based on Goal Oriented Requirements Engineering (GORE) and architectural description languages (ADL) were studied.

Concerning the requirements modeling language, we sought to include the research within the context of the LER (Requirements Engineering Laboratory). Thus, the choice of the i\* was quite natural since many research works have used that language<sup>1</sup>. It was verified that there are many variants of the i\* language and, therefore, we had to compare them and select from them the one to be used in our work. The results of that comparison were

---

<sup>1</sup> LER – Requirements Engineering Laboratory - [www.cin.ufpe.br/~ler](http://www.cin.ufpe.br/~ler)

published in [Lucena et al. 2008], in which the most representative  $i^*$  variations were presented. This research demonstrated some weaknesses in the  $i^*$  language, specially related to the complexity and modularity of the produced models. For address these issues, some works dealing with the  $i^*$  legibility [Alencar, et al, 2008] and  $i^*$  modularization [Lucena, et al, 2009b] were developed.

When studying architectural languages, we opted for a more generic language that included elements from other ADLs. This would allow for future generalizations of our strategy in order to have other ADLs as target languages. Acme language was chosen because it satisfied the requisite of being a generic language, besides having a simple visual representation.

At the same time we studied some requirement modeling languages and architecture description languages, we also analyzed some methods that integrate the requirements and architecture activities. Particularly we considered those methods that use requirements languages based on GORE [Lamsweerde, 2009][Bastos et al, 2005][Silva et al, 2007] . Our approach takes advantage of the  $i^*$  language, by analyzing alternative solutions for requirements and taking these alternatives to generate alternative architectural solutions.

Considering the acquired knowledge and preliminary results reached [Lucena et al, 2009c], we defined an approach called STREAM (Strategy for Transition between Requirements models and Architectural Models) which has the  $i^*$  model as requirements model and produces an architecture description in Acme, taking into account the application of patterns.

Finally, to evaluate this research, we applied our approach to a Web Recommendation System [Pimentel 2009], which was shortlisted for a software engineering world competition. The architectural design of this system was developed using an ad hoc approach, while, in this thesis, we applied the STREAM approach to generate an architectural design. The results were compared using proper metrics.

## **1.7 Organization of Chapters**

This thesis is organized as follows.

Chapter 2 – Source and Target models: this chapter presents an overview of requirements engineering and software architecture concept. In particular outlines i\* models and Acme ADL and their respective meta-models.

Chapter 3 – The STREAM Approach: this chapter presents our approach to generate architectural model from requirements model, showing the principal contributions developed. To illustrate our approach we present along the chapter a running example related to the e-commerce domain.

Chapter 4 – Running Example: this chapter presents details of the use of the STREAM approach in the context of the development of a real web-based recommendation system. The goal is to improve the understanding of the approach. We also discuss the benefits and limitations of our work.

Chapter 5 – Conclusions and Future Works: some considerations are presented together with proposal for future works.

# 2 Source and Target Models

---

This chapter presents an overview on requirements and architecture showing the main concepts and challenges. We also introduce the techniques used to describe requirements and architecture of this work, respectively  $i^*$  and Acme.

---

## **2.1 Requirements Process – a general view**

Requirements describe functionalities and constraints related to the system under development. Requirements Engineering (RE) is an area of Software Engineering that is concerned with describing the objectives of the real world for software systems and present their functions and restrictions. It includes methods, processes and activities to discover, elicit, describe and model the requirements of the system. In order to know whether the solution satisfies a problem, we must first understand correctly what problem needs to be resolved [Lamsweerde, 2008].

Requirements can be classified into three different, but strongly related, basic classes: functional, non-functional and organizational [Loucopoulos, 1995]. Informally, functional requirements describe what the system must do; non-functional requirements describe how these functional requirements will be satisfied by the software system and how they must be understood within an organizational context.

According to [Lamsweerde, 2000] and [Nuseibeh, 2001], Requirements Engineering covers many interrelated activities such as:

- Domain Analysis: studies the environment of the system. The relevant stakeholders are identified and interviewed. Problems with the current system are found and enhancement opportunities are investigated. This activity identifies the general objectives of the target system.
- Elicitation: alternative models of the target system are explored according to the identified objectives. Scenarios can be included to assist the elicitation process.
- Negotiation and agreement: alternative requirements and assumptions are evaluated; the stakeholders analyze risks; the best alternatives are selected.
- Specification: requirements and assumptions are precisely formulated.
- Specification Analysis: the specifications are checked with respect to problems like incompleteness, inconsistency and feasibility in terms of the resources required, development cost and so on.
- Documentation: the decisions taken in the process are documented with the respective rationale and premises.
- Evolution: the requirements are modified to perform corrections, reflect changes in the environment and incorporate new objectives.

Since our work is related to the description of requirements models and its impact in architectural models, we can highlight the activities of specification, documentation and evolution as strongly related to this work.

## 2.2 Goal Oriented Requirements Engineering

Even though traditional approaches of requirements engineering focus on the functionalities of the system, in the last few years, researches have evolved from software systems to a broader perspective that incorporate aspects of organizational environment [Loucopoulos, 1995]. In this context, one of the most used approaches is the Goal-Oriented Requirements Engineering (GORE), which concerns in discovering and describing the objectives of software systems.

In goal-oriented approaches [Lamsweerde, 2001], the role of RE is related to the following questions: discover, formulation, analysis and agreement of for *what* the problem is being solved, *why* the problem must be solved and *who* is responsible for resolving the problem. Therefore, the identification of these questions must be the main activities of software requirements development. Thus, according to [Lamsweerde, 2008], the problem may be structured in these three dimensions (). We have to know why it is necessary to develop the system, which necessities must be addressed, and who is responsible for doing such necessities.

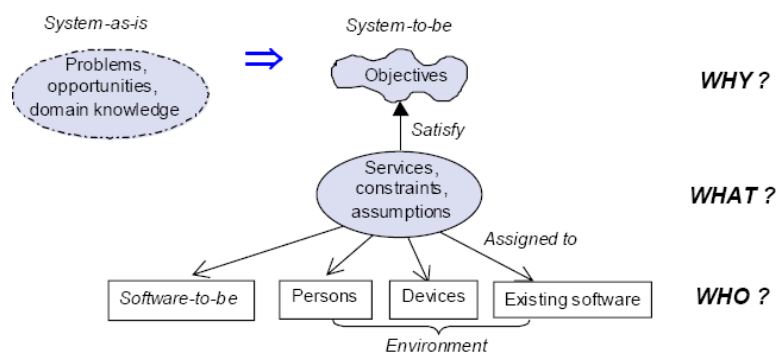


Figure 2-1 Dimensions of Problem ([Lamsweerde, 2009])

There are some advantages of using goal-oriented approaches in the RE process [Lamsweerde, 2001]. Among them, we can highlight:

- Completeness of requirements. Goals enable rich criteria for sufficient completeness of a requirement specification; the specification is complete with respect to a set of goals if all goals can become evident from the specification and the known properties regarding a domain.
- Avoid irrelevant requirements. Goals supply rich criteria for requirements pertinence; a requirement is pertinent with respect to a set of goals of a domain if their specification is used to prove at least one goal.
- Support the requirements rationalization for the stakeholders. Goals provide the rationale for requirements in the same way as goals of project in project process. Explicit goal refinement trees provide links for traceability from high-level goals to lower level requirements.
- Refinement of goals provides a usual mechanism to structure complex requirements documents and so increases readability.
- Refinement of goals allows the analysis of alternatives providing a high level of abstraction, which is adequate for validation of the choices or suggestion of new alternatives to guide a decision on a high level of abstraction.
- Conflict management between different points of view. Using goals the conflicts can be detected and later resolved.
- Separating stable information from volatile. A requirement represents a way of achieving some specific goal. Thus, the requirement is more suitable to evolve than the actual goal. The higher the level of abstraction of the goal, the more stable it will be.

Goals may be organized in structures that capture how they are and how they can be refined. Goals can cover from high-level concepts, such as strategic goals, to technical requirements to be designated to agents [Lamsweerde, 2009]. They may be requirements or expectations of the system in your environment.

As the use of goals grew in Requirements Engineering, there are several techniques where goals are used as abstraction, including KAOS [Dardenne, 1993], NFR Framework [Chung, 2000], V-Graph [Yu, 2004] and  $i^*$  [Yu, 1995][Yu, 2008]. Among these approaches, we highlight  $i^*$ , which is used in our thesis and will be presented in the next section.

## **2.3 *i\** Requirements Modeling Technique**

*i\** (i-star) [Yu, 1995] is intended to facilitate the reasoning about the system being developed, emphasizing the social aspects, represented by actors. It provides a graphical view of these actors and their intentions, dependencies, responsibilities and vulnerabilities. This approach is used to: (i) obtain a better understanding of the relationships of the organization, (ii) understand the reasons involved in the process of decisions, and (iii) describe potential alternatives of the desired requirements [Yu, 2002].

Thus, the *i\** approach views organizations as consisting of semi-autonomous units called actors, whose behaviors are not completely controllable or predictable, but regulated by social relationships. The actors have freedom of action within the context of social restrictions, and have intentional properties, such as goals, beliefs, abilities and commitments.

*i\** supports two views of its models: when the actors are contracted, the Strategic Dependency (SD) model hides the internal details of the actor, and when the actors are expanded, the Strategic Rational (SR) model describes an expanded view of the actors.

### **2.3.1 SD Model**

The SD model is structured in terms of a network of relationships and dependencies between actors. This model can help to identify the system stakeholders, analyze vulnerabilities and opportunities, recognize patterns of relationships and identify mechanisms to minimize vulnerabilities.

A SD model consists of a set of nodes and links. Each node represents an actor and each link between two actors represents a relationship, named Dependency, between them. The concepts used in the SD model are actors, goals, softgoals, tasks, resources and dependencies.

The actors represent people, organizations and systems involved in the context where the system is inserted. The goals represent the strategic interests of actors, that is, their intentions, needs or objectives to fulfill its role within the environment in which they operate.

There are four types of dependencies: softgoals, goals, tasks and resources (Figure 2-2). Softgoals also represent the strategic interests of the actors. In this case, these interests are of subjective nature, i.e., they are not measured in concrete terms, but are

generally used to describe the actors' desires related to quality attributes of their goals. The tasks represent a way to perform some activity, that is, they show how to perform some action to obtain satisfaction of a goal or softgoal. The resources represent data or information that an actor may provide or receive.

The actors involved in a dependency are named as: (i) "depender" or dependent actor, it depends on another actor who meets the dependency and (ii) "dependee" or the actor who is responsible for meeting the dependence.

Figure 2-2 shows an example of a SD diagram. The circles represent the actors that interact with each other through their dependencies, represented by links. Actor1 depends on Actor2 to meet their goals and softgoals dependencies. In this case, Actor1 is the depender and Actor2 is the dependee. On the other hand, the Actor2 depends on Actor1 to perform tasks and obtain resources. Therefore, in this case, the Actor2 is the depender and Actor1 is the dependee.

Thus, the SD model describes the external dependencies among actors of the organization and consists in a set of nodes and links among each other. These nodes represent the actors and each link indicates a dependency between two actors.

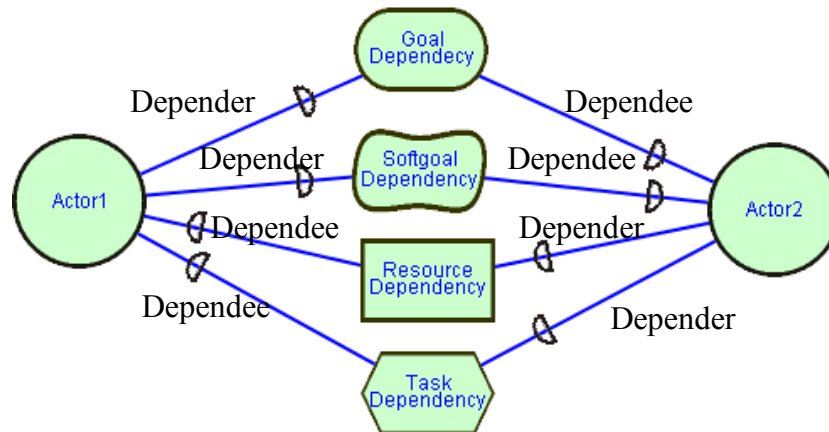


Figure 2-2 SD model

Although the SD model defines the actors and their relationships, it is not sufficient to support the process of suggesting, exploring and evaluating alternative solutions. The Strategic Rationale (SR) model is appropriate for capture the rationale.

### 2.3.2 Strategic Rationale Model

The SR model is used to: (i) describe the interests, concerns and motivations of participants in the organization, (ii) enable the evaluation of possible alternatives of process

definition, and (iii) to investigate, in more details, the reasons behind the dependencies between the various actors.

The same elements of the SD model (softgoals, goals, tasks and resources) are used in this model. They offer support to detail the motivations of the process. These elements appear within each actor, which is represented by a boundary, that is, a circle enclosing the analysis of the elements of this actor. Besides these elements, three new relationships are included: means-end, task-decomposition and means-end contribution. All these elements provide a framework to express the reasons involved in the process.

A means-ends relationship indicates a relationship between an *end* (purpose) and the *means* (tasks to be performed) to meet the end (see Figure 2-3). An end may be: (i) a goal to be achieved, (ii) a task to be performed, (iii) a resource to be produced, (iv) or a softgoal to be satisfied, a means to meet this end (purpose), which can be a task to be performed. Alternative tasks can be identified by Means-end relationships if one or more elements are intentional ways that contribute to the achievement of an end [Yu, 1995]. When there is more than one way to carry out an end, an OR (exclusive alternative) is assumed, indicating the different ways to achieve the end. This relationship suggests that there may exist other ways to achieve the same end, expressing the alternatives. Many alternative tasks can be identified in a means-end analysis of a goal. These tasks are or-exclusives, so at the end of the analysis, one of these tasks must be chosen and the other ones are not executed.

Therefore, the means-end relationship can be used as follows ()[Yu, 1995]: (i) Softgoals operationalize other softgoals; (ii) Tasks operationalize goals and softgoals; Tasks producing Resources and Tasks performing other tasks. (iii) Goals operationalize other goals. The case (ii) is considered more usual in *i\**, mainly, with goals and softgoals having operationalized by tasks.

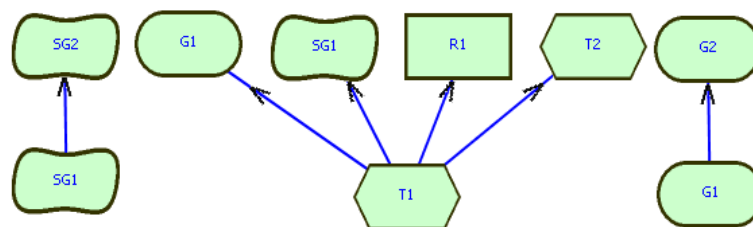


Figure 2-3 Use of *i\** means-end relationships

Therefore, an end can be a goal, a task, a resource or a softgoal. When an end is a goal and the means is a task, the task specifies how to achieve the goal through the

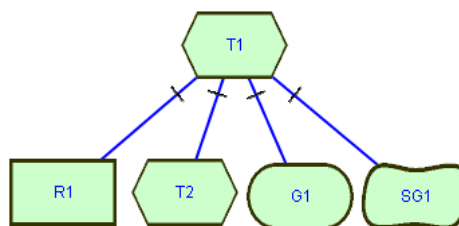
decomposition into components. When the end is a resource and the means is a task, the task specifies how to produce the resource through the decomposition into components.

The relationship between a task and a softgoal is a special case of means-end called contribution. The task is a means to satisfy the softgoal quality restrictions. This contribution can be positive or negative. So the relation is represented in a different way of the other means-end relations. It is represented with an extra attribute that identifies the sort of contribution of the tasks to the softgoal: satisfied (++), partially satisfied (+), conflict (-) and forbidden (--). presents an alternative way to represent the contribution labels [Grau, 2008].

**Table 2-1 Contribution link labels [Grau, 2008]**

| Contribution Link | Meaning                                                                                   |
|-------------------|-------------------------------------------------------------------------------------------|
| Make              | A positive contribution, strong enough to satisfy a softgoal.                             |
| Some+             | Either a make or a help contribution, a positive contribution whose strength is unknown.  |
| Help              | A partial positive contribution, not sufficient by itself to satisfy the softgoal.        |
| Unknown           | A contribution to a softgoal whose polarity is unknown.                                   |
| Break             | A negative contribution sufficient to deny a softgoal.                                    |
| Some-             | Either a break or a hurt contribution, a negative contribution whose strength is unknown. |
| Hurt              | A partial negative contribution, but not sufficient by itself to deny the softgoal.       |

In a task-decomposition relationship, a task is modeled in terms of decomposition of sub-components. These sub-components may be goals, tasks, resources or softgoals (see Figure 2-4). A decomposition of task in a goal indicates what this task wants to achieve this goal. When a task is decomposed in a sub-task, the sub-task defines an action to be performed. When a task-decomposition involves resources, means that an entity (physical or informational) will be used for this task. A softgoal serves as a quality goal for the task and guides the selection among alternatives in other decompositions of the task. In this case, there is an AND relation when a task is decomposed, meaning that the root task is performed when all their sub-elements are also performed. Figure 2-4 shows the possible use of task-decomposition.



**Figure 2-4 Use of i\* task-decomposition relationship**

Figure 2-5 shows an example of a SR model. The dotted circle represents the boundary of Actor2. The Goal element represents the main goal of actor2, which is refined by Tasks1 and Task2. These tasks are alternatives for the achievement of Goal. Task1 is refined by decomposition into two sub-tasks. Task 1-1 has a positive contribution for Softgoal, while Task 2 has a negative contribution. The analysis of the contribution helps the actor to decide which task must be executed.

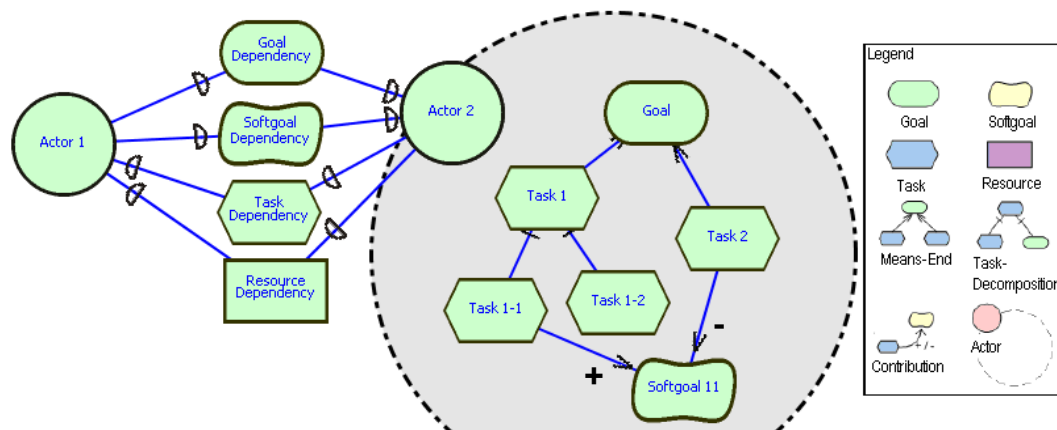
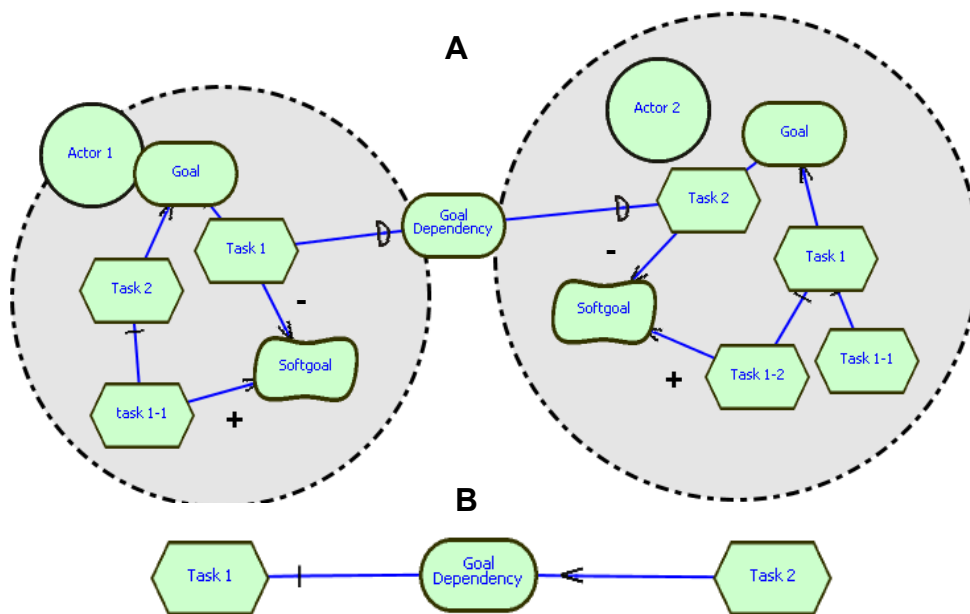


Figure 2-5 Example of SR model

Considering some issues related to SR models, specifically the actor's boundary, when the actor's boundary is expanded, its internal elements can be associated to a dependency to justify why that dependency is necessary or to indicate how the dependency is achieved. Thus, there are two situations: (i) an internal element depends on a dependency and (ii) a dependency depends on an internal element. For example, Figure 2-6-A highlights the internal element of actor2 (task 2) depending of goal dependency and the internal element of actor 1 (task 1) treating this goal dependency.

Other issue of  $i^*$  is that an outgoing dependency link from an internal task has the semantic of a task-decomposition link, i.e., the dependency is a sub-component of the task [Yu, 1995]. Besides, an incoming dependency link is also an implicit means-ends link, with the dependum being the “end” and the internal element being the means, i.e., to satisfy the dependency it s necessary to achieve the internal element [Yu, 1995]. For example, Figure 2-6-B highlights the meaning of dependency goal between internal elements of actor1 (task1) and actor2 (task2).



**Figure 2-6 Example of SR model with expanded actors (A) and the meaning of dependency between internal elements (B)**

To be compliant with the means-end constraints, the internal element must be a task. However, in some situations it is also allowed to use means-end relationships from Softgoal-to-Softgoal and Goal-to-Goal as presented in . Therefore, the internal elements for linking each depender and dependee actor must obey the rules of means-end and task-

decomposition links. Thus, the valid situations when we use an outgoing and incoming dependency link are presented in .

**Table 2-2 Valid situations outgoing dependency link and incoming dependency**

| <i>Depender</i> | <i>Dependency</i> | <i>Dependee</i> |
|-----------------|-------------------|-----------------|
| Task            | Task              | Task            |
| Task            | <i>Goal</i>       | Task            |
| Task            | Goal              | Goal            |
| Task            | Resource          | Task            |
| Task            | Softgoal          | Softgoal        |
| Task            | Softgoal          | Task            |

### 2.3.3 i\* Meta-model and Constraints

A meta-model defines the elements of a language and the relationships among them. Context conditions [Harel, 2004], also called static semantics, are constraints that define well formed allowed or forbidden i\* models. In our case, i\* has its own set of constraints, defining which constructions are not allowed. Some of these constraints, such as the cardinality of elements in a relationship, can be expressed in the metamodel. Other constraints can only be expressed using a constraint language, such as OCL [OMG, 2006].

Thus, to describe and understand the elements of the i\* [Yu, 1995] we have developed an i\* meta-model [Lucena et al., 2008]. In addition to this meta-model, a set of constraints are also needed to restrict the instances of the meta-model to valid i\* models. The meta-model is also useful due to the variety of versions that i\* has [Lucena et al., 2008]. Many research groups have adapted i\* to their specific necessities, resulting in a set of dialects. in our meta-model we (see fig 2-7) defined the elements of both SD and SR models in the same model. The constraints are declared in OCL [OMG, 2006] to show restrictions that cannot be expressed in this meta-model. shows the i\* meta-model and OCL expressions are presented in **Erro! Fonte de referência não encontrada..**



An InternalElement can be specialized in Resource, Goal, Softgoal, or Task. An InternalElement can also be a *means* to an *end* (another InternalElement) through a MeansEndLink. A Softgoal may have contributions from other InternalElement by a ContributionLink. A task may still be decomposed into other tasks, by decomposing the TaskDecompositioLink.

An Actor may enclose InternalElement, be a subtype of another actor (IsALink), or be part of other actors (Is-PartOf), or even be specialized in Role, Position or Agent. An Agent can play Roles (PlayLink) and can engage in a Position (OccupyLink). A Role can cover Positions (CoverLink).

The OCL Expressions 1-3 (Table 2-3) describe the different situations of means-end links presented in . While the OCL Expressions 4 and 5 (Table 2-4) describe valid situations of an outgoing dependency link and incoming dependency presented in Table 2.2.

**Table 2-3 OCL expressions to i\* meta-model**

|          | Element             | Constraint                                                                                                                                                                                                                                                                                                                   |
|----------|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| OC11     | <i>MeansEnd</i>     | <b>context</b> MeansEnd <b>inv:</b><br>self.means.ocIsTypeOf(Task) <b>implies</b><br>self.end.ocIsTypeOf(Goal) <b>or</b><br>self.end.ocIsTypeOf(SoftGoal) <b>or</b><br>self.end.ocIsTypeOf(Resource) <b>or</b><br>self.end.ocIsTypeof(Task)                                                                                  |
| O<br>CL2 | <i>MeansEnd</i>     | <b>context</b> MeansEnd <b>inv:</b><br>self.means.ocIsTypeOf(Goal) <b>implies</b><br>self.end.ocIsTypeOf(Goal)                                                                                                                                                                                                               |
| O<br>CL3 | <i>MeansEnd</i>     | <b>context</b> MeansEnd <b>inv:</b><br>self.means.ocIsTypeOf(SoftGoal) <b>implies</b><br>self.end.ocIsTypeOf(SoftGoal)                                                                                                                                                                                                       |
| O<br>CL4 | <i>DependerLink</i> | <b>context</b> DependerLink <b>inv:</b><br>self.depender.ocIsTypeOf(Actor) <b>or</b><br>self.depender.ocIsTypeOf(Task)                                                                                                                                                                                                       |
| OC15     | <i>DependerLink</i> | <b>context</b> Dependeelink <b>inv:</b><br>(self.dependee.ocIsTypeOf(Actor) <b>or not</b><br>(self.dependee.ocIsTypeOf(Resource))) <b>and</b><br>self.dependee.ocIsTypeOf(Goal) <b>implies</b><br>self.dependency.type=Goal <b>and</b><br>self.dependee.ocIsTypeOf(SoftGoal) <b>implies</b><br>self.dependency.type=SoftGoal |

## **2.4 Software Architecture**

Software Architecture area is a sub-area of Software Engineering that deals with notations, processes and approaches to develop artifacts related to software solution. This area emerged due to the difficulties inherent to the development software systems complex [Garlan, 1993]. Despite of being an established area in both academic and industry, there is no de facto standard terminology, neither are models languages and approaches. A list of software architecture definitions is presented in [SEI, 2009]. In general, the majority of the definitions emphasis that software architecture describes the structure of the parts of software (system) and their relations, and the principles and guidelines governing their design and evolution over time [Shaw et al, 1996][Bass et al, 2003][IEEE, 2000]. The abstract view of the system, allows a better understanding and communication between those involved in the development process. In addition, we it may be the basis for analysis of the system, allowing an early detection of design errors and possible failures that result in improvements in quality of software.

The role of software architecture in the development cycle is to serve as a bridge between requirements and an implementation [Clements, 2002]. Thus, its role is to bring the system requirements closer to implementation. There are several approaches, notations and tools to describe the architecture of the system. The process of architectural modeling is related to the documentation architectural models. These models are artifacts that capture design decisions on system architecture. To describe these models, we need a language that captures design decisions.

In this context, many architectural description languages (ADL) were developed with the purpose of describing architectural models. In [Medvidovic, 2000] a variety of ADLs are compared showing that they gave an explicit support to the concepts of components, connectors, interfaces and configurations. There are varieties of concepts that are captured in architectural models. Important terms when talking about architectural design are [Taylor, 2009]:

- i. Components: are the building blocks that encapsulate a subset system's features and data and restrictions to access them via a defined interface;
- ii. Connectors: are architectural building blocks that perform and regulates interactions between components;

- iii. Interfaces are points where components and connectors interact externally with other components and connectors;
- iv. Configurations: are a set of associations between specific components and connectors of software system architecture. Such associations can be captured by graphs whose nodes represent components and connectors, and whose vertices represent their associations;
- v. Reasoning: This is the information that explains why and for what purpose a particular architectural decision was made.

In addition to these concepts related to the notation used for architecture models, there are also other general concepts proposed by the IEEE in an attempt to standardize the used terms [IEEE, 2000], including views, viewpoints and architectural styles.

A view is a representation of a part of the system from the perspective of a set of related interests. Viewpoints define the perspective from which a view is considered. A view is an instance of a viewpoint to a specific system. Examples of viewpoints that are normally captured are: logical, behavioral, physical deployment. In this thesis, we are exploring the logical viewpoints that capture the logical entities (software) in a system and how they are interrelated.

Viewpoints are also called logical viewpoints Component-and-Connector (C&C) in architectural design approach proposed in [Clements, 2007]. When instantiating a logical viewpoint we will have a logical view of the architecture or a C&C view. The logical views are the first to be detailed in architectural design.

An architectural style is a collection of design decisions that are applicable in a given development context and a set of constraints in which the including can be used. The classic styles most frequently used were cataloged in [Shaw, 1996]: pipes and filters, layers, objects, blackboard repository.

In the next section, we will detail the architectural language used in this work.

## **2.5 Acme**

The proliferation of ADLs, each one with their own goals and their individual characteristics favoring one or more activities (reporting, verification, analysis) in different areas, provided the investigation of various aspects of architecture modeling. With the exposure of different features of modeling and how to exploit these characteristics, all

proposals helped to expand the understanding of the role that architectural modeling can take on software development.

Moreover, this way of working independently, as each ADL is concerned about a specific domain, makes it difficult to combine features several ADLs. Thus, the adoption of a new ADL often implies an expense of resource for installation of support tools and time to learn to use them effectively. Therefore, Acme [Garlan, 1997] was proposed with the primary goal of providing an interchange format for tools and environments for architectural development. Thus, the language provides the integration of several tools developed to support the various ADLs, providing a mechanism for the exchange of architectural information. Furthermore, Acme has four secondary objectives, as it:

- i. provides a representation scheme for the development of new tools for analysis and visualization of the structures of architecture;
- ii. enable a basis for the development of new ADLs, possibly domain-specific;
- iii. serves as a vehicle for the creation of conventions and standards for architectural information;
- iv. provides meaningful descriptions that parties interested in the system (stakeholders) can read and write.

To meet these goals, Acme supports the following elements:

- i. architectural framework, that is, the organization of the system against its constituent parts;
- ii. properties of interest, information about the system or its parts that provide an overview of the operation considering the functional and nonfunctional aspects;
- iii. types and styles, definition of classes and families of architectures.

The architectural structure in Acme is described by a small set of elements: Components, Connectors, Ports, Roles and Systems. Components are structures consisting of computer interfaces known as Ports. Ports define the interaction between the components and the environment. They are connected to ports of other components using connectors whose Porte role interfaces. Systems are abstractions that represent the configurations of components and connectors. Systems include a set of components, a set of connectors, and a set of attachments, which describe the topology of the system.

Attachments define a set of associations between ports and roles. Figure 2-9 shows the graphical representation of these Acme's elements.

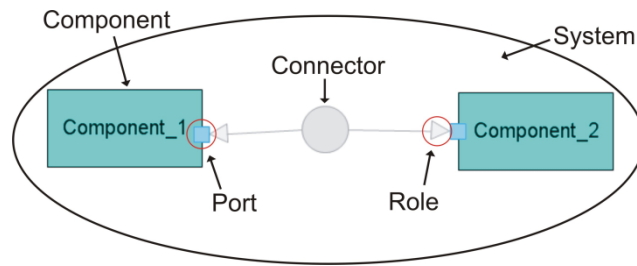


Figure 2-8 Graphical Representation of Structural Elements of Acme

In addition to these architectural elements, Acme supports Representations and Bindings. Representations are detailed descriptions of a given element (component, connector, port or role). These descriptions may correspond to alternative architectural views of an element or a detailed hierarchical decomposition of an element. Acme elements can contain more than one representation, each corresponding to a different conceptual view of the element or one form among several alternative ways to decompose the element. Bindings associate ports of a component to ports of their internal representation.

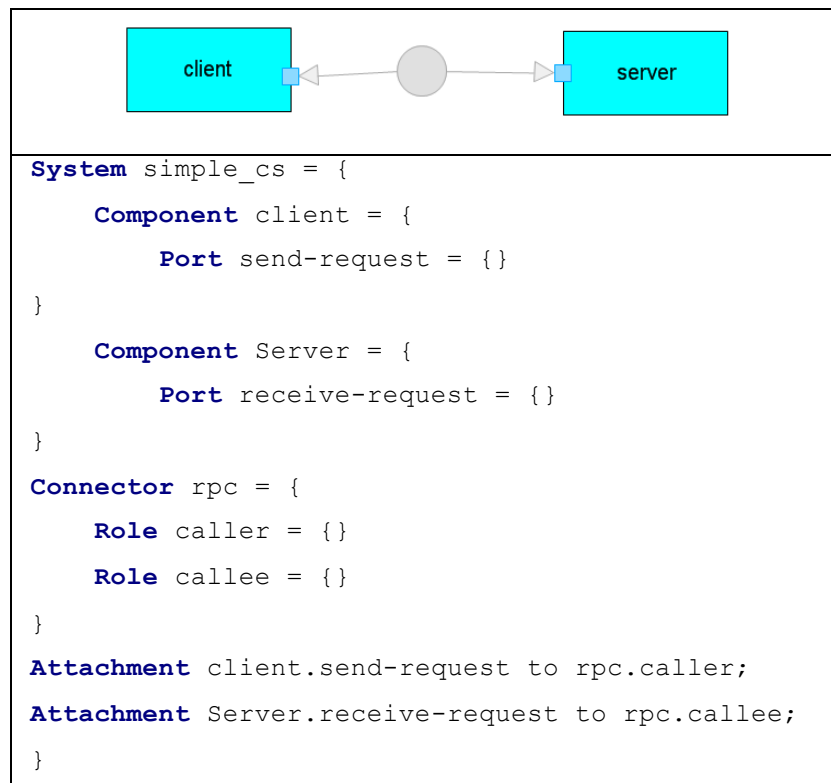


Figure 2-9 Example for the textual and graphical representation of Acme

Concerning the properties of interest, each of the structural elements defined in Acme (except attachments) can have its own set of properties. A name, an optional type

and a value define a property. Properties are seen by Acme as a value not interpretable. Properties can be used, for example, by analysis and transformation tools, or as mechanisms for extensibility. Figure 2-10 shows an example using the graphical and textual language of Acme.

Finally, Acme can define architectural styles through Family and Type constructors. An architectural style describes a family of systems in terms of a structural organization pattern, defining a vocabulary of component types and a set of constraints that indicates how the elements are combined [Shaw, 1996]. Type is used to define a vocabulary of abstract types of components, connectors, ports and roles; the Family constructor is used to define architectural styles in terms of abstract types of elements.

Figure 2-11 presents an Acme meta-model considering its main elements and relationships. The relationships between the elements are described below. Systems aggregate instances of elements that can be other systems, components, ports, connectors or roles. A system defines the configuration of an architectural solution. Each connector has two roles, which are attached to ports of components. A port of a sub-component (that is, part of an internal representation of a component) has a binds relationship with an external port of the parent component; this means that a component of an internal representation has a link with an external component.

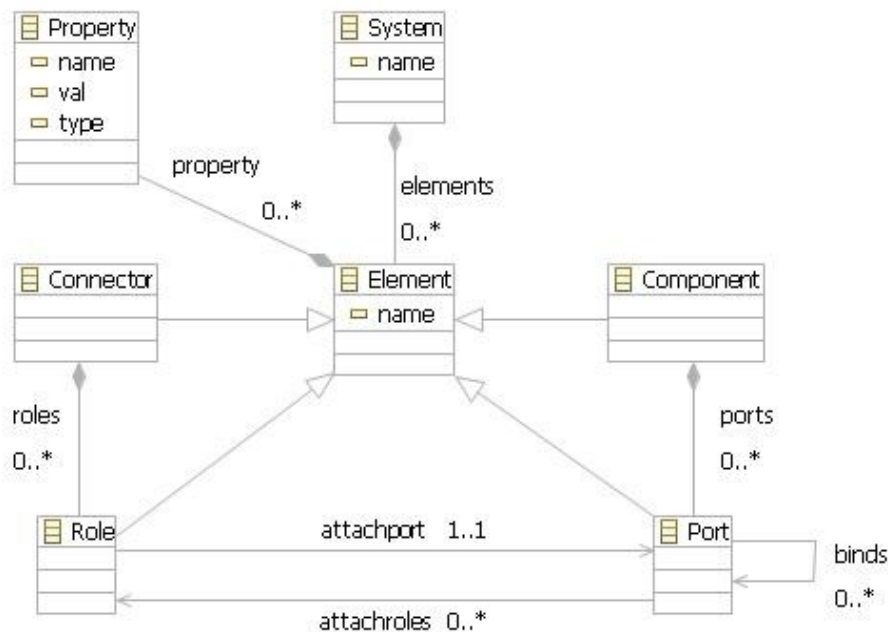


Figure 2-10 Acme meta-model

## **2.6 Summary**

This chapter presented the main concepts required for the understanding of our thesis. We outlined requirements and software architecture concepts and issues work. GORE approach is presented showing the main advantages in the context of this thesis.

The  $i^*$  modeling technique, a goal-driven requirements approach, is reviewed. In order to define the main  $i^*$  concepts a meta-model was also developed. Furthermore, a set of restrictions written in OCL was specified. Besides, the main concepts of software architecture used in this work were outlined. In particular, we presented the Acme ADL, discussing its elements and describing its metamodel.

It is important to notice that the meta-models of  $i^*$  and Acme were used to define the semantic of the involved elements of used languages in this thesis. These meta-models will be useful to describe the derivation rules used between these languages (See Chapter 3).

# 3 STREAM Approach

---

This chapter presents an approach for generating Acme architectural models from  $i^*$  requirements models. Besides, we present the main motivation used to elaborate this approach. It consists of the following activities: preparing requirements models, generating architectural models, selecting an architectural solution and refinement of architecture. To illustrate each activity, we rely on a running example related to e-commerce domain. The end of this chapter presents related work.

---

### 3.1 Introduction

In a software development process, software architecture models should be related to requirements models. The relationship between the Requirements Model and Architectural Model is usually ad hoc relying on the expertise of the software architect. STREAM contributes to reduce the gap that exists between what must be done (requirements) and how to do it (architecture).

A challenge in MDD area is to define the transformations rules. They guide the software developer through the process. In this thesis, we propose a strategy based on MDD that smoothly transforms the initial requirements model into the architectural model.

This chapter introduces our approach for generating architectural models from i\* requirements models. The approach was named *Strategy for Transition between REquirements models and AArchitectural Models* (STREAM). It consists of: (i) a set of transformation rules to prepare requirements models; (ii) a set of transformation rules to generate architectural models; and (iii) a systematic process to assist developers. The process includes four main macro-activities that will be detailed in the next sections: (i) Prepare Requirements Models, (ii) Generate Architectural Models, (iii) Select an Architectural Solution, and (iv) Refine Architecture.

Figure 3-1 presents the flow of these macro-activities using the SPEM notation [SPEM, 2007]. A macro-activity is called work-definition that will be detailed afterwards. Each work-definition aggregates a flow of activities.

In this thesis, the requirements model (source model) are represented using i\* notation and the architectural models (target models) are represented using Acme ADL.

Before detailing the STREAM approach, we will present the running example used through the chapter.

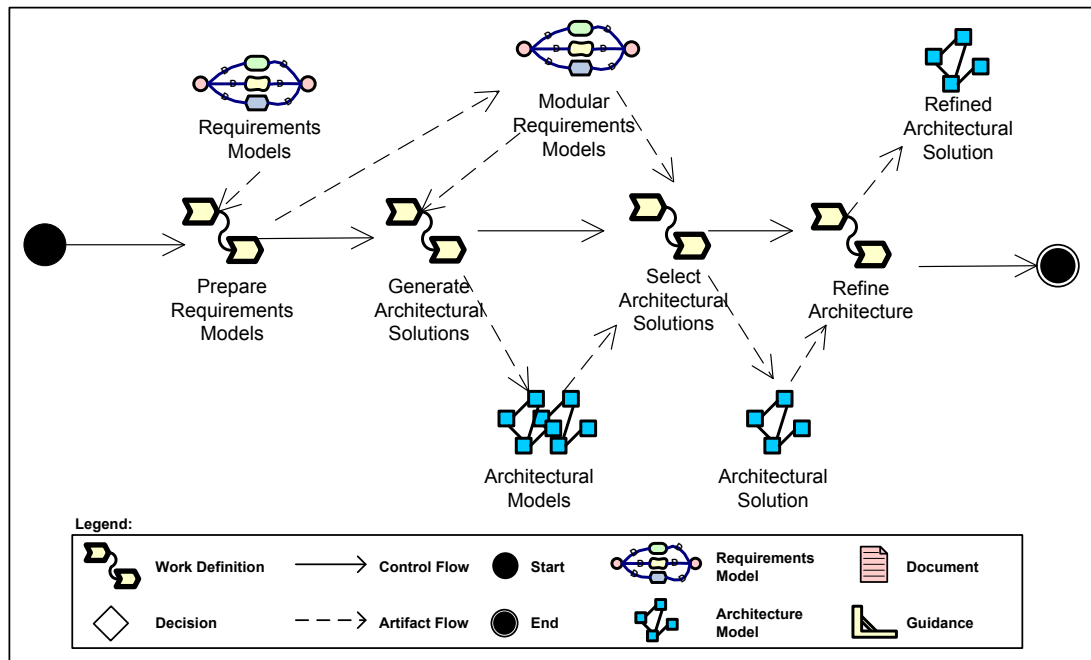


Figure 3-1 Overview of STREAM Process

### 3.2 Running Example

To illustrate our proposal we introduce the *Media Shop* example originally presented in [Castro et al, 2002]. *Media Shop* is a store that sells and ships different kinds of media items. To increase market share, *Media Shop* has decided to use the *Medi@* system, a business to customer retail sales front-end on the Internet (Figure 3-2).

A process to design *i\** models is composed by two phases: early requirements and late requirements. Early requirements are concerned with the understanding of a problem by studying an organizational setting; the output of this phase is an organizational model that includes relevant actors, their respective goals and their interdependencies [Yu, 1995]. Figure 3-2 shows the SD model for the Media Shop adapted of [Castro et al, 2002]. The main actors are *Customer*, *MediaShop*, *Bank Cpy* and *Media Producer*. Customer depends on *MediaShop* to fulfill his/her goal, *Buy Media Items*. On the other hand, *MediaShop* depends on *Customer* to *Increase Market Share* and satisfying its customers. Since the *Satisfied Customers* dependency indicates some qualitative goal to be achieved, it is represented as a softgoal. The Customer also depends on *MediaShop* to *Catalogue Consulting* (task dependency).

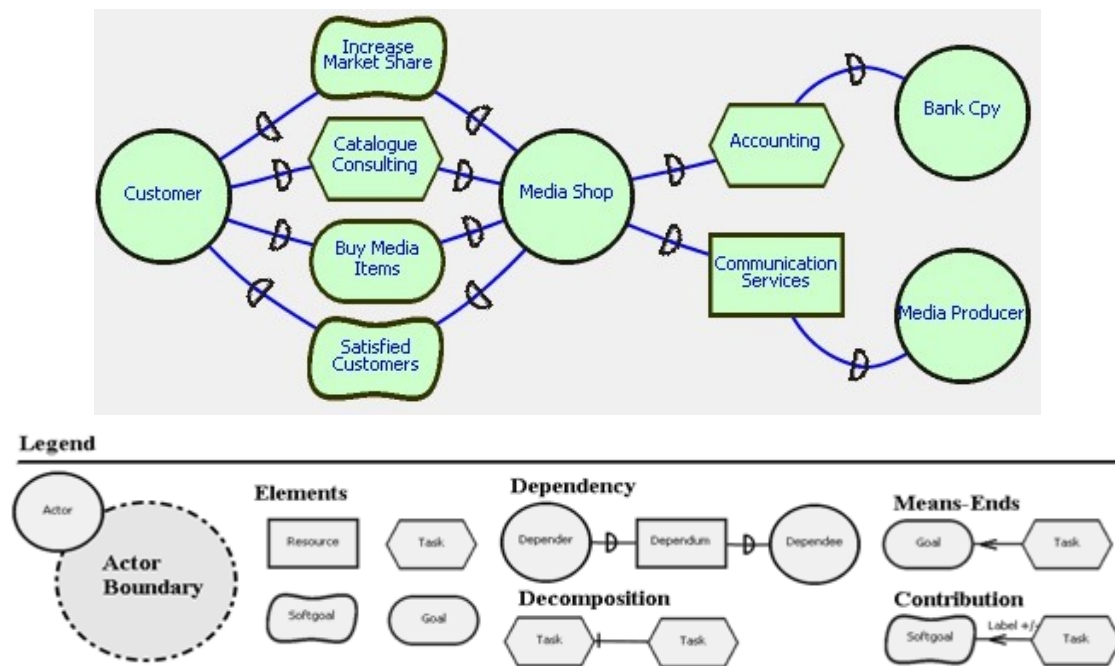
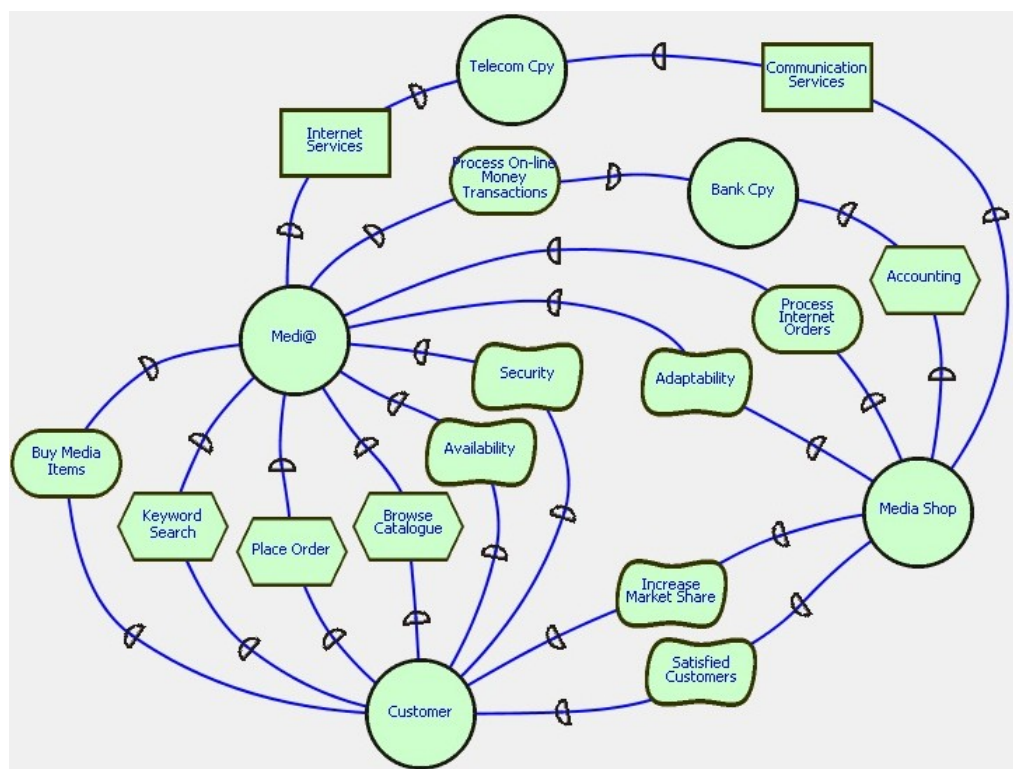


Figure 3-2 SD i\* Model to Media Shop

Note that at this stage a software actor is not present. Only after describing the motivations and intentions of actors it is possible to decide to include (or not) a computation actor (software system) as an alternative for satisfying the wishes of the stakeholders. Thus, Figure 3-3 shows the system-to-be, represented by Medi@ actor, within its operational environment. In the late requirements, the Medi@ actor is described and detailed with relevant functions and qualities.



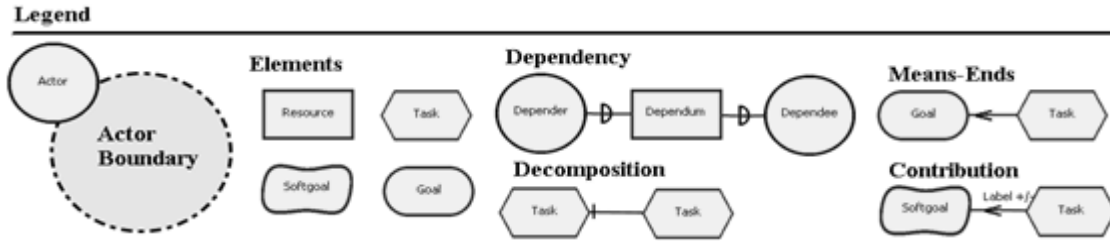


Figure 3-3 Medi@ actor in SD i\* Model to Media Shop

In our approach, we use as input model the SR models generated by late requirements phase, since they are more detailed and include expanded the descriptions of given actors. For instance, in our example, the *Medi@* actor is expanded (Figure 3-4). This actor include the root *Internet Shop Managed* task. It is firstly refined into softgoals (*Adaptability*, *Security*, *Availability* and *Attract New Customers*), tasks (*Produce Statistics* and *Adaptation*), and goals (*Internet Orders Handled* and *Item Searching Handled*). These elements are further refined by using task-decomposition, means-end and contribution links to describe the *Medi@* system requirements.

As presented in previous chapter, the task-decomposition links refine what should be done to perform a certain task. Thus, the relationship between the *Internet Shop Managed* task and their sub-elements implies that all these sub-elements need be performed. In a similar way, if a sub-task has a task-decomposition link, its sub-tasks/elements need to be performed. Thus, the *Produce Statistics*, *Adaptation*, *Shopping Cart*, *Catalogue Consulting*, and *Query Database* sub-tasks are also refined using a task-decomposition link.

The means-end links suggest that one intentional element can be offered as a means to achieve another intentional element. When there is more than one element as means, one of them will be chosen according possibly. The analysis of some quality attributes (softgoal). For example, check the relationship between the *Use Standard Form* or *Use Form with Encryption* tasks with the *Identification Details Collected* goal. The contribution links suggest how a task can contribute (positively or negatively) to satisfy a softgoal. For instance, the *Use Standard Form* or *Use Form with Encryption* tasks contribute to the *Security* softgoal.

Apart from these three types of relationships, there are external relationships between internal elements and other actors. For instance, the *Security* softgoal is related to an external *Security* dependency, while *Catalogue Consulting* task is associated to external *Browse Catalogue* dependency.

Having concluded the presentation of our example, we can now continue to present the four macro-activities of our STREAM approach.

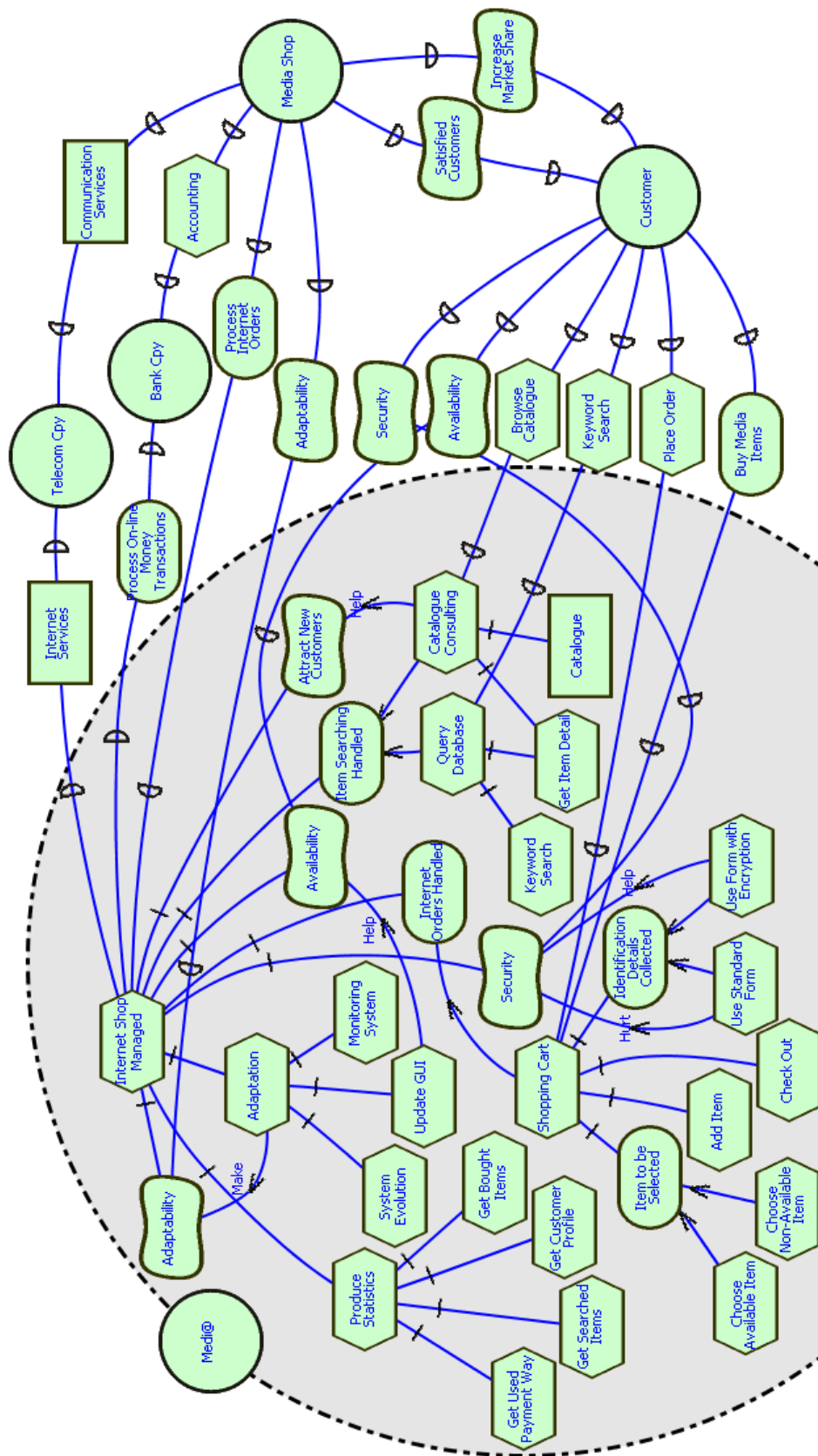


Figure 3-4 Medi@ SR Model

### **3.3 Prepare Requirements Models**

This section presents the motivation and general view of this activity, in addition, the three sub-activities are presented: Analyse of Internal Elements, Apply horizontal rules, Evaluate i\* models. Besides each sub-activities present the main guides used.

#### **3.3.1 Motivation**

i\* offers expressive models to capture social and intentional characteristics of a system organizational context. Actors and dependencies are core concepts. Actors have desires and needs. They are concerned with opportunities and vulnerabilities. Thus, i\* is used to explicitly capture stakeholders' motivations and rationale in a requirements model [Yu, 1995].

i\* models are often overloaded with information that captures features of both the system organizational environment and the software system itself. The more detailed i\* models are, the more complex they become. Hence, i\* models can become unnecessarily hard to read, understand, maintain and reuse.

Accordingly to Estrada [Estrada et al, 2006], the i\* modeling language can be evaluated using several aspects, from which we highlight modularity management. Modularity measures the degree to which the modeling language offers well-defined building blocks for building model. The building blocks should allow the encapsulation of internal structures of the model in a concrete modeling construct. This characteristic ensures that changes in one part of the model will not have to be propagated to other parts whereas complexity management measures the capability of the modeling method to provide a hierarchical structure for its models, constructs and concepts.

Although i\* incorporates a decomposition mechanism based on strategic actors, which could be used to improve modularization of i\* models, often the way in which this mechanism is used is not suitable to produce models that are easy to maintain and reuse. Current methods for i\* modeling, represent the rationale of an actor in a monolithic way. Sometimes several refinements are described in a scattering way, making it hard to visualize the boundaries of sub-graphs related to specific domains. This poor modularity compromise the management of the complexity of the models, an important pre-requisite for the adoption of i\* in industrial settings [Estrada et al, 2006].

We propose to take the modularity problem by means of a divide and conquer strategy, whereas the strategic actor construct can be used as a decomposition mechanism that divides complex actors into meaningful and manageable sub-actors. We claim that

some pre-processing can greatly improve the modularity of  $i^*$  models. In the next section, we present a work-definition that aims at the improving modularity of the expanded/refined system actor. It allows delegation of different issues of a problem, initially concentrated into a single actor, to new actors so that it is possible to deal with each of them separately, following the separation of concerns principle [Dijkstra, 1976].

### 3.3.2 Overview of Prepare Requirements Model

This work-definition is aimed at improving the modularity of  $i^*$  models. It is based on three activities:

- (i) Analysis of internal elements: this activity is related to identify which internal elements can be factored out from software actor.
- (ii) Application of model transformation rules: this activity use transformation rules to systematically move internal elements from software actor, identified in the activity (ii), to new actors.
- (iii) Evaluation of  $i^*$  models: this activity is used at the begin and the end of this process, it is concerned in evaluating  $i^*$  models to know if they need to be modularized;

Figure 3-5 presents an overview of this process. To carry out these activities, respectively, it is necessary to use:

- (i) heuristics to guide the decomposition of the software actor;
- (ii) a set of rules to transform  $i^*$  models;
- (iii) metrics to evaluate the degree of modularity of initial and final  $i^*$  models.

At the end of the Prepare Requirements Models process, the evaluation metrics are used to assess the level of modularity of the  $i^*$  generated models and compare with the metrics generated initially. Moreover, it is possible that the new actors generated by this process need to be modularized, thus new iterations can be necessary in order to ensure that all actors, that represent the software, are ready for the following activities.

This is a semi-automatic process, since some activities can be automated, such as the evaluation of models and the application of the rules. While the analysis of internal elements activity depends on requirements engineers and domain experts. In this case, heuristics are presented that help this activity.



For example, in the  $i^*$  model shown in Figure 3-6, which describes the details of the Medi@ actor and its relationships, the requirements engineer can identify whether those elements are related to the application domain (E-Commerce). The sub-graphs highlighted in Figure 3-6 (A, B, C and D) are independent of the E-Commerce domain and therefore can be moved from the software actor (*Medi@*) to another actor. After doing so, the final model is semantically equivalent to the original model. Figure 3-6 shows that the elements related to Production Statistics functionality, sub-graph A, could be used as part of a software of a different domain. The same reasoning could be applied to three sub-graphs featured in Figure 3-6 (sub-graphs B, C and D).

It is important to note that the choice of the elements to be factored out depends on the experience of the requirements engineer. For each selected element, according to the heuristics, a specific transformation rule is applied. These rules are presented in the following section.

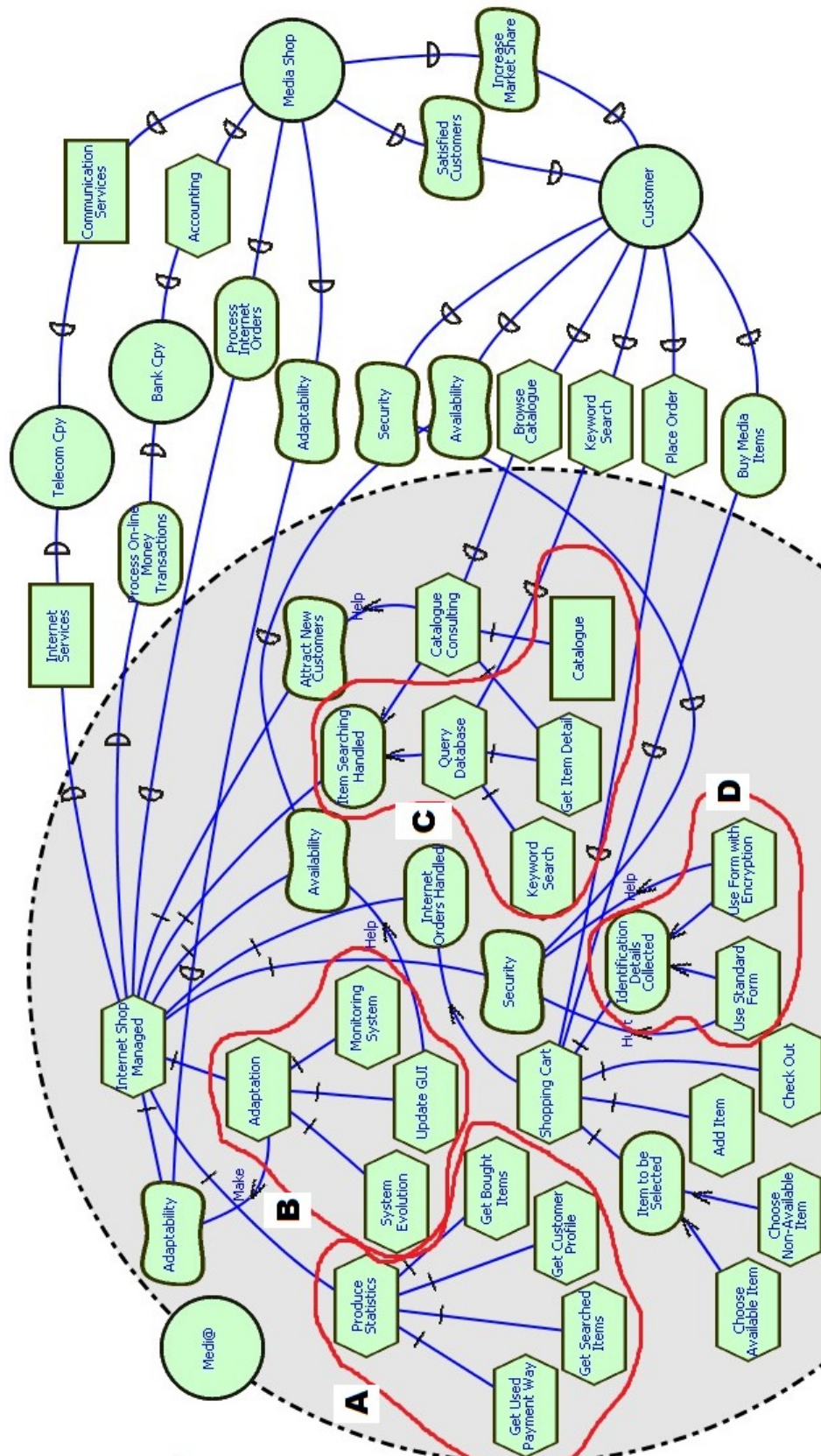


Figure 3-6 SR model of Medi@ with highlighted Internal Elements

### 3.3.4 Apply Horizontal Transformation Rules

In this activity, one of the transformation rules will be chosen considering the type of internal element to be moved and its relationship within the sub-graph. These transformation rules are applied to goals, softgoals and tasks elements, as these types can be refined justifying the transfer to another actor.

The transformation rules are intended to delegate internal elements of the actor other software (new) actors. This delegation must ensure that new actors and the original actor have a relationship of dependency. Thus, the original model and the final model are supposed to be semantically equivalent. The equivalence relation is based on the definition of dependency relationship (Section 2.3.2, Figure 2-6). A Dependency relationship (Actor1 – Dep – Actor2) is equivalent to a Task-Decomposition relationship (Element of Actor1 – Dep) and a Means-End relationship (Dep – Element of Actor2). It is beyond the scope of this proposal to show the equivalence between these models, more details can be found in [Yu, 1995].

At the end of this activity, the actors representing the software are easier to understand and maintain since they have fewer internal elements. Moreover, this approach promotes reuse of actors representing the software. Since new software actors are independent of domain application, they may be present in the requirements specification of other domains. Therefore, separating elements in other independent actors contributes to promote reusability and maintainability of the specification of software requirements.

#### 3.3.4.1 Horizontal Transformation Rules

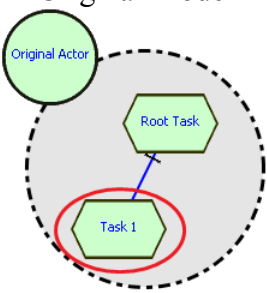
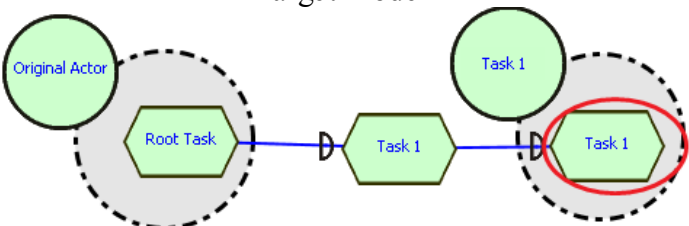
The transformation rules are structured as follows:

- (i) the name of the rule, for instance Transformation Rule 1 or TR1 (see Table 3-3);
- (ii) a figure to illustrate the context in which the original model can match before applying the rule;
- (iii) a figure to illustrate the resulting model after applying the rule;
- (iv) a description in which conditions the rule should be applied, and
- (v) a description of the effects produced by the rule.

The transformation rules are TR1 and TR2 (including their variations), TR3, TR4 and TR5 address, respectively, elements that participate in task-decomposition and means-end relationships (Table 3-3 and Table 3-5). While TR3 and TR4 are more restrictive rules, that treat they address situations involving softgoals and elements, respectively, that

participate in different sub-graphs (Table 3-7 and Table 3-8). Besides, TR3 is a corrective rule since it handles inconsistent  $i^*$  models produced by the other rules. TR3 is necessary because TR1 and TR2, including their variations can produce invalid models. Hence, the corrective rule deals with inconsistencies that can be generated by the rules TR1 and TR2, including their variations. Thus, from a inconsistent initial model, the set of proposed rules when applied may generate a consistent  $i^*$  model. Rule TR5 renames actors created by previous rules TR1 and TR2.

**Table 3-1 Transformation Rule to move elements in a task-decomposition link (TR1)**

| TR1 – Move a sub-element in a task-decomposition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                                                                        |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|
| <p>Original Model</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                     | <p>Target Model</p>  |
| <p><b>Pre-condition:</b> A root task of a graph is decomposed into sub-tasks that do not share any sub-element through task-decomposition or means-end links.</p>                                                                                                                                                                                                                                                                                                                                                                                           |                                                                                                        |
| <p><b>Effects:</b> The sub-graph that is independent from application domain (e.g. the sub-graph whose element is the Task 1) is moved from the original actor to a new actor. This new actor has the same name of the element of the transferred sub-graph. This element (Task 1) will be replicated as a dependency relationship of same type relating the original actor, as the depender, and the new actor, as the dependee. Besides, all the existent external dependencies with this transferred sub-graph will be transferred to the new actor.</p> |                                                                                                        |

TR1 address sub-element of a decomposition (Table 3-3). This transformation rule was defined based on the equivalence of outgoing actor dependence [Yu, 1995], presented in Section 2.3.2, where an outgoing dependence on an internal element has the same meaning of task decomposition. In this transformation, the intentional elements of dependencies entering the new actor are replicated in this new actor, as in the version of  $i^*$  used in Tropos [Bresciani, 2004]. Observing the valid elements between dependers and dependees, presented in Table 2-2, we have that on the dependee side the valid elements are tasks, goals and softgoals. That is, the possible sub-elements that may be moved by TR1 are tasks, goals, and softgoals. However, we propose that softgoals remain in the main actor since they can receive contributions from different intentional elements. Hence, if the softgoal is moved to another actor then many new dependencies would also have to be moved to the target model.

Notice that the precondition of TR1 (and some of the following rules) restricts its application to elements that do not share sub-elements with other element. This is because we have first to decide who keeps the shared elements. In these cases, we may lift the restriction applying TR4 first.

Table 3-4 presents a variation of the TR1, TR1-v1, showing a goal as element in a task-decomposition and the target model after applying this rule. Note that the number of actors to be created will depend on the number of alternatives related to the goal in a means-end relationship. Each dependency between an original actor and a new actor receives a label (A1, A2, etc) to highlight that it corresponds to a specific alternative of the means-end link.

Table 3-5 presents TR2, where the sub-graph to be moved has the root as a “means” in a means-end relationship. In this case, this sub-graph (alternative graph) is moved to a new actor. The root element of the original graph is replicated both inside the original actor and as a dependency from the element inside the original actor to the root of the sub-graph moved to the new actor.

**Table 3-2 Transformation Rule to move goals in a Task-decomposition link (TR1v1)**

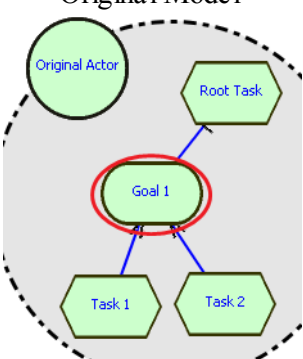
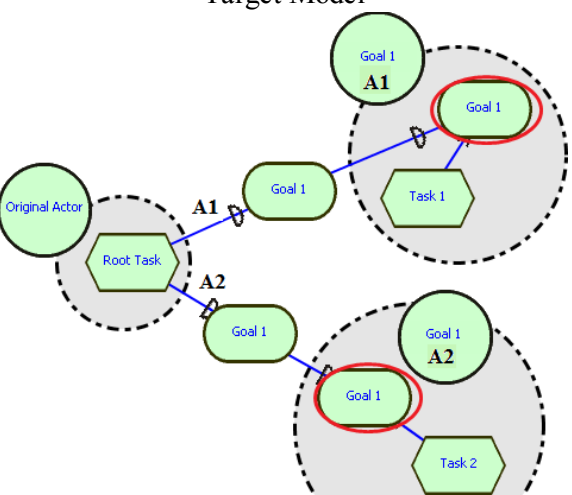
| TR1v1 – Move a sub-goal in a task-decomposition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                                                                          |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------|
| <p>Original Model</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | <p>Target Model</p>  |
| <p><b>Pre-condition:</b> A root task of a graph is decomposed into a sub-element (goal) that is also root of sub-graphs (alternatives), but do not share any sub-element through task-decomposition or means-end links. If there are more than one alternative each one will be moved to a new actor with respective goal.</p>                                                                                                                                                                                                                                                                                       |                                                                                                          |
| <p><b>Effects:</b> The sub-graph that is independent from application domain is moved from the original actor to a new actor. This new actor has the same name of the root of the transferred sub-graph. This root element will be replicated as a dependency relationship of same type relating the original actor, as the depender, and the new actor, as the dependee. This new dependency will receive a label related to alternative of means-end link to identify the corresponding. Besides, all the existing external dependencies related to this transferred sub-graph will be moved to the new actor.</p> |                                                                                                          |

Table 3-6 presents a variation of the TR2, TR2-v1, showing a goal as a means in a means-end relationship and the target model after applying this rule. Note that this situation, although odd, can occur in i\* models (see [Yu, 1995]). This situation will be handled as in TR2. However, in i\* models based in i-Star Guide [Grau et al, 2008] there is only task as means in a means-end relationship, thus, this variation does not apply.

**Table 3-3 Transformation Rule to move alternatives (means) in a means-end link (TR2)**

| TR2 - Move “means” sub-graph in a means-end link                                                                                                                                                                                                                                                                                                                                                                                                          |                     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|
| <p>Original Model</p>                                                                                                                                                                                                                                                                                                                                                                                                                                     | <p>Target Model</p> |
| <p><b>Pre-condition:</b> A root goal of a graph is an “end” in one or more means-end relationships and at least one of its “means” is a sub-graph (alternative) that does not share any element (through task-decomposition or means-end links) with other sub-graphs (independent sub-graph).</p>                                                                                                                                                        |                     |
| <p><b>Effects:</b> Each independent sub-graph (alternative) will be moved to a new actor with the same name of the sub-graph’s root. The roots of the transferred sub-graphs must be replicated inside the original actor to keep the original mean-end relationship. From each of these replicated elements, a new dependency of the same type and name must be created from the original actor to the root of the sub-graph moved to the new actor.</p> |                     |

**Table 3-4 Transformation Rule to move a goal (means) in a means-end link (TR2-v1)**

| TR2-v1 - Move a goal (means) in a means-end link                                                                                                                                                                                                                                                                                                                                                                                                                        |                     |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|
| <p>Original Model</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                   | <p>Target Model</p> |
| <p><b>Pre-condition:</b> A root goal of a graph is an “end” in one or more means-end relationships and at least one of its “means” is a goal and its sub-graph that does not share any element (through task-decomposition or means-end links) with other sub-graphs (independent sub-graph).</p>                                                                                                                                                                       |                     |
| <p><b>Effects:</b> Each independent sub-graph (alternative) will be moved to a new actor with the same name of the sub-graph’s root-goal. The root-goal of the transferred sub-graphs must be replicated inside the original actor to keep the original mean-end relationship. From each of these replicated elements, a new dependency of the same type and name must be created from the original actor to the root-goal of the sub-graph moved to the new actor.</p> |                     |

After applying rules TR1 and TR2 (including their variations), it may happen that the final model is not in accordance with the restrictions of the i\* metamodel (Chapter 2). For example, a contribution relationship in the original model can result in a relationship that crosses the actor's boundary (see Table 3-7 - original model). In this case, we need to use a corrective rule such as TR3 (Table 3-7). TR3 suggests the replication of a softgoal involved in a contribution relationship within the original actor and the creation of a dependence softgoal from the new actor to the original actor. This rule was defined to preserve the information about contribution relationships, maintaining the consistency of i\* models as proposed in [Horkoff, 2007]. Observing the Table 2-2 of Chapter 2, it is noticed that only task elements can be present on the depender side. Therefore, we modified the way the softgoal is treated in [Horkoff, 2007]. Instead of having a dependency going from the softgoal as suggested in [Horkoff, 2007], the request of the dependency comes from the root element, which in all rules will always be a task. This occurs because only the tasks may have softgoals as sub-element since there are no softgoals as means to goals but softgoals as sub-element of tasks [Yu, 1995].

**Table 3-5 Transformation Rule to move a contribution link crossing the actor's boundary**

| TR3 - Contribution link crossing actor's boundary                                                                                                                                                                                                                                                                                    |                                                 |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------|
| <p style="text-align: center;">Original Model</p>                                                                                                                                                                                                                                                                                    | <p style="text-align: center;">Target Model</p> |
| <p><b>Pre-condition:</b> There are elements such as task, goal or softgoal contributing to the achievement of softgoals that are going out of the actor's boundary.</p>                                                                                                                                                              |                                                 |
| <p><b>Effects:</b> A softgoal element, with the same name, must be created inside the actor from where the contribution link goes out. The contribution link is kept inside that actor. A softgoal dependency, with the same name, must be created from the root element (task 2) to the softgoal present in the original actor.</p> |                                                 |

TR4 is applied when the sub-graphs to be moved has a shared sub-element with other sub-graphs (Task 2, in the original model presented in Table 3-8). One of the previous rules (TR1 or TR2) needs to be applied to move a sub-graph (in this case the sub-graph with Task 1 as root) to another actor. But there is a shared element; then for such cases the rule TR4 is used. This rule suggests that a policy priority is established to define where the shared element (Task 2) will remain. Thus, the following cases are analyzed:

- Check the types of root elements in the sub-graphs that share the element. The shared elements will remain with the sub-graph that has the root with type of highest priority. The priority for the type of the root element is goal, softgoal and task (in this order);
- If all the roots of sub-graphs are of the same type, verify the position of the roots of sub-graphs that share the element. The root of the sub-graph that is closer to the most general root will keep the shared element;
- If all the roots of the sub-graphs have the same type and are on the same level relative to the root of the general graph, then the shared element will stay with the sub-graph that remains in the original actor.

According to TR4 new dependencies will be created to maintain the connection between the elements that will be separated. Another alternative to do not need to create new dependencies is replicating the shared element, in the case the shared element is a leaf element.

**Table 3-6 Transformation Rule to move shared elements**

| TR4 - Move a shared sub-element                                                                                                                                                                                                                               |                     |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|
| <p>Original Model</p>                                                                                                                                                                                                                                         | <p>Target Model</p> |
| <p><b>Pre-condition:</b> There is an element (Task 2) that is shared by different sub-graphs (through task-decomposition, contribution or means-end link) and at least one of these sub-graphs is moved to a new actor (after applying rules TR1 or TR2).</p> |                     |
| <p><b>Effects:</b> The shared element will remain in the sub-graph whose root element has the highest priority. The relationships with the remaining elements will be replaced by dependencies, as stated by the rules TR1, TR2, TR3, and variations.</p>     |                     |

Many cases could be represented by variations of this rule showing specific situations. Table 3-8 shows an example of a more general case when there are different types of elements, all at the same level of sub-graph, associated to the same sub-element (Task 2). Only when the root element is a task, its sub-elements can be different types, such as, goal, softgoal and task.

After applying the previously rules it is necessary to rename the new actors created according to the function that they achieve. Thus, a rule to rename the new actors is presented in Table 3-9.

Almost all the rules consider the restrictions related to the i\* model, ensuring that a dependency goes out from a task (depender) while tasks, goals or softgoals are placed in the dependee side according to Table 2-2 (Chapter 2). However, there are cases where these restrictions do not apply, as the rule TR2-v1, when a goal is a means that is moved to another actor. These cases are rare because in most situations a task as means is a means-end relationship [Grau et al., 2008].

**Table 3-7 Transformation Rule to change the name of New Actor**

| TR5 – Rename new actors                                                                                                                                                                                |                                                                                     |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| Original Model                                                                                                                                                                                         | Target Model                                                                        |
|                                                                                                                       |  |
| <b>Pre-condition:</b> There is a new actor created after applying rules TR1 or TR2 that received the same name of a task or a goal.                                                                    |                                                                                     |
| <b>Effects:</b> The name of actor will be changed according to the functionality of this actor changing the name of action, represented by a verb, to a name of the subject that executes this action. |                                                                                     |

Our rules could be used for refactoring purpose, but they are not complete enough to handle all situations where a refactoring could be appropriate. In the scope of this thesis, our rules aims to increase the modularity of the models i\*.

We now can show how the rules could be applied to our running example. Measurements are performed to compare the original i\* models against the final (transformed) models. Hopefully an increase in the modularity will be achieved.

### 3.3.4.2 Applying Transformation Rules in Medi@ example

Our first step is to select the suitable transformation rules to be applied to the identified elements in Figure 3-6 The choice depends on the type of relationship that these elements have with the remaining elements. For instance, TR1 should be applied to the *Produce Statistics* task because it is a sub-element of *Internet Shopping Managed* task, which belongs to a task-decomposition relationship. Other sub-graph where TR1 is applied is the *Adaptation* task sub-graph. However, the use of TR1 can generate contribution links that exit the actors' boundaries. In this case TR3 is applied to replace contribution links

crossing actors' boundaries by softgoal dependencies, aiming at maintaining the consistency with i\* syntax and semantics. Thus, the *Adaptability* and *Availability* softgoals appear in the final model.

On the other hand, for the sub-graph with *Database Querying* task is a means to *Item Searching Handled* goal. Hence, TR2 should be applied. It moves sub-elements that are alternatives to another actor. However, before applying TR2, TR4 must be applied, since there is a shared element (*Get Item Detail* task) related to *Catalogue Consulting* task and *Query Database* task.

Furthermore, TR1v1 is applied to the D sub-graph, generating two new actors where each one corresponds to specific alternative. Thus, the final model receives labels that identify each alternative related to the Dsub-graph. Moreover, TR3 is also used to replace contributions links by softgoals dependencies.

After all rules are applied, the new actors are renamed according to TR5. Thus, Produce Statistics stays Statistics Producer actor, Query Database stays Database actor, Identification Details Collected stays Detailed identification Collector, and Adaptation was renamed to Profile Manager. Figure 3-7 shows the resulting model after applying the transformation rules introduced in Section 3.4.4 are applied.

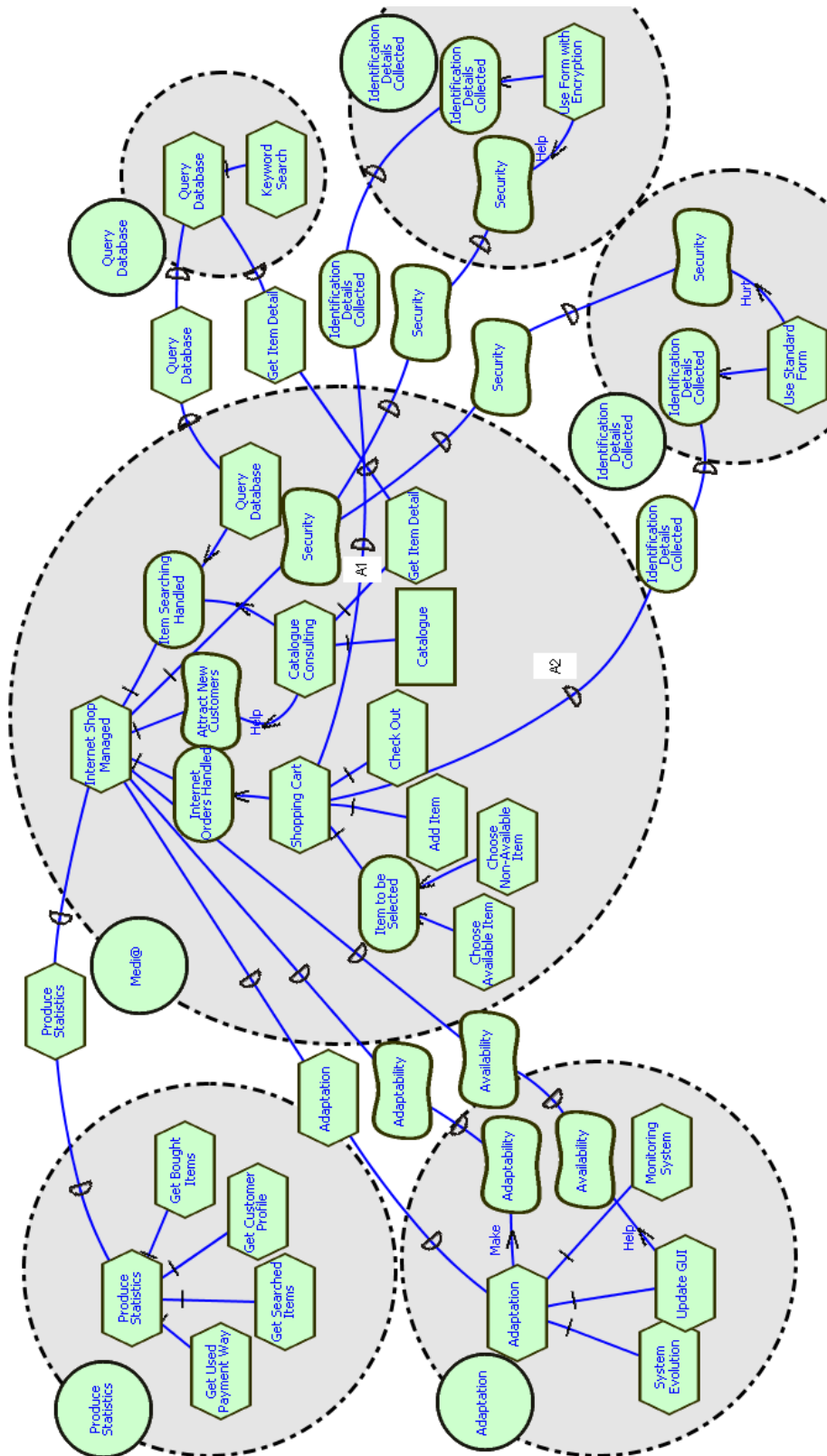


Figure 3-7 i\* SR model after applied transformation rules

### 3.3.5 Evaluate i\* Models

The process starts with the evaluation of the initial i\* models using modularity metrics. Since experimental results have shown that i\* models often suffer from lack of modularity [Estrada et al, 2006], we adapted the metrics of modularity proposed by [Sant'Anna et al, 2003] for i\* models. According to [Sant'Anna et al, 2003], the metrics related to the modularity of software artifacts are separation of concerns (SoC), coupling and cohesion.

The metric for separation of concerns evaluates whether the concerns are spread among multiple components and if there is an interconnection between concerns. The coupling metric verifies how many components depend on a particular component. The cohesion metric determines how much internal information of a component does not contribute to a specific concern. Table 3-1 shows the principal metrics used by this thesis work.

The adaptation of the modularity metrics introduced by [Sant'Anna et al, 2003] depends on our definition of the concepts to be used. In this thesis, the concepts of components, operation and LOC (line of code) are mapped to the elements of the i\* models. shows this correlation.

To maintain the compatibility between what the metric does and the sort of elements that it uses, each metric above was adapted according to the correlation between the elements of i\* and the elements of Sant'Anna's framework. Thus, based on these correlations, the following metrics were used to evaluate the i\* models:

(i) Concern Diffusion over Intentional Elements (CDIE), adapted from CDO: counts the number of Intentional Elements (e.g., Tasks, Goals) whose main purpose is to contribute to the achievement of a concern and the number of others intentional elements that require them;

(ii) Concern Diffusion over Graphic Element (CDGE), adapted from CDLOC: counts the number of transition points for each concern through the refinement of the Actor (e.g. Actor's rationale). Transition points are points in the refinement where there is a change (switch) of concern;

(iii) Coupling between Actors (CBA), adapted from CBC: counts the number of other Actors to which a specific actor is coupled;

(iv) Lack of Cohesion in Intentional Elements (LCIE), adapted from LCOO: measures the lack of cohesion of an actor in terms of the amount of intentional elements that do not contribute to the main concern.

**Table 3-8 Metrics from Sant'Anna's Framework [Sant'Anna et al, 2003]**

|                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Metrics of SoC   | <i>Concern Diffusion over Operations (CDO)</i> . CDO counts the number of primary operations whose main purpose is to contribute to the implementation of a concern.                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|                  | <i>Concern Diffusion over LOC (CDLOC)</i> . CDLOC counts the number of transition points for each concern through the lines of code. The intuition behind it is that they are points in the program text where there is a “concern switch”. For each concern, the program text is analyzed line by line in order to count transition points. The higher the CDLOC, the more intermingled is the concern code within the implementation of the components; the lower the CDLOC, the more localized is the concern code.                                                                                                                            |
| Coupling Metrics | <i>Coupling between Components (CBC)</i> . CBC is defined for a component (class or aspect) as a total of the number of other components to which it is coupled.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Cohesion Metric  | <i>Lack of Cohesion in Operations (LCOO)</i> . This metric measures the lack of cohesion of a component. If a component C1 has $n$ operations (methods and advices) $O_1, \dots, O_n$ then $\{I_j\}$ is the set of instance variables used by operation $O_j$ . Let $ P $ be the number of null intersections between instance variables sets. Let $ Q $ be the number of non-empty intersections between instance variables sets. Then: $LCOO =  P  -  Q $ , if $ P  >  Q $ , $LCOO = 0$ otherwise. LCOO measures the amount of method/advice pairs that do not access the same instance variable. As such, it is a measure of lack of cohesion. |

**Table 3-9 Correlation between i\* elements and concepts of Sant'Anna's metrics framework**

| I* element                                                   | Concept    | Correlation                                                                                                                                                                   |
|--------------------------------------------------------------|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Actors                                                       | Components | Actors and Components are first class elements that are used as the main structure of i* and programs, respectively.                                                          |
| Tasks, Goals, Softgoals, Resources                           | Operations | The intentional elements of i* (tasks, goals, resources and softgoals) define the internal structure of actors, so as operations define the internal structure of components. |
| Tasks, Goals, Softgoals, Resources, Actors and relationships | LOC        | Every graphic element (intentional elements, actors, and relationships) of an i* model is associated to LOC.                                                                  |

So, the process begins with the evaluation of the initial i\* model. If the measurements indicate some modularization problem then we could try to improve the modularization performing the second activity Analyze Internal Elements.

### 3.3.5.1 Evaluating i\* models of Medi@ example

In order to evaluate the results of the modularization we applied a set of metrics presented in *Evaluate i\* models* activity. These metrics measure properties related to the modularity such as the concern diffusion, coupling and cohesion. The Medi@ model was measured before and after the use of the transformation rules. Recall that the metrics rely on the concern concept. In our running example the concerns of interest are the ones highlighted by *Analyze Internal Element* activity (Query Database, Produce Statistics, Identification Details Collected and Adaptation) as well as the main concern: *Sell Media products*.

Table 3-10 Coupling and cohesion metrics

| Metrics                                            | Medi@ | Modular Medi@ |
|----------------------------------------------------|-------|---------------|
| CDIE (Concern Diffusion over Intentional Elements) | 22    | 10            |
| CDGE (Concern Diffusion over Graphic Elements)     | 3     | 0             |
| CBA (Coupling between Actors)                      | 0     | 4             |
| LCIE (Lack of Cohesion in Intentional Elements)    | 10    | 3             |

Table 3-10 present the results of to the measurement. Note that there was a reduction of diffusion in both CDIE and CDGE metrics. The reduction in the CDIE metric (from 22 to 10) can be explained by the subtraction of elements from the Medi@ actor. Since the concerns that were diffused in the actor Medi@ are now grouped in other actors, the elements that remain in the actor Medi@ are no more affected by these concerns. The value of CDIE after the use of the transformation rules corresponds to the dependencies that were added. These dependencies are now required to link the internal elements to the concerns. Similarly, the CDGE metric presents reduction of diffusion. The point of intersection that switches from a concern to another disappear after applying the transformations. Hence, the value of CDGE varies from 3 to 0.

On the other hand, the value of coupling increases from 0 to 4 after the use of the transformation rules. The new configuration of the actor Medi@ is coupled to new 4 actors that did not exist before.

Last but not least, the LCIE metric is greatly reduced. The number of elements that do not contribute to the main concern of actor Medi@ decreases from 10 to 3. Almost all these elements (e.g.: tasks *Produce Statistics*, *Adaptation*, *System Evolution* and *Monitoring System*), which do not contribute to the main concern were modularized in new actors.

The results presented in Table 3-9 show that this modularization approach contributes to the improvement of modularity reducing the diffusion of concerns and the lack of cohesion. The increasing of coupling is a negative side effect that it was expected

since the nature of this approach contributes to increase the number of actors in the model.

Figure 3.8 depicts the final  $i^*$  model. Observe that the actors that interact with the software actor tend to be smaller. Note that the choice and use of transformation rules is in essence a first step towards the initial architecture. Some key early design decisions are made. Observe that the *MediaShop* and *Customers* actors, that interact directly with the system, should be taken into account for the generation of the initial architecture. They will represent interface components of an early architectural design. However, for simplicity we consider only *MediaShop* actor that it is renamed to *MediaShop Interface* (see ).

Having concluded that the horizontal phase, which consists of the preparation of a more modular requirements model, we can now move to the next phase of the STREAM approach. Namely, the production of an early architectural solution.

We have now concluded the preparation of the requirements model. However, before moving to the generate architectural solution work definition, it is recommended to evaluate the modularity of the new model. Hence, we re-run the Analyze Internal Elements activity. If the evaluation indicates poor modularity, the rules of transformation will have to be applied. In the Medi@ example, the new model contains cohesive actors so one iteration was sufficient to improve the modularity.

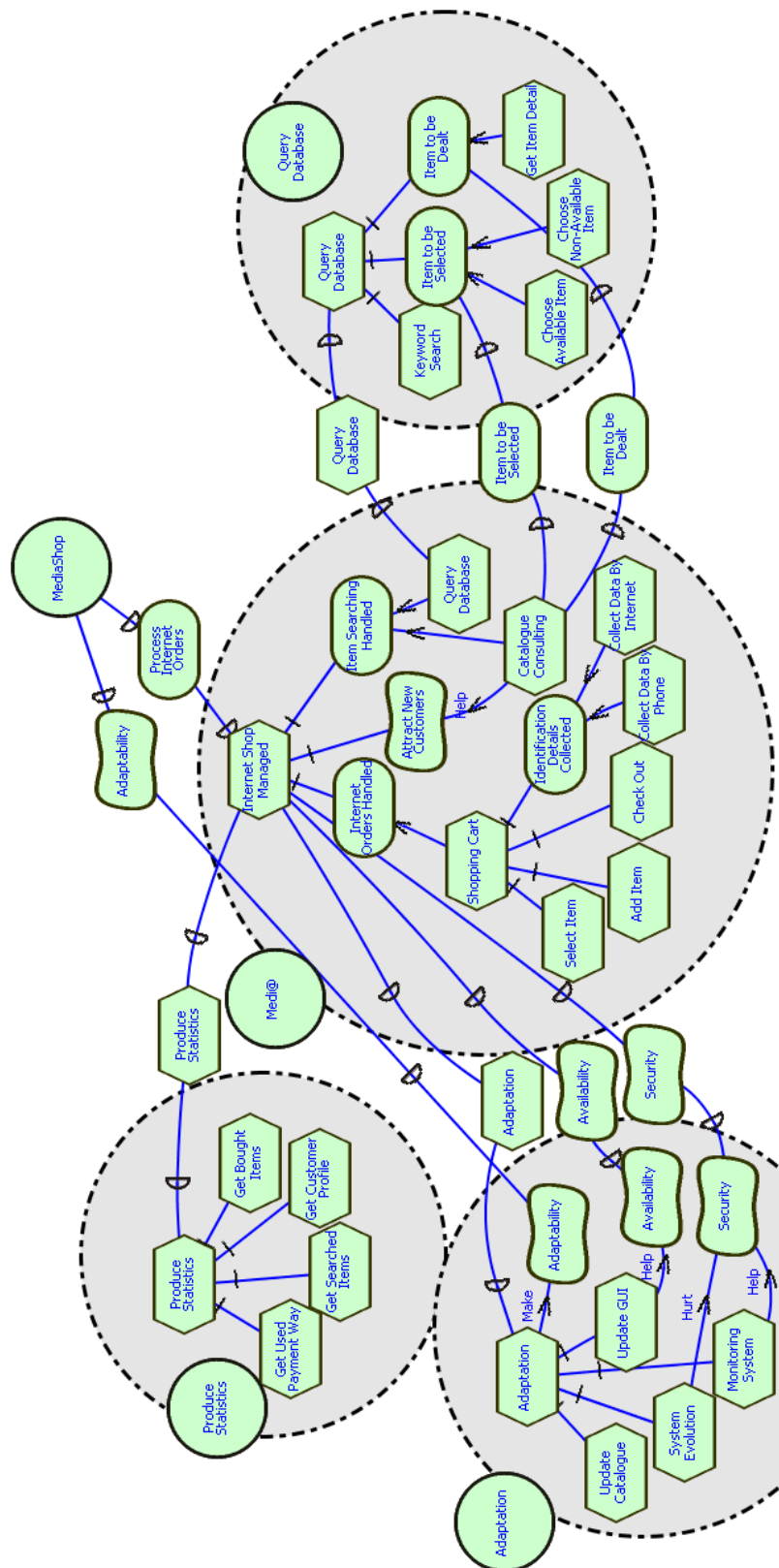


Figure 3-8 Modular SR i\* model with MediaShop actor

### 3.4 Generate Architectural Solutions

We start to establish the vertical transformation rules considering only actors and dependencies to map i\* elements to Acme elements. Figure 3-9 shows a generic mapping without considering the type of dependency between actors and dependencies, in i\* (Figure 3-9-a), to components and connectors in Acme graphical (Figure 3-9-b) and the corresponding textual language (Figure 3-9-c).

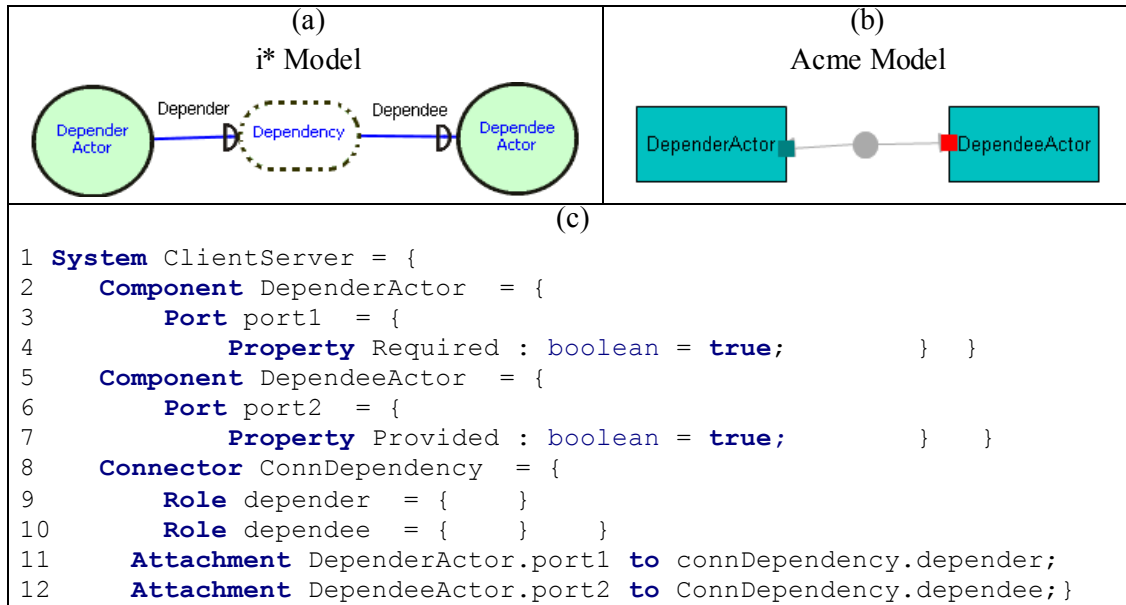
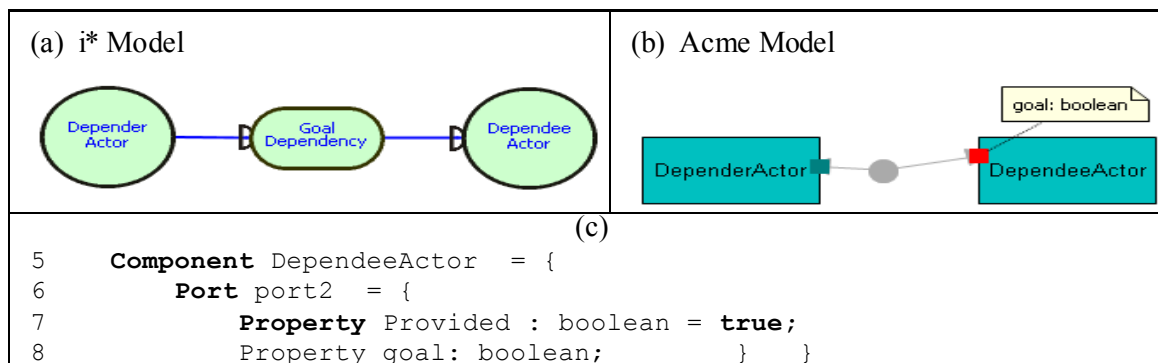


Figure 3-9 Mapping a generic dependency between i\* actors to Acme

A component in software architecture is a unit of computation or a data store that has interacting points (ports) to interact with external world [Medvidovic&Taylor, 2000]. An actor in i\* is an active entity that carries out actions to achieve goals by exercising its knowhow [Yu, 1995]. The actor representing the software establishes a correspondence with modules or components [Grau et al, 2007]. In addition, an actor may have as many interactions points as needed. Hence, an actor in i\* can be represented in terms of a component in Acme (Figure 3-9-a).

Connectors are architectural building blocks that regulate interactions among components [Medvidovic and Taylor, 2000]. In Acme, connectors mediate the communication and coordination activities among components. In i\*, a dependency describes an agreement between two actors playing the roles of depender and dependee, respectively [Castro et al, 2003]. Thus, we can represent a dependency as an Acme connector. Interfaces are points of access among components and connectors. However there are no ports in i\*, but points where dependencies interact with actors. Depending on role of dependency we can know when an actor is a depender or a depended upon actor.

Hence, the roles of depender and dependee are mapped to connector roles (Figure 3-9-b). Thus, we can distinguish between required ports (where the actor is a depender) and provided ports (where actor is a dependee). For instance, Figure 3-9 shows the use of Required property (Figure 3-9-c, line 4) and Provided property (Figure 3-9, line 7) indicating the direction of communication between the *DependerActor* and *DependeeActor* components. Therefore, in *i\**, a depender actor depends on a dependee actor to accomplish a type of dependency. In *Acme*, a component requires that another component carries out a service and the requisition of this service is expressed by a required port, while the result of this service is available at a provided port. Thus, a connector allows the communication between these ports. A component offers services to another component using provided ports and a component requires services using its required port. Similarly, each actor will be represented as component and the dependencies will be represented as connectors. Besides, each interaction between actors will be mapped to provided port or required port, according to the direction of dependency between actors.



**Figure 3-10 Mapping a goal dependency to *Acme***

In *i\**, the type of dependency between two actors describes the nature of the agreement established between these actors. There are four types of dependency: goals, softgoals, tasks and resources. Each type of dependency will define different architectural elements in connectors and in ports that play their interfaces. A goal dependency is mapped to a Boolean property related to a provided port of the component that offers this port (Figure 3-10-c, line 8). This property represents a goal that this component is responsible to fulfill by using a provided port.

The property is of Boolean type in order to represent the goal satisfaction (true) or denial (false). Applying this goal dependency mapping in *Media* example implies that *Database* component will be added a Boolean property in the provided port.

A task dependency represent that an actor depends on another to execute a task [Yu, 1995] . A task describes or involves processing [Grau, 2007], while a port in Acme corresponds to external interfaces of components and offer services. Hence, a task dependency is mapped directly to a provided port of component that offers this port (Figure 3-11-c, line 6). In our Media example the *Statistics Producer*, *Database* and *Profile Manager* components will have their provided ports associated, respectively their tasks dependencies.).

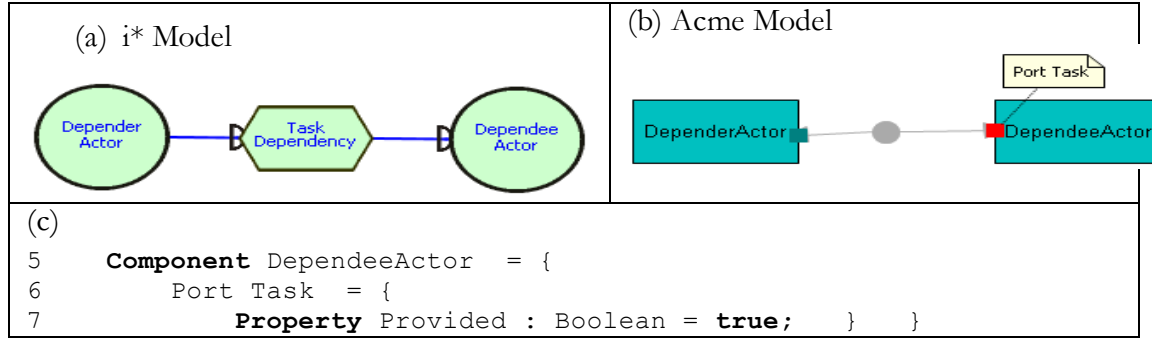


Figure 3-11 Mapping a task dependency to Acme

In a resource dependency, an actor depends on another actor to provide information. Therefore, a resource dependency is mapped to a return type of a property of a provided port (Figure 3-12-c, line 8). This return type represents the type of the resulting product that an operation related to some service that the component is responsible to perform. Note that a task that is generated by an actor in a component is mapped to an element inside a port. In our Media example, we do not have resource dependency related to software actors (see Figure 3-9).

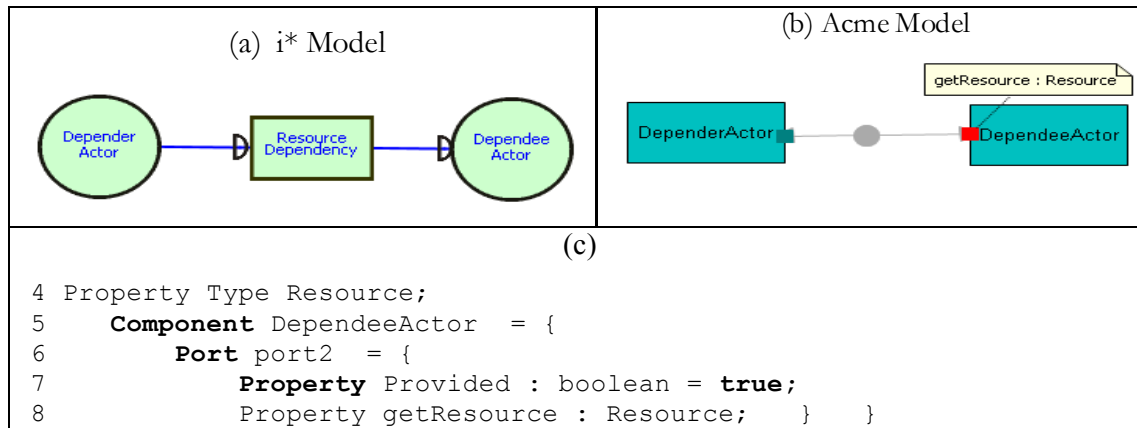


Figure 3-12 Mapping a resource dependency to Acme

A softgoal dependency is similar to goal dependency but its satisfaction is relative. There are degrees of satisfaction. A softgoal can be fully satisfied, partially satisfied, totally denied, partially denied, etc. A softgoal is related to a non-functional requirement that will be operationalized by a task or refined into more specific softgoals. Hence, a softgoal

dependency is mapped to a property with enumerated type present into the port that plays the dependee role of the connector (Figure 3-13, line 9-c). This enumerated type is used to describe the degree of satisfaction of the softgoal. For the Media example, the *Profile Manager* component has two provided ports that interface with *Availability* and *Security* connector. Each provided port would have a property softgoal defined as enumeration type.

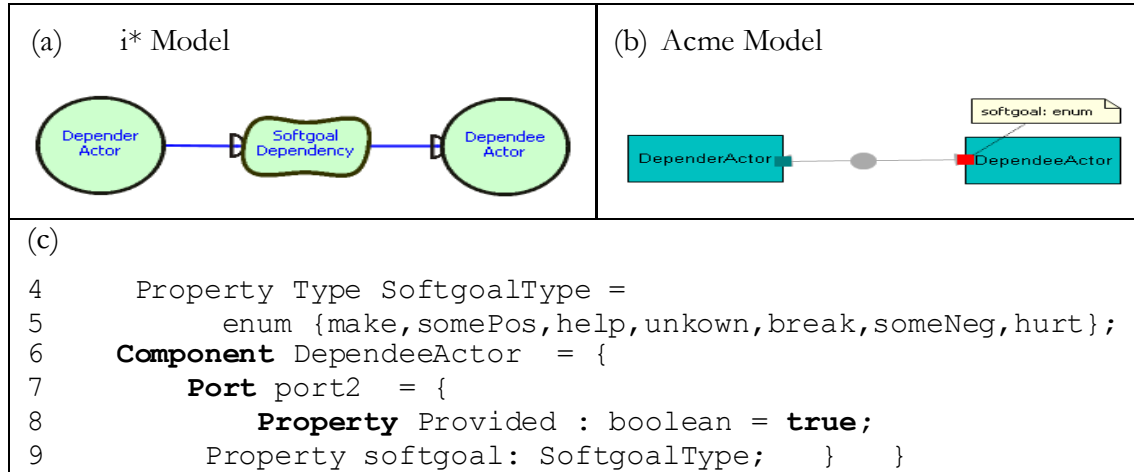


Figure 3-13 Mapping a softgoal dependency to Acme

Note that the Media Component aggregates in the same required port three connectors, this occurs because the solicitation of these services comes from of the same element (Internet Shop Managed task, see Figure 3-9). At the end of this activity, we have two architectural solutions (Figure 3-14) where *Details Identifier* component holds two representations to describe the two alternatives of Figure 3-9. These solutions will be analyzed in the next activity. Each of these solutions satisfies specific softgoals (non-functional requirements) from requirements models (Figure 3-9).

Acme graphical language uses labels to highlight added elements used in textual part. However, using AcmeStudio tool, information of types of dependency only is presented in textual language. Hence, we have added notes in the generated architectural model as it is shown in Figure 3-14. Appendix A presents the complete textual representation of Figure 3-14 for Media Example.

From these architectural solutions described in Acme, it is possible to choose a specific architectural solution, based on the softgoals present in the SR model. Thus, these softgoals can influence the decision about which architectural solution to use. This decision is taken during the *Select an Architectural Solution* work definition.

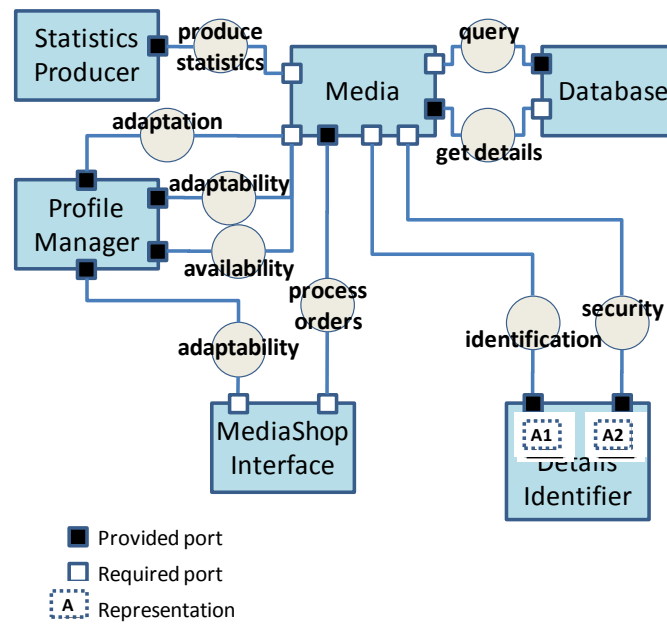


Figure 3-14 Architectural Solution with component using representations

### 3.5 Select an Architectural Solution

After generating the full set of possible architectural solutions, the architect needs to decide the most appropriate one for a given context. To support this activity, the softgoals from requirements models are analyzed. They will provide valuable information to assist in the decision.

Thus, this activity receives as input the modular requirements model and the set of possible architectural solutions, both generated by previous activities. These artifacts are required because the set of architectural solutions need to be analyzed to check how they satisfy the softgoals from the requirements model.

For instance, in the Media system, we have two architectural solutions (see Figure 3-14) generated from the SR i\* model (see Figure 3-9). However, in a more complex situation where more alternative components are available, many more architectural solutions could be considered if we had more alternative components. Each solution tries to optimize the satisfaction of a set of specific softgoals. For example, the first solution tries to fulfill the security, availability and adaptability softgoals, while the second solution aims at satisfying the availability and adaptability softgoals, but neglects the security softgoal. In this activity, the information related to softgoals present in requirements model are visualized in the generated architectural solution. Thus, as in the initial architecture, the non-functional requirements that were chosen to be achieved by the software are also attached to the architectural solution.

In our example, the Security, Availability and Adaptability softgoals are represented in the architectural solution that describe the A1 representation (Figure 3-14). They will guide the application of a proper architectural pattern according to the Refine Architecture activity.

### 3.6 Refine Detailed Architecture

Having produced an early architecture solution we can try to refine it. In this thesis we rely on some architecture patterns for this activity. The components of the architectural model will be refined according to the non-functional requirements presented in the original architectural solution. In the previous activity, we have only indicated the need for the architecture and its components to address concerns, which are usually non-functional requirements, such as security, accuracy, availability, and usability. This activity is divided in two sub-activities: application of architectural patterns and use of architectural refinement patterns. Representative architectural patterns are used in the first sub-activity, while the architectural refinement patterns are used in the second sub-activity. The architectural patterns are applied globally to all architecture, while the architectural refinement patterns are applied inside components (internal configuration) if it is necessary to treat non-functional requirements in internal components. Figure 3-16 shows the main artifacts (consumed and produced) of *Derive Detailed architecture* activity.

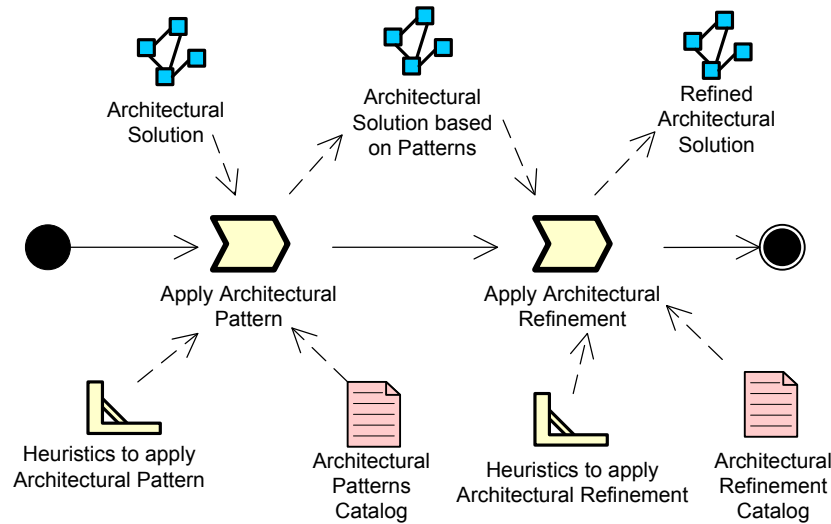


Figure 3-16 Activities Flow of Derive Detailed Architecture Work-definition

Architectural patterns used in our thesis are Client and Server, Layers, Model-View-Controller and Blackboard. These patterns were chosen because they are used to achieve a subset of non-functional requirements such as Performance and Reliability [Mairiza, 2010]. The architectural refinement patterns used in this work are Maintain Availability, Maintain

Availability, Maintain AccurateData, Maintain FaultTolerantCommunication and Maintain Interoperability. Besides, according to a study of the use of architecture patterns presented these patterns as the most popular patterns [Harrison, 2008].

Since the derivation of detailed architecture is highly influenced by non-functional requirements [Mairiza, 2010] [Bass, 2004], we need to review them.

### **3.6.1 Non-functional requirements**

In this thesis we often refer to non-functional requirements, such as (i) Availability, (ii) Usability, (iii) Maintainability, (iv) Performance, (v) Reliability, and (vi) Security. In general, the non-functional requirements are divided in two groups: operational, such as performance that , and non-operational, such as maintainability.[Reza, 2005]

Availability is concerned with system failure and its associated consequences. A failure occurs when the system no longer delivers a service consistent with its specification. Such a failure is observable by the system's users, either humans or other systems. Among the areas of concern are how system failures are detected, how often system failure may occur, what happens when a failure occurs, how long a system is allowed to be out of operation, when failures may occur safely, how failures can be prevented, and what kinds of warnings are required when a failure occurs.

Usability specifies the end user interactions with the system and the effort required to learn, operate, prepare input, and interpret the output of the system. Usability is concerned with how easy it is for the user to accomplish a desired task and the user support the system provides.

Maintainability or modifiability describes the capability of the software product to be modified that may include correcting a defect or make an improvement or change in the software. It involves decomposition of system functionality and the programming techniques used within a component. In general, a system is modifiable if changes involve the minimum number of decomposed units.

Performance requirement is related to how fast the system will run, that is, throughput, latency, number of clock cycles spend finishing a task. They specify the capability of software product to provide appropriate performance relative to the resources needed to perform full functionality under stated conditions. The principal attributes for this requirement is response time, space, capacity, latency, throughput, computation, execution speed, transit delay, workload, resource utilization, memory usage. Performance is about timing. Events (interrupts, messages, requests from users, or the passage of time) occur, and the system must respond to them. There are a variety of characterizations of

event arrival and the response, but performance is concerned with how long it takes the system to respond when an event occurs.

Reliability specifies the capability of software product to operates without failure and maintains a specified level of performance when used under specified normal conditions during a given time period.

Security is concerned about preventing unauthorized access to the system, programs, and data. Security is a measure of the system's ability to resist unauthorized usage while still providing its services to legitimate users. An attempt to breach security is called an attack and can take many forms. It may be an unauthorized attempt to access data or services or to modify data, or it may be intended to deny services to legitimate users [Bass et al, 2003].

Usually, the quality of a system can never be achieved in isolation. Therefore, satisfying one quality may contribute or compromise to satisfy another non-functional requirement [Chung, 2000]. The Table 3-6 presents the relationship among these non-functional requirements [Chung, 2000][Reza, 2005]. This table shows how the non-functional requirements that are in the horizontal affect those in the vertical. Thus, we can observer that availability compromise reliability and security, for instance, but helps usability.

**Table 3-11 NFR vs. NFR based on [Reza, 2005]**

| NFR           | NFR | Availability | Usability | Modifiability | Performance | Reliability | Security |
|---------------|-----|--------------|-----------|---------------|-------------|-------------|----------|
| Availability  |     | unknown      | Make      | Help          | Help        | Hurt        | Hurt     |
| Usability     |     | Make         | unknown   | Make          | Break       | Break       | Break    |
| Modifiability |     | Make         | Make      | unknown       | Hurt        | Break       | Some+    |
| Performance   |     | Make         | Make      | Make          | unknown     | Make        | Break    |
| Reliability   |     | Make         | Make      | Make          | Help        | unknown     | Make     |
| Security      |     | Make         | Hurt      | Make          | Hurt        | Some+       | unknown  |

In our example, the Security, Availability and Adaptability non-functional requirements need to be achieved in the architectural solution (Figure 3-14). Looking at the Table 3-5, we have the relationship between Availability and Security, where Availability compromise (Hurt) the Security, however Security contributes strongly (Make) to Availability. Besides, according to the requirements models of Medi@ presented in [Castro, 2002] the Security has priority over the others. Adaptability, in turn, is a non-functional requirement related to Self-Managed Systems [Kramer, 2007] and treating this requirement is out of the scope of this work.

The patterns used in this work regard achieving these non-functional requirements.

### 3.6.2 Architectural Patterns

As previously stated, architectural patterns are an established tool for designing and documenting software architectures. They are proven approaches to architectural design. Many architectural patterns have been identified, based on extensive experience with real systems [Buschmann, 1996], [Shaw, 1996]. In this section, we present some patterns used to achieve the non-functional requirements inspired by [Lamsweerde, 2003]. We define a systematic way for their use. The architectural patterns used in our approach are Layers and Model-View-Control. The description of patterns consists of a (i) Context, (ii) Problem, (iii) Solution, (iv) Structure, (v) Representation, (vi) Impacts on Non-functional requirements, and example of use. The context shows the principal domains and systems that the pattern is applied, according to [Buschmann, 1996] and [Harrison, 2008]. The problem, solution and configuration describe information about these aspects of pattern, according to [Buschmann, 1996] and [Avgeriou, 2005]. Impacts on non-functional requirements are based on [Buschmann, 1996] and [Harrison, 2007]. To represent the patterns, we use Acme. The Medi@ example is used to show how we apply a pattern to the Media System. These patterns are present in the sequence.

#### **Pattern: Layers**

**Context.** A large system that requires decomposition.

**Problem.** Architecture configurations that merge high and low-level services, where high-level services rely on low-level services, compromise understandability and maintainability. Mixing different levels of tasks and services also turns hard to switch to different platforms, to divide the development among teams of programmers, and to substitute parts by alternative implementations.

**Solution.** The system is divided in layers and placed one in the top of each other. Each level requires services from the level below it and provides services for the level above. This hierarchy establishes the level of abstraction; the highest the layer is the highest is its abstraction level.

**Structure and Representation.** The main characteristic of the Layer pattern is that the services provided by a layer are only used by the layer above. A layer may be a complex entity consisting of different components that may use the services of each other. Each layer provides an interface that allows isolation from the services that it provides and their implementation.

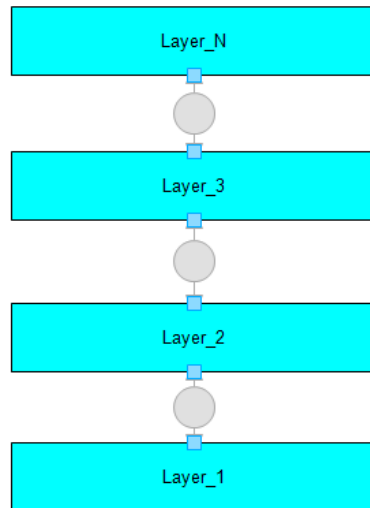


Figure 3-XX Layer architectural pattern represented in Acme

**Impacts on Non-functional requirements.** This pattern contributes positively to security, modifiability and reliability, since it defines well-established interfaces between layers. A liability is the efficiency, in data intensive application, going through many layer can be inefficient. Usability is not affected by this pattern.

#### **Pattern: Model-View-Control (MVC)**

**Context.** An application, like a Web based system, offer multiple user interfaces. Each user interface represents all or part of some application data.

**Problem.** Changes to the data should reflect automatic and flexibly to all the different user interfaces. It should be also possible to modify any one of the user interfaces, without affecting the application logic associated with the data.

**Solution.** The system is divided into three different parts: Model, Views and Controller. The Model encapsulates some application data and the logic that manipulates that data, independently of the user interfaces; one or multiple Views display a specific portion of the data to the user and the Controller associated with each View receives user input and translates it into a request to the Model. Views and Controllers constitute the user interface. The users interact through the Views and their Controllers, independently of the Model, which in turn notifies all different user interfaces about update. Figure 3-17 presents MVC pattern described in Acme.

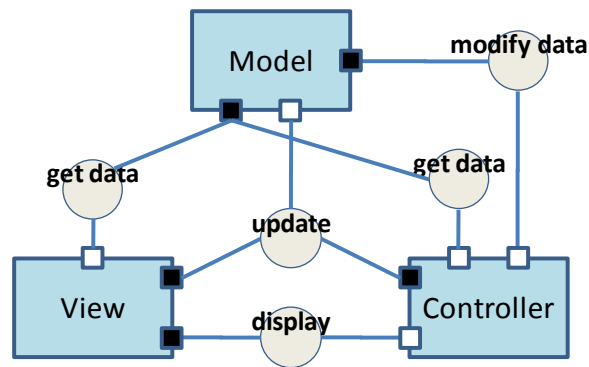


Figure 3-17 MVC pattern represented in Acme

**Structure and Representation.** The behavior of MVC pattern relies on the functions performed by the following components of this pattern: Model, View, and Controller. The Model provides the functional core of the application and notifies views about data changes. Views retrieve information from the model and display it to the user. Controllers translate events into requests to perform operations on the view and model elements.

**Impacts on Non-functional requirements.** This pattern contributes positively to the Usability NFR since the Views are synchronized. However, it compromises the modifiability, because some coupling of views and controllers are introduced to the model. Also, it affects negatively the of data access in view. Its impact on Security and Performance is neutral.

How to apply.

1 – Analyze and relate the components of the original architecture with the elements of the architectural pattern observing the similarities of their roles and responsibilities.

2 – Analyze and compare the existing connectors in architecture with the pattern connectors and associate them when they are similar.

3 - When a component of the architecture has responsibilities belonging to more than one component of the pattern, the component should be divided and its parts associated with the components of the pattern.

4 - Add connectors between the components when necessary.

Example of use.

Now we apply the MVC pattern to the Media example. We start classifying the components of the architecture according to the elements of the MVC pattern (Model, View, and Controller). *Statistics Producer*, *Media*, *Database*, and *Profile Manager* are concerned in the main functionalities of the system and so on are classified as Model components.

*Media Shop* Interface component suits to two MVC roles; it is both View and Controller. Therefore, we need to split it into two components. Finally, *Details Identifier* is a Controller.

The next step is to analyze the connectors. First, we identify which connectors of the architecture have the same role of the connectors of the pattern. Between the *Media* (Model) and *Details Identifier* (Controller), the security connector plays the role of an update connector. A new connector is needed to set up the update connection. Applying the same tactic to the other connectors, we get the set up shown on Figure 3-18.

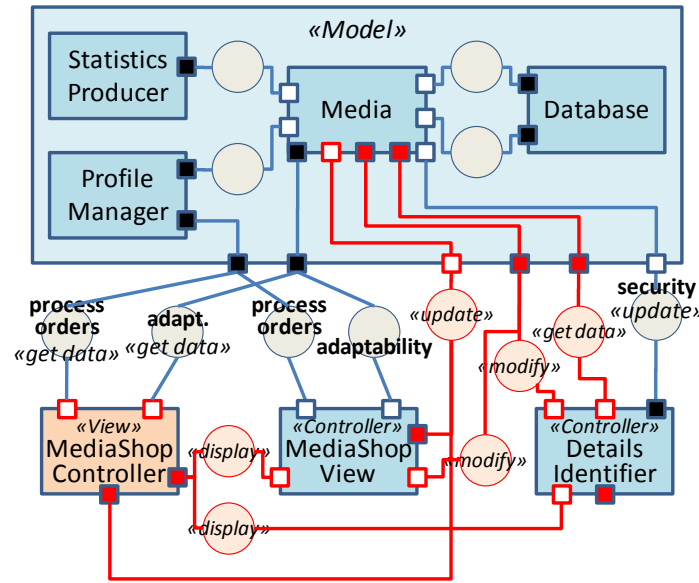


Figure 3-18 Media after applying MVC pattern

### 3.6.3 Architectural refinement pattern

The aim of this activity is to refine softgoals within pairs of interacting components or within components. Thus, we can refine connectors or components. For each softgoal to be reached, the following guidelines can be used:

(i) Identification of connectors and components related to softgoals to be achieved: an analysis of softgoal spread in the architecture shows the architectural elements that are being affected by these softgoals helping to identify which components are related to the softgoals concerned. Moreover, other elements of the architecture can be affected indirectly by softgoals. This occurs, for example, when an architectural element that is not explicitly related to a softgoal, but is related to another element that is explicitly linked to one or more softgoals.

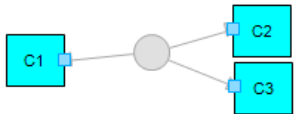
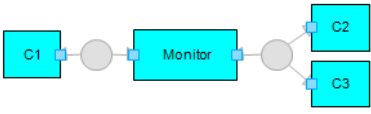
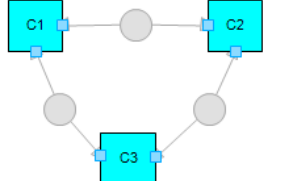
(ii) Instantiate the refinement pattern to each situation found previously: for each architectural element that is related to a softgoal, associate the context of this element with a refinement pattern that best meets these softgoals.

(iii) Refine each of them to meet this instantiation:

Maintain Availability refinement pattern

This architectural refinement pattern intends to satisfy the availability requirement. Availability is related to monitoring, compensating and recovering the connection between the components of the system. The pattern has three components (C1, C2 and C3). The component C1 depends on the availability of both C2 and C3. The pattern identifies this context and introduces a *Monitor* component to reduce the effect of the unavailability of either C2 or C3. However, the existence of new components can contribute to the increased redundancy of the connectors. Figure 3-11 shows two possible contexts and a possible solution to achieve availability among components C1, C2 and C3. Figure 3-11-a presents the components C1, C2 and C3 interconnected with just one connector, in this case when the connection fails these components are unavailable.

Table 3-12 Maintain Availability refinement pattern

| Maintain Availability refinement pattern                                                       |                                                                                                 |
|------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| Problem                                                                                        | Achieve availability                                                                            |
| Context                                                                                        | Solution                                                                                        |
|  <p>(a)</p> |  <p>(c)</p> |
|  <p>(b)</p> |                                                                                                 |

In the Media example, one of the non-functional requirements that the architecture must meet is availability. The system must not fail in situations such as the loss of the Database connection. Thus, the initial architecture generated for Media to satisfy the availability softgoal, will be adapted according to the Maintain Availability refinement pattern. A Monitor component is placed between the component *Media* and the *Database*, *Statistics\_Producer*, *Profile\_Manager* and *Item\_Manager* components (Figure 3-19). The Monitor component can be lately implemented as a proxy or as a buffer so that the occasional unavailability of a component will not be sensed by the Media component.

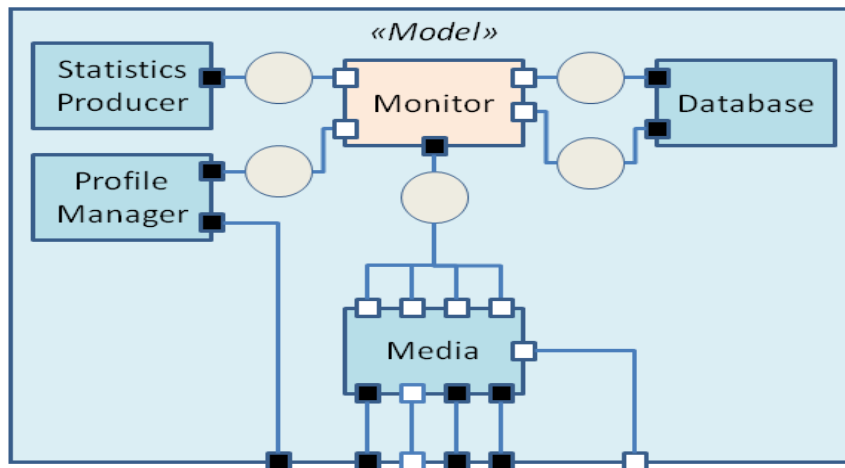


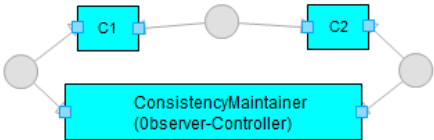
Figure 3-19 Architectural Solution for Medi@ after application of maintain availability refinement pattern

### Maintain Accurate Data Refinement pattern

Data Accuracy is related to maintaining a consistent state between related objects. In some cases, the state of the system is spread over many components. Every component may own a specific representation of the data. To maintain the consistence, any modification of the data must be available for the related components. In addition, conflicts among concurrent modifications must also be resolved by the system.

The Maintain Accurate Data refinement pattern (Table 3-13) introduces a new component to maintain the data consistency. The dataflow between the original components is redirected to the Consistency Maintainer (CM). The CM provides an interface to retrieve and update their state. When a component needs the current state, it gets it from the CM. When a component modifies its data, it updates CM's state.

Table 3-13 *Maintain Accurate Data* refinement pattern

| Maintain AccurateData refinement pattern                                                       |                                                                                                 |
|------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| Problem                                                                                        | Maintain Accurate Data                                                                          |
| Context                                                                                        | Solution                                                                                        |
|  <p>(a)</p> |  <p>(b)</p> |

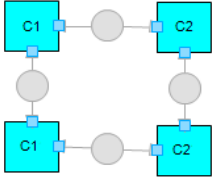
### Maintain Fault Tolerant Communication Refinement Pattern

In the communication between two components using a connector, any element may be faulty. A fault-tolerant communication scheme is related to ensuring that the

communication is not affected in the case of a fault in the components or in the connector. The communication scheme must detect and correct, in a transparent way, the elements that fail.

The Maintain Fault Tolerant Communication Pattern is implemented by a redundancy-based scheme. The redundancy is achieved by duplicating the participant elements (components and connector). In addition, copy connectors are introduced between each original components and their copy. A component and its copy have the same behavior but only one of them is active in a given time. The (spare) in active component stays ready to take over in case of fail. When the spare component takes over, it becomes the main component and the other one is reset. After the correction, the reseted component becomes the new spare. The copy connector is used to synchronize the components and to detect fails.

**Table 3-14 Maintain FaultTolerantCommunication refinement pattern**

| Maintain FaultTolerantCommunication                                                            |                                                                                                 |
|------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| Problem                                                                                        | Fault Tolerant Communication                                                                    |
| Context                                                                                        | Solution                                                                                        |
|  <p>(a)</p> |  <p>(b)</p> |

### Maintain Interoperability refinement pattern

Interoperability is related to the necessity of exchanging data and services between systems. Interoperability is a critical issue for heterogeneous systems. These systems are composed of components implemented in different platforms. Some of them may be legacy systems that must be incorporated into new systems. The communication between components requires an extra effort to adapt the components. In addition, the components are vulnerable to changes in the interfaces of the components.

Even homogenous systems can benefit from interoperability. Interoperability makes the software architecture design to be more robust and flexible. Future requirements changes may lead to design extensions but the interoperability solution minimizes the effects.

**Table 3-15 Maintain Interoperability refinement pattern**

| Maintain Interoperability |                           |
|---------------------------|---------------------------|
| Problem                   | Maintain Interoperability |
| Context                   | Solution                  |
| <p>(a)</p>                | <p>(b)</p>                |

The maintain Interoperability Refinement Pattern (Table 3-15) replaces a connector between two components with a pair of proxies components that implements a well-established communication protocol. The proxies prevents that changes in the interface of a component do not affect the component of the other side, only the proxy needs to be adapted.

### 3.7 Related Work

As said previously, goal oriented approaches are promising for dealing with reasoning and understanding of requirements [Lamsweerde, 2001]. They are also appropriate for reasoning and developing architecture. They offer a unified framework in which both functional and non-functional concerns can be integrated and refinement/abstractions links are defined precisely and provide the basis for various forms of qualitative, quantitative or formal reasoning [Lamsweerde, 2001]. Thus, we can cite a few approaches that produce architecture considering goal models as source models: KAOS [Lamsweerde, 2003], AOV-graph [Silva, 2007], i\*[Bastos, 2005].

The SIRA approach [Bastos, 2005] focuses on a systematic way to assist the transition from requirements to architecture. It describes a software system from the perspective of an organization, as stated by the Tropos methodology [Castro, 2002]. Both requirements and architecture models are described in i\*. An organizational architectural style is chosen based on a catalogue of non-functional requirements presented in [Kolp et al., 2006]. Thus, an architectural model is created considering similarities between the requirements elements and the elements of the chosen organizational style.

Axel Lamsweerde [Lamsweerde, 2003] defines a method to produce architectural models from KAOS requirements models. In this approach, requirements specifications are gradually refined to meet specific architectural constraints of the domain and an abstract architectural draft is generated from functional specifications. The resulting

architecture is recursively refined to meet the various non-functional goals analyzed during the requirements activities. It relies on KAOS modeling language, which consist of a graphical tree and a formal language.

In [Silva, 2007] a set of mapping rules is proposed between the Aspectual Oriented V-graph (AOV-graph) and the AspectualACME, an architecture description language based in Acme. Each element (goal/softgoal/task) present in an AOV-graph is mapped to an element of AspectualACME, depending on its position in the graph hierarchy. The information about the source of each element in the AOV-graph is registered in the properties of a component or a port in AspectualACME. These properties make it possible to keep both traceability and propagation of change from AspectualACME to AOV-graph models and vice versa.

In the context of this thesis, we are working specifically with  $i^*$  models to produce architectural models in Acme. To the best of our knowledge, there are no studies in  $i^*$  that generates architectural descriptions in Acme. Concerning works involving goal-oriented requirements techniques, previously mentioned, we have the SIRA approach [Bastos, 2005] which both input (requirement) and output (architectural) models are described using the  $i^*$  modeling language. Other approaches have KAOS [Lamsweerde, 2003] and AOV-Graph [Silva, 2005] as input models. In the KAOS approach, the generated architecture uses KAOS specifications instead of an architectural description language. In the approach proposed by [Silva, 2007], AOV-graph models are used to specify requirements while AspectualACME models are used to describe architectural design. Although all the requirements specification techniques used by these approaches are goal-oriented, there is no preprocessing of the requirements models, since it is well-known that goal-oriented requirements specifications present complex representations.

In our work we use  $i^*$  models as requirements models. The relation between  $i^*$  models and architectural models promote the mapping between these models [Grau, 2007], while this does not occur in KAOS and AOV-graph models because these models do not have a structure that collect their rationale. Besides, we use Acme, a generic architectural description language (ADL), which is also an interchange language. Using Acme as a target language, in the transition from  $i^*$  to architecture, brings to the  $i^*$  community [Grau, 2007] an approach to generate an architecture described in an architectural language, thereby enabling a better correlation between these activities and without having to adapt  $i^*$  language for an architectural view. For the software architecture community, the use of  $i^*$

in a transition process of requirements allows that the requirements information is represented in architectural models in Acme

The  $i^*$  modeling language is rich and expressive in describing the system requirements [Yu, 1995]. The approaches that use  $i^*$  modeling language as the starting point of software specification, such as RISD [Grau, 2005a], TROPOS [Castro, 2002], and PRIM [Grau et al, 2005], do not support a systematic transition from requirements specifications to architecture description. Often architects perform an ad hoc description of the architectural design, but do not benefit from all the expressiveness and richness offered by the requirements models. Filling this gap between requirements and architecture will allow the  $i^*$  models to drive subsequent software development phases, relating requirements models to architectural models and making generated software systems closer to the stakeholders needs.

### **3.8 Conclusion**

This chapter presented STREAM (Strategy for Transition between REquirements models and Architectural Models), a novel approach to generate an architectural model described in Acme, from  $i^*$  requirements models. The first activity was to prepare the requirements model to balance the responsibilities of a system actor, delegating them to other new system actors. A set of horizontal rules [Lucena et al, 2009a] were used to generate modular  $i^*$  models these which are closer to architectural design. From the modular  $i^*$  model the second activity of the STREAM approach derive a set of possible architectural solutions described in Acme [Lucena, 2009c]. These mappings were inspired by the some previous work [Castro et al, 2003]. The third activity of our approach is related to the selection of an architectural solution for a given set of NFR.

Finally, the fourth activity, refine detailed architecture, inspired by architectural patterns proposed by [Buschmann, 1996] and refinement architectural pattern proposed by [Lamsweerde, 2003] produced a detailed solution considering NFRs as critical factor for the use of these patterns.

From these architectural solutions described in Acme it is possible choose which architectural solution will be used since that information about non-functional requirements come from  $i^*$  actors of SR model. Thus, these non-functional requirements influence the decision about which architectural solution will be selected. This process of choice constitutes the activity Select an Architectural Solution.

Finally, the activity Derivate Architecture was presented using as input the chosen architectural solution from the previous activity. In this activity the architectural patterns proposed by [Buschmann, 1996] and refinement pattern proposed by [Lamsweerde, 2003] were used. According to non-functional requirements present in the architectural solution these patterns are applied to generated a more refined architectural model. At end of this process we have an architectural model different of initial, once that requirements model was modified using transformation rules, afterwards this new model is mapped to decisions of design and these are mapped to architectural solutions. We assessed our approach using an e-commerce system (Media).

Although it may look a transformation of one-to-one between the elements of  $i^*$  to elements of Acme, the transformation from requirements to architecture involves other steps before and after the change of notation. In the transformation rules of  $i^*$  often occurs division actors. While in the activity of refinement of the architecture can occur by the grouping of components. Thus, analyzing the initial requirements model and the final architectural model, we can see that there is not a relationship of one-to-one between the elements. We can say then that our mapping is many-to-many. This is in accordance with [Lago and Avgeriou, 2006] , which argue that the relationship between requirements and architectural solutions are many-to-many relationship.

The metrics used in the Evaluate  $i^*$  models activity can help the requirements engineer to choose which elements within an actor should be moved to another actor to improve the modularity of  $i^*$  models, however this is out of scope of our work.

# 4 Running Example

---

This chapter presents an application of our approach for the use a real web-based recommendation system showing the main steps of our process. This running example is based on the documentation of the BTW-UFPE Project: if you go, my advice to you [Pimentel et al, 2009], which was world finalist of the SCORE competition held at ICSE 2009 [SCORE, 2009]. The STREAM approach is used to generate an architectural model based on the BTW  $i^*$  models. This solution is compared against the original one proposed by the BTW Project.

---

## **4.1 Introduction**

This chapter is structured as following: initially we describe the chosen example; then we recall the activities of STREAM, and then we show each step of the approach to reach the architectural solution. At the end of this chapter, we present a comparison between the SCORE UFPE team architectural solution and the architectural solutions produced by the STREAM approach. For this comparison, we applied a set of modularity metrics and then analyzed the results.

## **4.2 BTW Project**

BTW consists of a route-planning system that helps users through the recommendation of advices about the route. Once defined the route, the user of the system can receive information about these routes, posted by other users, to help in his/her travel planning. The advices might be filtered to provide the user with relevant information about the place that he/she intends to visit. Therefore, this system moves beyond the official GIS (Geographic Information System) information that might be provided by a government or private agency, and allow the traveling public to provide advices. The BTW-UFPE team generated artifacts that include i\* requirement models.

This project was chosen by two reasons: it is not a toy problem, but a real case study that resulted in a software system; and the models are not very big but have enough complexity to present our approach properly. We applied our approach to the generated i\* models in the late requirement phase. Specifically, the SR model was used to derive the architectural models.

The requirements engineering phase included activities of requirements elicitation, requirements modeling and documentation, requirements analysis and requirements validation. In this Chapter we describe the i\* SD and SR models generated in the early and late requirements phase. The SR model is the artifact used by our approach. For more details about this project consult in [Pimentel et al, 2009].

### **4.2.1 Early Requirements BTW SD Model**

In Tropos, the early requirements analysis and specification is performed using i\* models. Therefore, the SD and SR models are used in the Early Requirements phase. As explained in the Chapter 2, the i\* models are used to represent persons and organizations and their relationships.

shows a Strategic Dependency (SD) model that represents the actors related with trip planning and their social context. The main actors of this model are:

- (i) *Travelers* – actors that plan and realize a trip. The *Traveler* is not capable of planning a trip without interacting with other actors. This interaction is modeled in terms of dependencies between the *Traveler* and other actors. The activities that the *Traveler* does, while planning and executing a trip, requires information and services provided by several different actors.
- (ii) *Community Support* - A *Traveler* needs to interact with the community to obtain information about places or conditions that will help in the planning process.
- (iii) *Transportation* - If some part of a trip requires the motion from one place to another and cannot be done by foot, the *Traveler* may consider taking different transportation vehicles, like subway or bus.
- (iv) *Internet* - Nowadays maps services and information about places are available in Internet and the *Traveler* may use this information while planning a trip.
- (v) *Hosting Place* - If a trip will require a break, the *Traveler* can select a hosting place and schedule the stops in advance.

In addition, we identified that a *Traveler* can be of different types. A *Traveler* can be a *Person with Cognitive Challenges (PWCC)*, which needs more interaction and a more detailed planning. Other type of *Traveler* is the *Hitchhiker*, which plans and executes a travel without considering the use of a private vehicle on his own. The *Cyclist* is a *Traveler* that requires a specific infrastructure.

In short, we identified that the *Traveler* needs to interact with many actors to plan a trip. A traveler seeks to reduce the number of his dependencies and increase the reliability of his information.

The next section presents the SR model that details each actor showing how their respective dependencies are treated internally. This model assists the detection of new software systems inside organizational configuration.



### 4.2.2 Early Requirements BTW SR Model

The Strategic Rationale (SR) model details how each actor supports the goals, softgoals or tasks that depend on it. Each goal is decomposed with a means-end link, the tasks are decomposed with decomposition links and the softgoals are decomposed with contribution links. shows a SR model that represents the actors rationale related to trip planning and their social context. Each gray circle represents an action boundary of an actor. Arrows that go out and go in to the actor boundary represents the relation among actors' activities. Goals are the needs of the actors. For example, *Travelers* wish to achieve the *Trip to interesting places realized* goal. Softgoals are subjective goals that have a degree of satisfaction, instead of just being satisfiable or not. Thus, *Travelers* wish to *Get fun*, *Get all moments of travel enjoyed*, *Life safed* and *Safe money*. *Cyclist* and *Hitchbiker* have the same goals of *Travelers*. *PWCC* has a specific goal to be satisfied: *Spontaneous or recreation trip intended*. *Transportation* expects the *Transfers people* goal and the *accessibility* softgoal to be achieved. The *Community support* wants to obtain the *Support provided* goal. *Hosting Places* wants to obtain the *Hosting provided* goal. Finally yet importantly, the *Internet* expects the *Service Provided* goal to be satisfied.

Note that the in the Early Requirements model (Figure 4-2) the software system actor is not present. Only during the Late Requirements phase is taken the decision to introduce (or not) a software actor to assist in the fulfillment of stakeholders wishes.



### 4.2.3 Late Requirements BTW SD Model

The Late requirements phase describes the system within its operational environment. Here we specify its functional requirements and quality requirements (or non-functional requirements).

Like in the Early Requirements discipline, the Late Requirements discipline is composed of three workflows: Late requirements elicitation, which comprises the activities of system requirements elicitation with users; Late Requirements analysis and specification, which includes analysis and modeling of functional and non-functional requirements of the system, and Late Requirements models validation, which is responsible for validating the generated models. These models are the same of the Early Requirements phase, but now they are focused on the software system itself.

The Late requirements are specified with the Strategic dependency model (SD model) and the Strategic rationale model (SR model). These artifacts are described in the following subsections.

is a Strategic Dependency model of the organization, i.e., the actors that are within the project scope and their relationships, now including the software system. In this model the system is represented by the BTW actor, which is related to other actors. The included system affects the organization by redirecting some dependencies from other actors to the BTW actor. For example, before, when a participant of this context wanted to publish or read information about a trip he needed to use several internet services for each kind of information. With the inclusion of the system (the *BTW* actor), these services, which were represented in the model by Goals, could be grouped in a unique application. The BTW system is now responsible for satisfying these needs through a recommendation system.

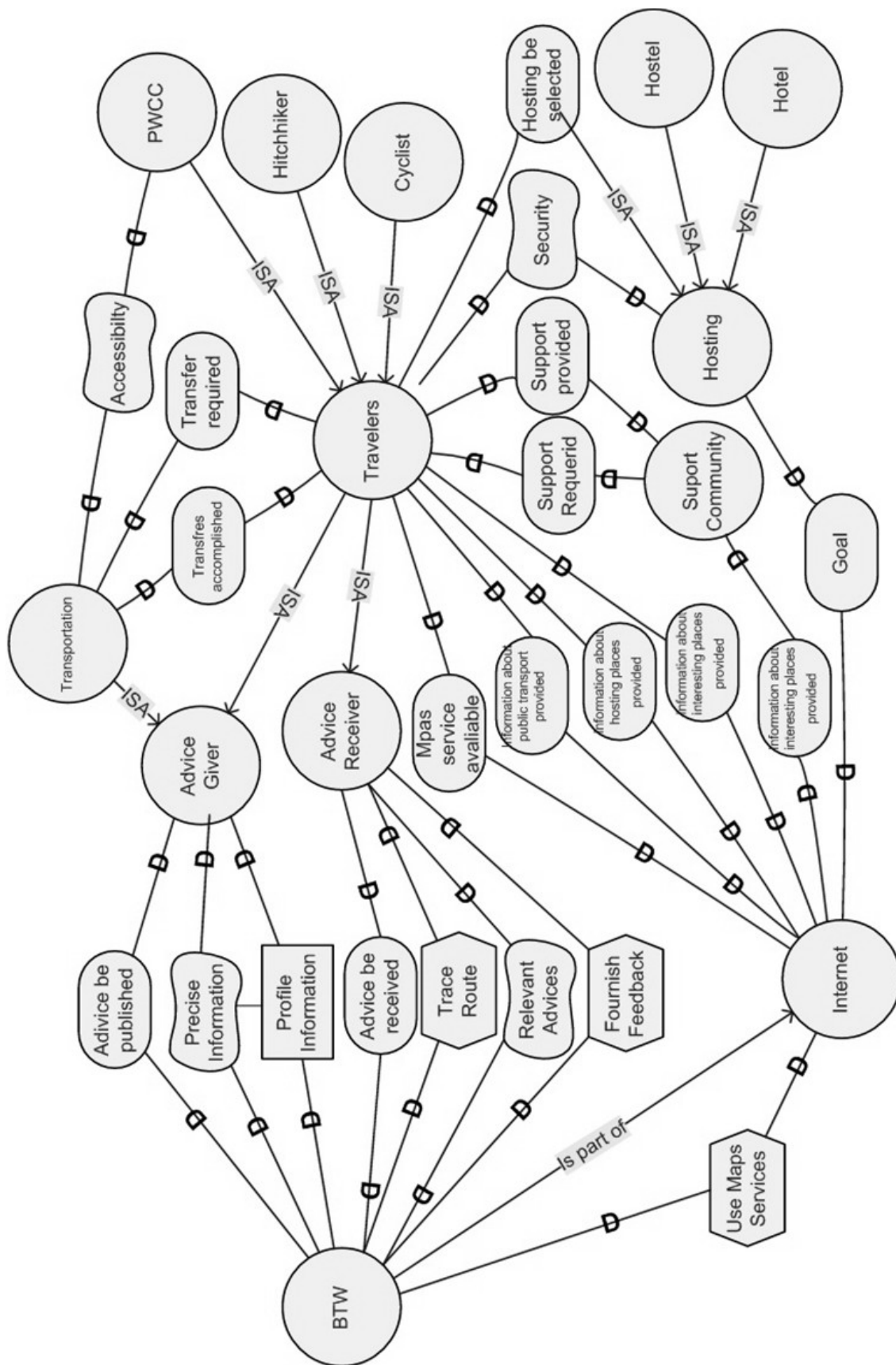


Figure 4-3 Late Requirements BTW SD Model

The SD diagram shows dependency among the system actor (*BTW*) and the other actors (*Advice giver*, *Advice receiver*, *Internet*). This makes it clear who are the actors whom use the BTW system and what are the goals that it must comply. Thus, the main goals of the system are the following:

- *Advice is published*: The system needs the *Advice Giver* to include advices that will be published to other travelers.
- *Precise information*: The system needs the *Advice Giver* to supply precise information to *BTW* system.
- *Profile information*: The *Advice Giver* supply of information depends on user profile.
- *Advice is received*: The *Advice Receiver* and *Traveler* access the system to look up for advices about their trips.
- *Trace route*: The *Advice Receiver* uses the BTW system to trace route to be followed.
- *Relevant advices*: The *Advice Receiver* wishes to obtain relevant advices about trip route.
- *Furnish feedback*: The BTW system collects feedback from the *Advice Receiver* actor.
- *Use maps services*: The BTW system uses the advices and users database, as well as geographic data and maps from the Google Maps.

The next section presents the SR model that details the system actors that are present in the Late Requirements SD model.

#### 4.2.4 Late Requirements BTW SR Model

The Strategic Rationale (SR) model represents the rationale that motivates the dependencies or that satisfies the dependencies. Thus, the SR model explains why a given dependency is necessary and how the actors pretend to work in order to satisfy that dependency.

The Figure 4-4 represents the SR model of the BTW system. This model shows how the *BTW* actor satisfies the needs of the other actors. As the main objective of the system is to provide trip advices, the high-level goal is *Trip advices be provided*. Other goals are *User access be controled* and *Map be handled*. The tasks that accomplish these goals of the Late Requirements BTW SR model (Figure 4-4) are described below.

The *Provide Advising Service* task details the main system requirements, which are decomposed in sub-tasks representing the main activities related to the recommendation mechanism: *Add Advice* and *Show Advices* tasks. Besides, this main task (*Provide Advising*) presents the *Advice be Updated* sub-goals that need to be achieved. It can be achieved in three ways:

- *No user feedback* task: users do not include a feedback.
- *Implicit feedback by monitoring* task: BTW system collects feedback by monitoring profile users and implicit information.
- *Require explicit feedback* task: user can include an explicit feedback about advices.

The *Add advices* task can be refined to obtain the *Information be published in map* goal, *Select advices theme* and *Add advice content* sub-task. The *Information be published in map* goal can be obtained by three sub-tasks: *Write information about a path*, *Write information about a point* or *Write information about an area*. These alternatives represent a way to display information in maps to users. Whereas, the *Select advices theme* meaning that advices can be selected by specific themes, for example, security, comfort, challenges, constraints to PWCC, etc.

The *Shows advices* task can be refined to obtain the *Precise advices*, *Relevant advices be chosen*, *Relevance* goals and the *Filter advices for a route* sub-task. The *Precise advices* and *Relevance* softgoals are quality attribute to advices that will be shown to users. The *Relevant advices be chosen* goal can be obtained by two sub-tasks: *Select advices by user history* or *Select advices by user profile similarity*. These are ways to system chooses first advice proposed to users, when they access system. The *Filter advices for a route* sub-task allows system choose how it presents advices to user in a route. It uses two sub-tasks that are *Access maps database* and *Calculate intersection*. The *Add advice content* sub-task shows way to add information about advices. It uses Photos, Texts and place marks. All those sub-tasks are restricted by a constraint, which is the Security softgoal.

The *Manage Users Access* task is responsible for safe access of system user. It is refined in two sub-tasks: *Manage user profile* and *Control Access to services*. The *Manage user profile* sub-task maintains profile users through the *Update profile*, *Fulfill initial profile*, *Compare profile* and *Maintain History* sub-tasks. The *Control Access to services*: provide security to system services through sub-tasks *Require password* and *Access specific services*.

The *Provide Maps Service* task is responsible for handle services to maintain maps and plot advices in maps. It is refined in *Display maps*, *Search by address* and *Display route in maps* sub-tasks. The *Display maps* sub-task shows maps to users using GIS tools. Whereas, the *Search by address* sub-task searches a place in a map by address key. While, the *Display route in maps* sub-task design in a map the selected path in a route.

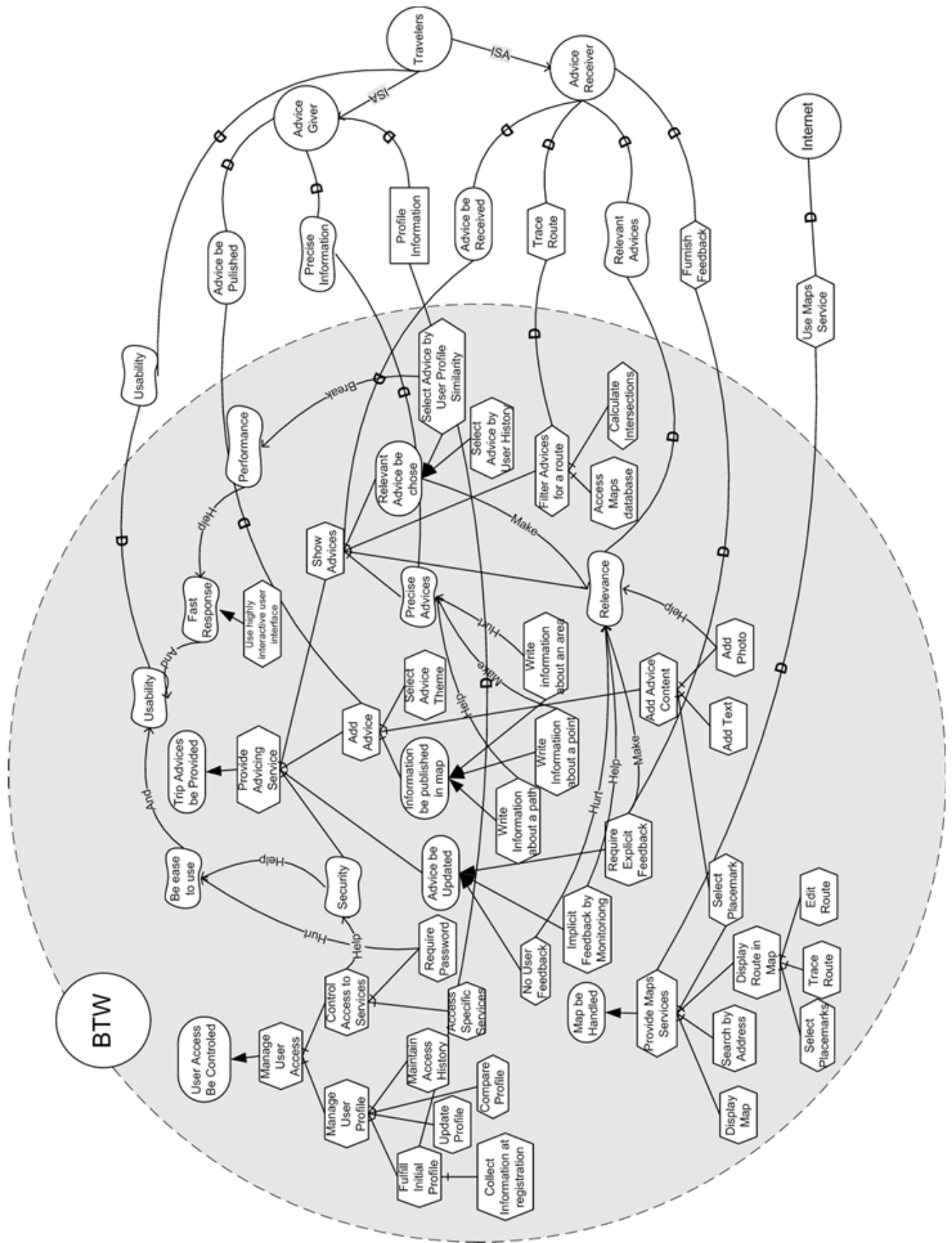


Figure 4-4 Late Requirements SR Model of BTW System

As input for our approach we use the SR model of the BTW system (Figure 4-4). As shown in Chapter 3, the STREAM approach consists of four main macro-activities: (i) Prepare Requirements Models, (ii) Generate Architectural Solutions, (iii) Select an Architectural Solution, and (iv) Refine Architecture. The following section we start with the first activity of this process: Prepare Requirements Models.

### 4.3 Prepare Requirements Models

i\* models are intrinsically complex and need to be managed. The first activity of the STREAM process addresses the complexity of SR models. Note in Figure 4-5 that this activity consume SD models and produce modular SD models. It is divided in three sub-activities:

- (i) Analyse Internal Elements: this sub-activity use of heuristics to guide the software ator's decomposition;
- (ii) Apply Horizontal Transformation Rules: this sub-activity uses horizontal rules to produce modular i\* models;
- (iii) Evaluate i\* models: this sub-activity uses modularity metrics to evaluate the initial i\* models and modular i\* models.

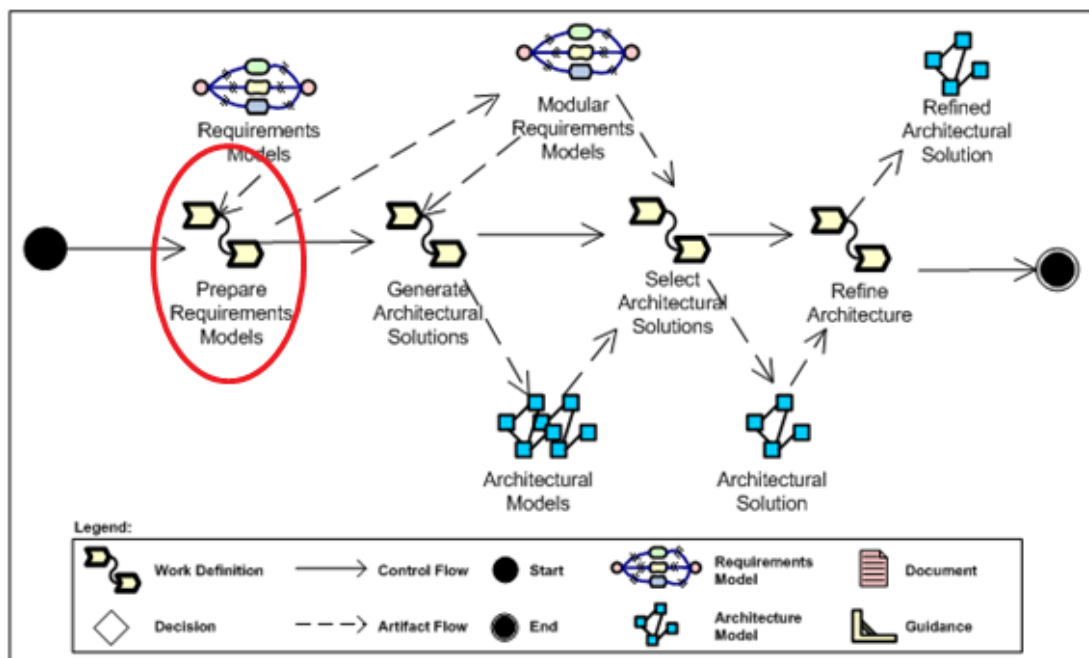


Figure 4-5 First activity of the STREAM approach

### 4.3.1 Analyze Internal Elements

In the first sub-activity of Prepare Requirements activity, the internal elements of software actors are analyzed to know if they can stay inside actor or not. Therefore, this activity receives i\* models and produces a list of elements of these models that can be moved to another actor. Figure 4-6 shows the input and output artifacts of this activity. The decomposition criterion is based on the separation and modularization of elements that are not strongly related to the application domain.

In the BTW SR model (), which captures the web recommendation system requirements, we can identify those elements that are not fully related to the application domain (recommendation). At this point, a requirements engineer must perform the analysis. Some sub-graphs internal to the BTW actor are considered independent from the recommendation application domain. Therefore, they can be moved to new software actors. Thus, sorting out the independent elements into other actors can improve reusability and maintainability of system specification at the requirements level. In fact, considering the BTW SR model, the following elements could be used as part of a system of a different application domain (Figure 4-7): *User Access be Controlled* (sub-graph A), *Map to be Handled* (sub-graph B), and *Information to be published* (sub-graph C). The main graph is the BTW actor. Thus, the sub-graphs A, B and C will be treated in the next sub-activity: Apply Horizontal.

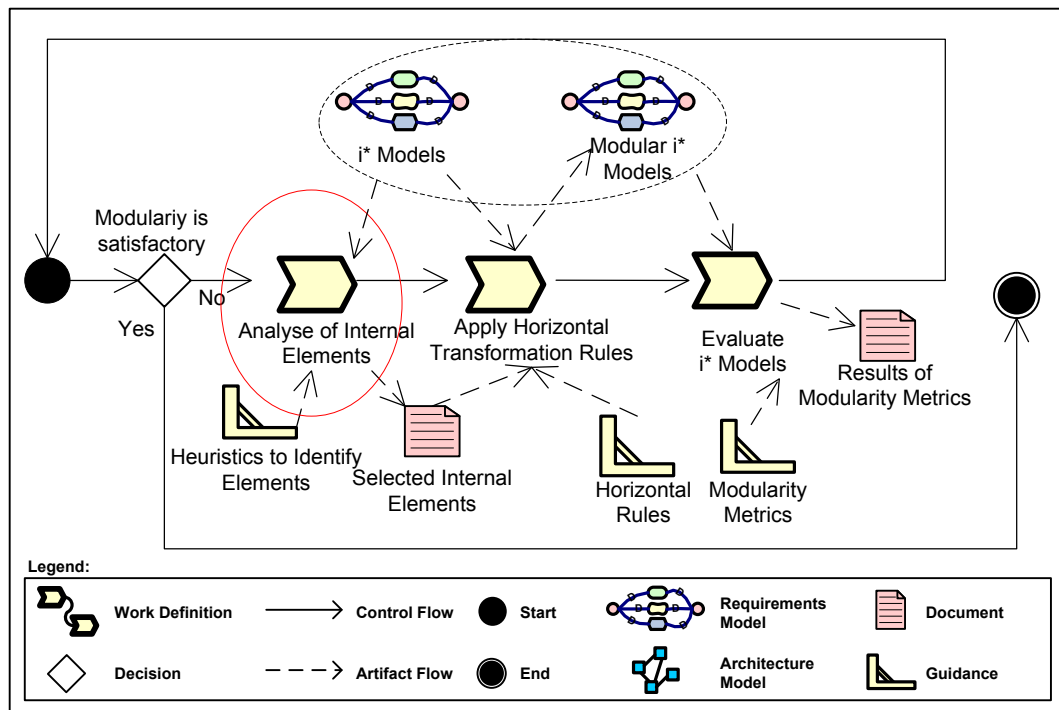


Figure 4-6 First sub-activity of the Prepare Requirements activity



### 4.3.2 Apply Horizontal Rules

In this activity, an appropriate horizontal transformation rule must be applied. For this, an i\* model, a set of selected internal elements and a set of horizontal rules are the input of this activity. While a modular i\* model is the output of this activity. Figure 4-8 the flow of artifacts of this activity.

The rule to be applied depends on the type of relationship between the elements to be moved and the elements that will remain in the original system actor. The general purpose of the horizontal rules is to delegate internal elements from the system actor to other actors. This delegation establishes a dependence relationship between the new actors and the original actor, maintaining the semantics of the original model in the resulting model.

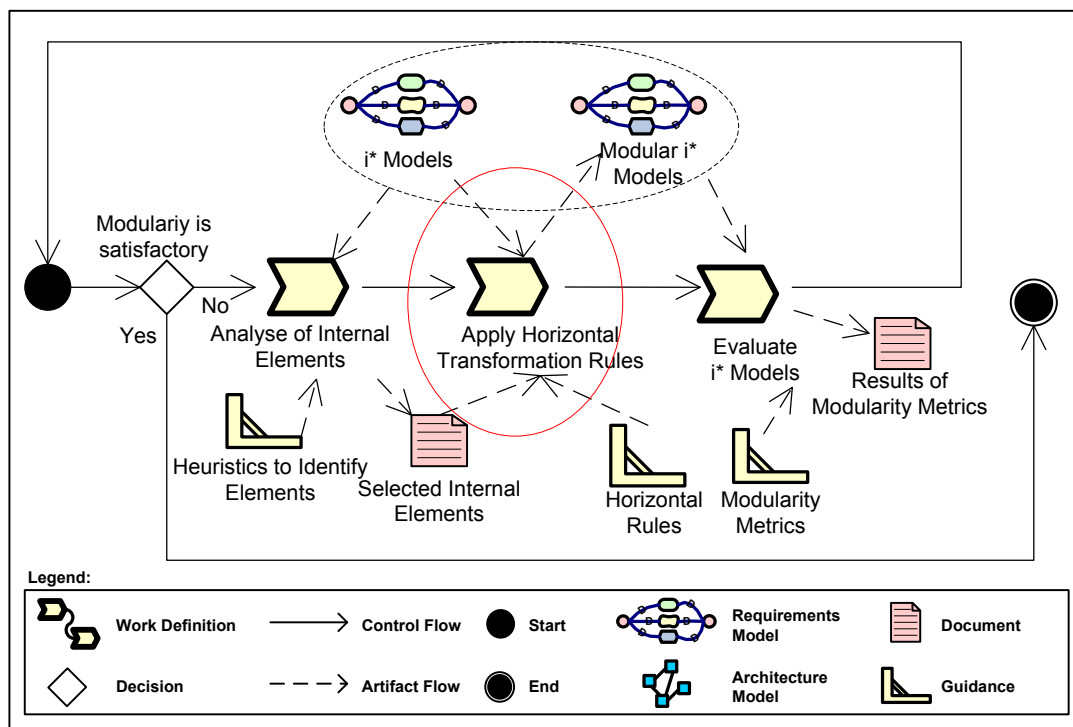


Figure 4-8 Second sub-activity of the Prepare Requirements activity

TR1 is a transformation rule that moves a sub-element present in a task-decomposition to another actor. TR1-v1 is a variation of TR1 where a sub-element is a goal in a task-decomposition. TR2 considers the situation where the sub-graph to be moved has the root element as a “means” in a means-end relationship. TR2-v1 is a variation of TR2 where a sub-element is a goal in a means-end relationship. After applying rules TR1, TR2, and their variations, the resulting model may not be in conformity with the i\* notation. In this case, we need to use a corrective rule, TR3. This rule was defined to preserve the information about contribution links and maintain the information about contribution links

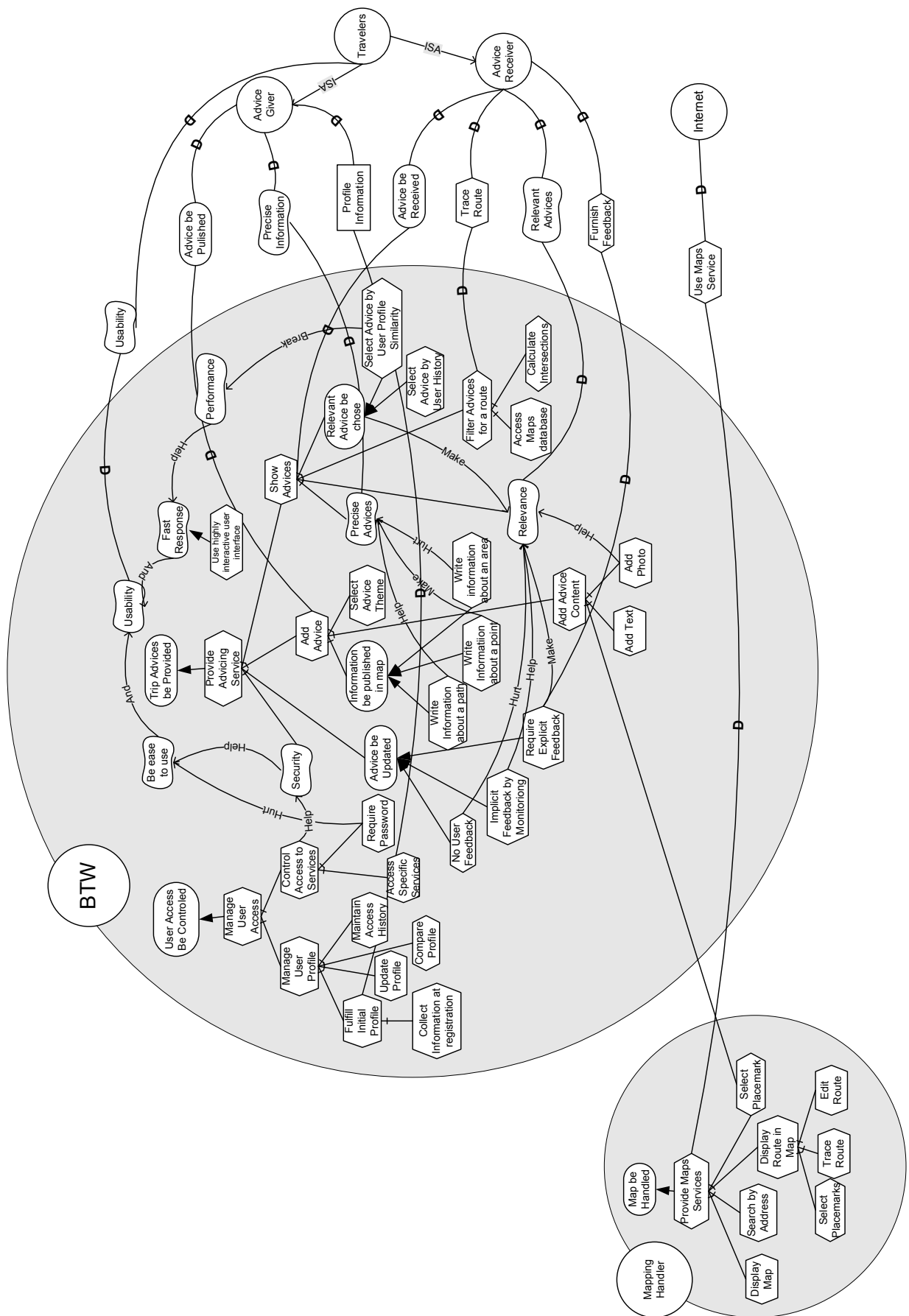
and coherence of  $i^*$  models. Besides, TR4 is applied when the sub-graph to be moved out has a sub-element shared with other sub-graphs.

Having the selected elements in the previous activity (Analyze Internal Elements), three new actors will be created to accommodate the sub-graphs A, B and C (Figure 4-7). The rules are used to move the sub-graphs and adjust the external dependencies and internal relationships related to these sub-graphs. The sub-graphs are move one at a time; however, the order usually does not affect the result. The following sequence of rules application was used for the BTW System:

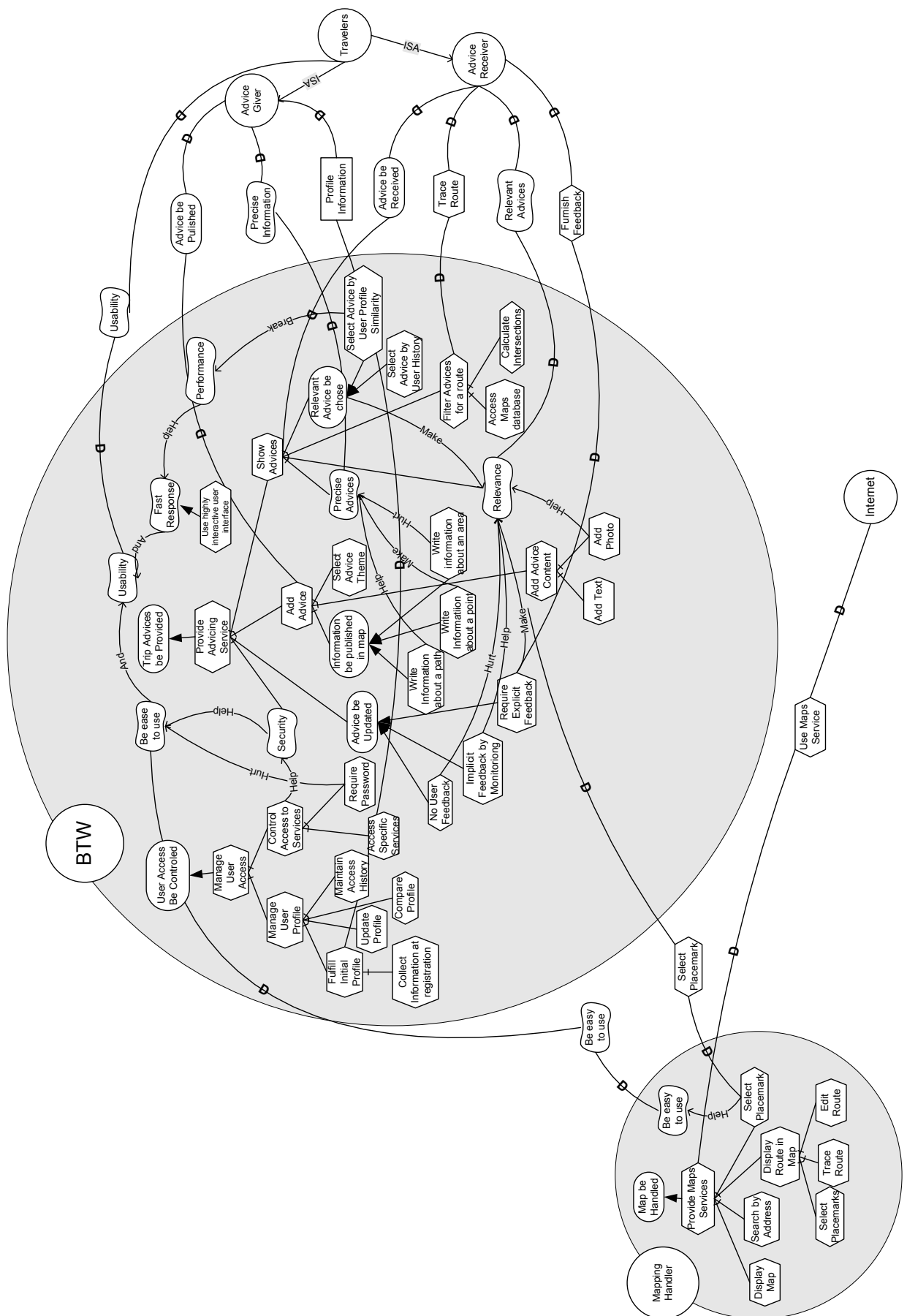
- Moving sub-graph B: the *Map be Handled* goal and its sub-graph is moved to a new actor (Figure 4-9). As there is a sharing element (*Select Placemark* task), the TR4 rule is used first to decide which actor will stay with the shared element. After applying TR4, TR1 will be used to *Select Placemark* task and TR3 will be used to treat *Be Easy to Use* softgoal. Figure 4-10 shows the new dependencies created after applying TR1 and TR3.
- Moving sub-graph C: the *Map Information Published* goal and its sub-graph is moved to a new actor (Figure 4-11). TR1-v1 is applied creating three new actors to deal with the alternative dependencies, *information be published in map*. Then, TR3 is used three times to treat the *Precise Advices* softgoal (Figure 4-12).
- Moving sub-graph A: The *User Access be Controlled* goal and its sub-graph is moved to a new actor (Figure 4-13). TR3 is used to deal with the Security, Be Easy to Use, Performance softgoals.

Figure 4-14 shows the resulting of this activity. Note that each new actor was renamed using TR5. As these new actors receive the name of their root element (goal or task), TR5 suggest to use a specific noun related to the domain of these elements. Thus, we have respectively Mapping Handler (sub-graph B), User Access Controller (sub-graph C), and Map Information Publisher (sub-graph A).

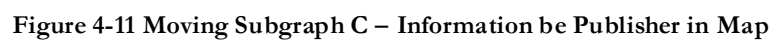
The resulting model (Figure 4-14) will be used in the next sub-activity to compare with the original  $i^*$  model (Figure 4-4).



**Figure 4-9 Moving Sub-graph B – Map be Handled Goal**



**Figure 4-10 Applying TR1 and TR3 after moving sub-graph B**





**Figure 4-13 Moving Subgraph A – User Access be Controlled Goal**



### 4.3.3 Evaluate i\* models

This activity evaluates the results of application of horizontal rules. Thus, we applied a set of modularity metrics adapted from [Santanna, 2003]. Table 4-1 presents a summary of these metrics. This activity receives both the original i\* model (Figure 4-4) and the modular i\* model (Figure 4-14), as shown in Figure 4-15.

They measure properties related with modularity are the separation of concerns, coupling and cohesion. Since the concept of concern is the base of these metrics, we consider as concerns of model the four concerns highlighted by our approach, *Access Control*, *Mapping Handle* and *Information in Maps and Recommendation* as the main concern. These concerns were selected because they are related to the system actor and the new actors created by the application of horizontal rules. Therefore, for each actor we associated a concern.

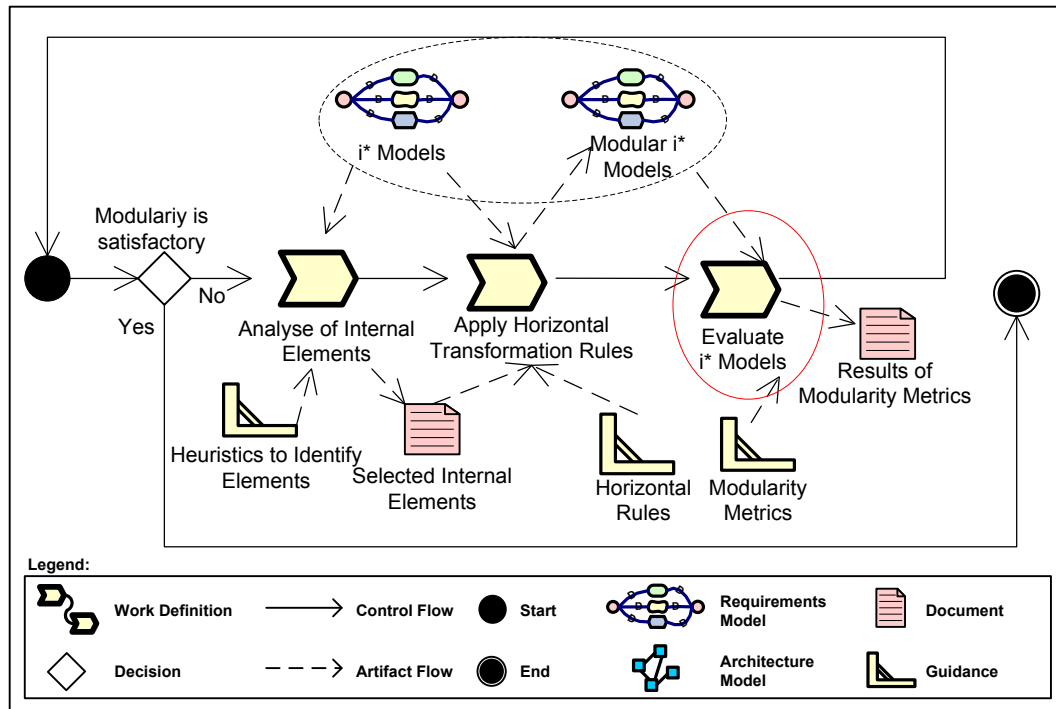


Figure 4-15 Third sub-activity of the Prepare Requirements activity

Table 4-1 present the results of modularity metrics to the original i\* model (4-4) and modular i\* model (4-14). The results show that the modular i\* model presents a reduction of diffusion of concerns in the CDIE metrics. The CDIE metrics shows a reduction from 28 to 7, it can be explained by the remotion of elements from the BTW actor. Once that the concerns that were diffused in the BTW actor are now grouped in other actors. The elements that remain in the BTW actor are no more affected by these concerns. The CDGE metrics does not varies after applying the transformation rules.

**Table 4-1 Summary of modularity metrics**

|                                                    |                                                                                                                                                                                                                   |
|----------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Concern Diffusion over Intentional Elements (CDIE) | Counts the number of Intentional Elements (e.g., Tasks, Goals) whose main purpose is to contribute to the achievement of a concern and the number of others intentional elements that require them                |
| Concern Diffusion over Graphic Element (CDGE)      | Counts the number of transition points for each concern through the refinement of the Actor (e.g. Actor's rationale). Transition points are points in the refinement where there is a change (switch) of concern; |
| Coupling between Actors (CBA)                      | Counts the number of other Actors to which a specific actor is coupled                                                                                                                                            |
| Lack of Cohesion in Intentional Elements (LCIE)    | Measure the lack of cohesion of an actor in terms of the amount of intentional elements that do not contribute to the main concern.                                                                               |

On the other hand, the coupling of the BTW actor increases from 0 to 5. It happens because in the new configuration the BTW actor is connected with 5 new actors that did not exist before.

Finally, in the LCIE metrics, the number of elements that do not contribute to the main concern of BTW actor was reduced from 26 to 15. Almost all the elements that do not contribute to the main concern were modularized in new actors. However, some new elements that do not contribute to the main concern of BTW appear, such as, the dependencies between BTW and the new actors, what prevents a bigger reduction of the LCIE.

Evaluating the results of these metrics, we can conclude that the application of the transformation rules contributes to improvement of modularity, reducing the diffusion of concerns and the lack of cohesion. The increasing of coupling is a negative side effect that was expected, since our approach contributes to increase the number of actors in the model.

The Prepare Requirements Model activity ends when we are satisfied with the result of the metrics. The resulting model is now ready for the next activity. In addition, we are considering the *Advice Receiver* and *Advices Giver* actors representing the actors that interact

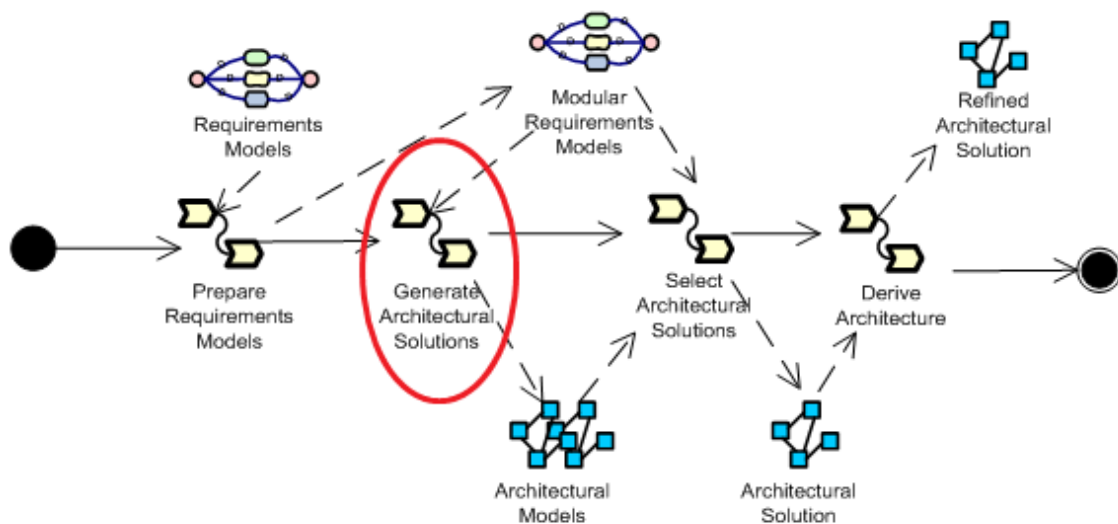
with the system and *Internet* actor as available internet services. Thus, the next section presents the application of the vertical transformation rules to produce Acme architectural models.

**Table 4-2 Results of the application of modularity metrics to BTW i\* models**

| Attribute | Metric | BTW i* model | BTW Modular i* |
|-----------|--------|--------------|----------------|
| SoC       | CDIE   | 28           | 7              |
|           | CDGE   | 2            | 2              |
| Coupling  | CBA    | 0            | 5              |
| Cohesion  | LCIE   | 26           | 15             |

#### 4.4 Generate Architectural Solutions

We start to establish the vertical transformation rules considering only actors and dependencies to map i\* elements to Acme element. Figure 4-16 shows the flow of this activity. Thus, each actor is mapped to a component and each dependency is mapped to connector. Furthermore, the roles of depender and dependee are mapped to connector roles that are comprised by the connector. In addition, required ports (where the actor is a depender) and provided ports (where actor is a dependee) are presented.



**Figure 4-16 Second activity of the STREAM approach**

Therefore, all actors of Figure 4-14 are mapped generating the following components: *BTW*, *Mapping Handler*, *User Access Controller*, *Map Information Publisher*, *Advice Receiver*, *Advice Giver* and *Internet*. Each dependency is mapped to a connector and the roles

of their connectors will be depender and dependee according the direction of dependency. For instance, when an actor has at least one dependency as a dependee, its equivalent component will have at least one provided port. Therefore, the *Mapping Handler* component will have a provided port considering the *Select Placemark* task dependency. Having all components, ports, connectors and roles mapped, the next step is analyze each type of dependency.

A goal dependency is mapped to a Boolean propriety related to a provided port of the component that offers this port. This property represents a goal that this component is responsible to fulfill by using a provided port. The type of property is Boolean in order to represent the goal satisfaction (true) or no satisfaction (false). Applying this goal dependency mapping in *BTW* system implies that *BTW* and *Map Information Publisher* components will add new Boolean properties to their provided ports.

A task dependency is mapped directly to a provided port of component that offers this port. This mapping rule affects the Mapping Handler and Advice Receiver components where the name of their respective provided ports will receive information from respective task dependencies.

In a resource dependency, an actor depends on another actor to provide information. Therefore, a resource dependency is mapped to a return type of a property of a provided port. In our *BTW* example we do not have resource dependencies related to software actors.

A softgoal dependency is similar to goal dependency but its fulfillment cannot be defined precisely. A softgoal is related to a non-functional requirement that will be treated by a task or a more specific softgoal. Hence, a softgoal dependency is mapped to a property with enumerated type present into the port that plays the dependee role of the connector. The *BTW*, *User Access Controller*, and *Map Information Publisher* components will be affected by this mapping rule. For the *BTW* system, the *Publisher* component has a provided port that interface with *Precise Advices* connector by the dependee role. This provided port will have a property softgoal defined as enumeration type. The same mapping is done with the security dependency between *BTW* and *Access User Controller* component, and so one.

Figure 4-17 presents the *BTW* architectural model in Acme from these vertical transformation rules. Observing Figure 4-14, note that the *Map Information Publisher* actors (Area, Point, Path) have the same dependencies with the *BTW* actor. Therefore, after applying the vertical rules, the *Map Information Publisher* components will have the same

interface (set of ports). In this case, the three options are grouped by the *Map Information Publisher* component. Where each alternative (Path, Area and Point) is modeled as a representation of *Map Information Publisher*. Therefore, we get three alternatives (representations) for the *Map Information Publisher*. In the final architecture only one of the representations will persist. The choice is taken in the following activity.

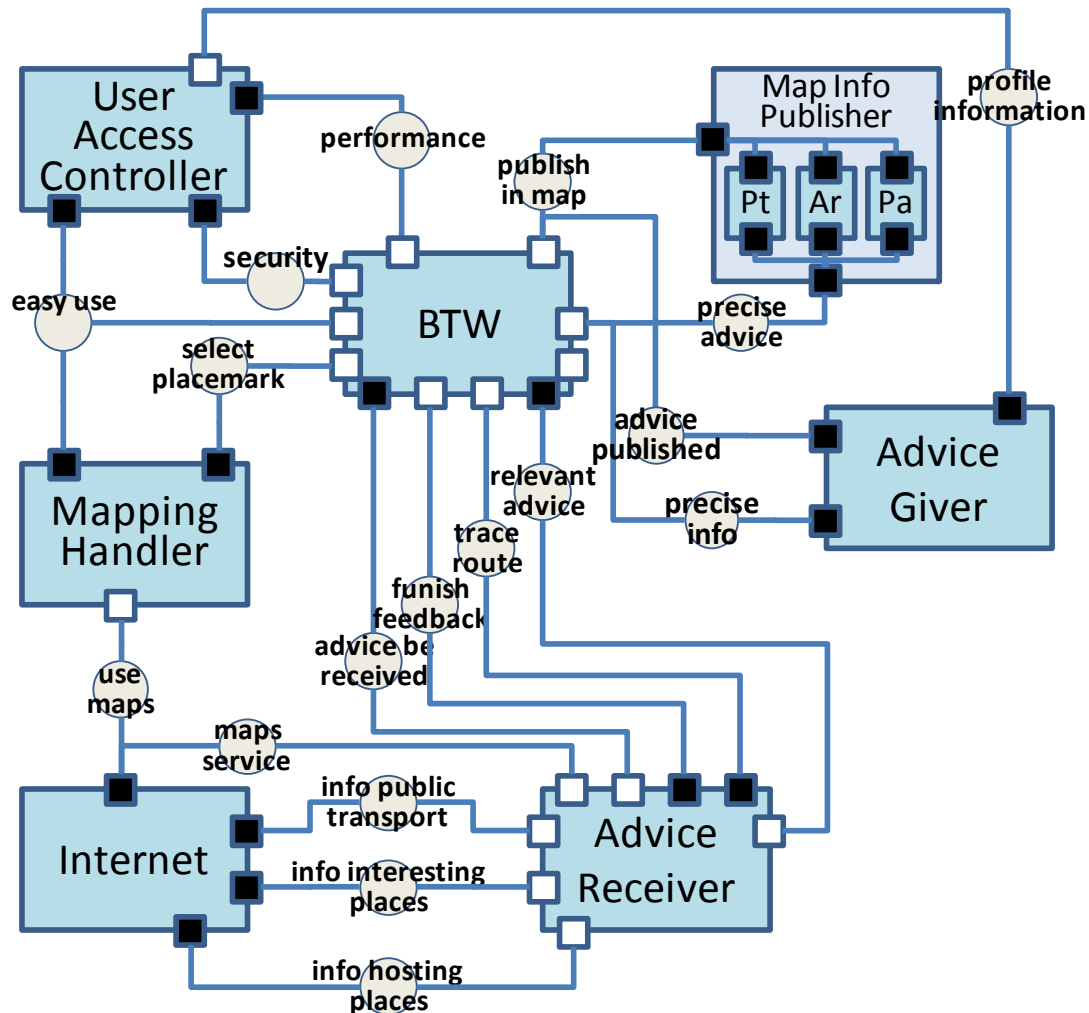


Figure 4-17 BTW Architectural model produced by the third activity of STREAM

## 4.5 Select Architectural Solutions

This activity, as shown in Figure 4-19, receives as input the modular BTW i\* model (Figure 4-14) and the architectural model produced by application of the vertical rules (Figure 4-17). The architectural model produced encloses three architectural solutions. Since *Map Information Publisher* component has three potential representations. The choice of which alternative will be used relies on the softgoals of the actors that originate the alternative components. The type of contribution that the softgoal receives determines the choice. For the BTW System, *Map Information Publish Point* actor has a *make* contribution for

the *Precise Advice* softgoal. The other two alternatives have *help* contributions for the softgoal. So, according to the strength of the contribution (see Table 2-1), *make* contributions have priority over *help* contributions. Figure 4-20 shows the selected architectural alternative.

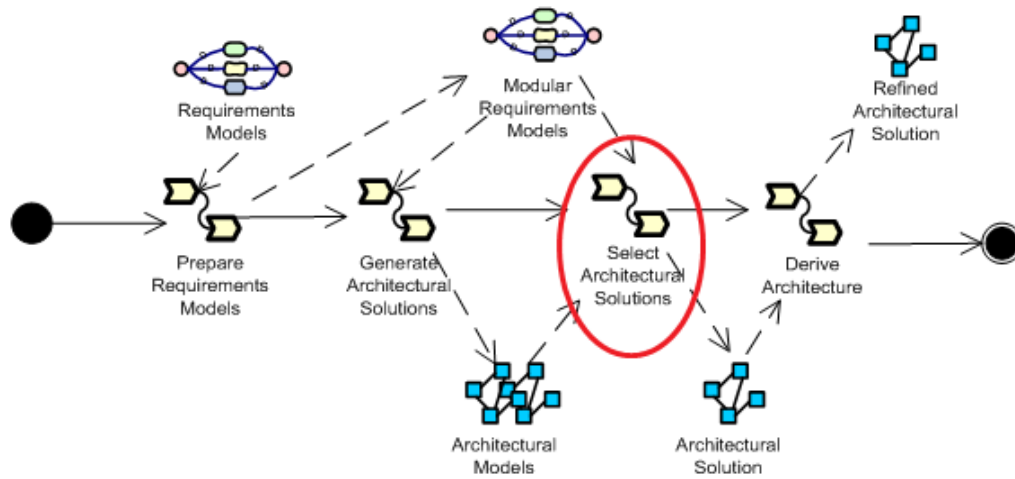
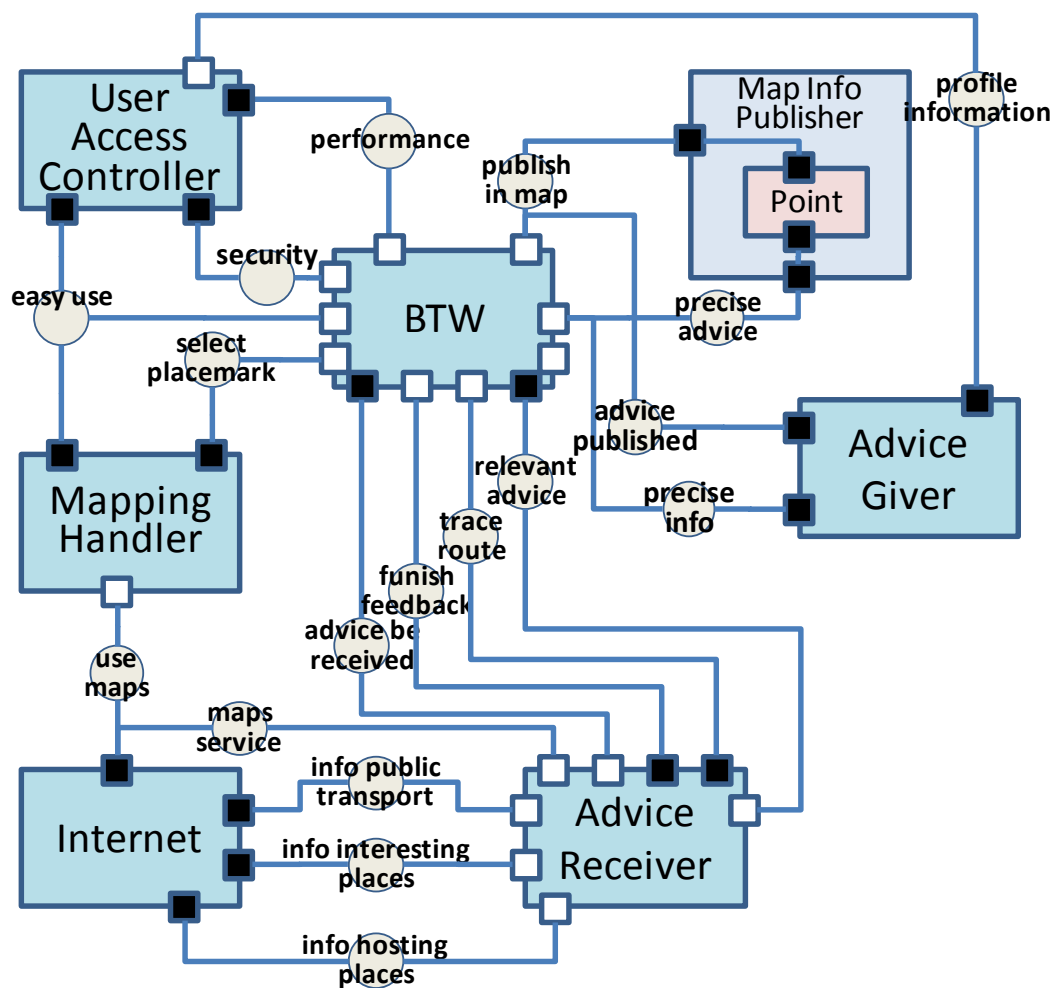


Figure 4-18 Third activity of STREAM



**Figure 4-19 Selected Architectural Solution for the BTW System**

## 4.6 Refine Architecture

The final activity of our approach (Figure 4-20) is to refine the architectural solution to satisfy the non-functional requirements. The non-functional requirements are not yet properly addressed in the architectural model. In the previous activity, we have only indicated the existence of non-functional requirements.

In our approach, this activity relies on some commonly used architecture patterns, such as MVC, Layers, Client-Server. The architectural patterns are applied globally to all architecture. The design decision of selecting as architectural pattern must rely on the non-functional requirements. In the BTW System, the main non-functional requirements are usability, accessibility, and performance. Therefore, two candidate architectural patterns are MVC and Layers. Since we have already used the MVC pattern to illustrate the example of Chapter 3, here we will apply the Layers pattern.

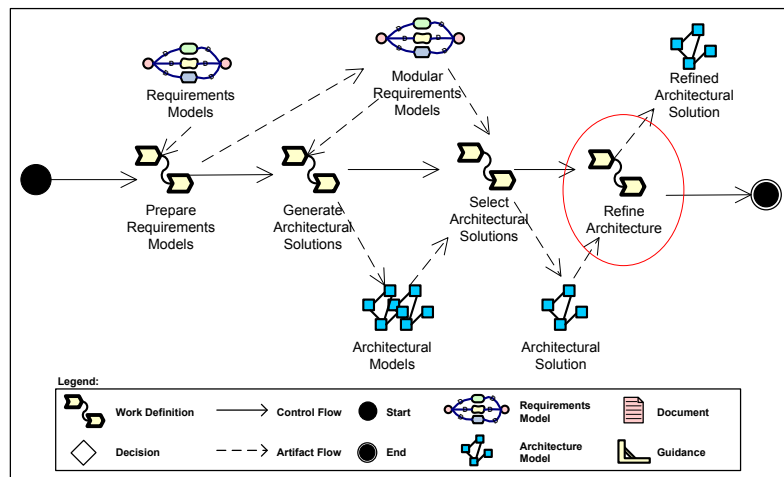


Figure 4-20 Forth Activity of STREAM

The first step is to analyze the components of the original BTW architecture and compare them with the elements of the Layers pattern observing the similarities of their roles and responsibilities. Layers that depend only on the layer below it characterize the Layers patterns. Therefore, we rearranged the components to conform to this constraint. Components at the bottom provide services to the components above them, and so on. There is no restriction for components at the same level. Therefore, components that mutually rely on each other must stay in the same level. Each level, or set of adjacent levels, can be viewed as a layer. To do so, we must group the elements of the layers and identify its role and responsibilities. We identified three layers, as shown on Figure 4-21. Their responsibilities are Interface, Business and Services.

The second step is to analyze the connector. There is no need to create or remove connectors since there is no defined role for the connectors of the Layer pattern.

The third step is to introduce components to adjust the architectural model to the pattern. The strict definition of the Layers patterns requires that a layer must rely only on the layer immediately below of it. In the BTW System architectural model, Advice Receiver component (top layer) relies on Internet component (bottom layer). The connections bypass the middle layer. Therefore, to respect the strict definition of the Layers pattern, a new component must be introduced in the middle layer to provide the Internet services to the top layer.

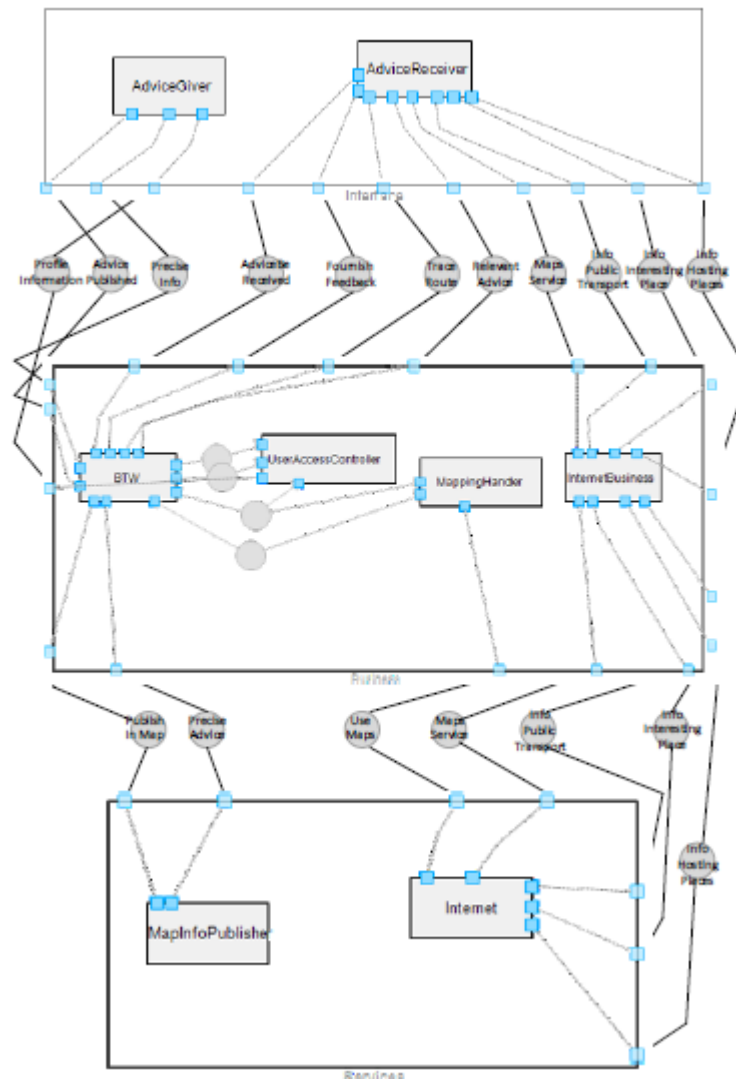


Figure 4-21 BTW architecture after applying the Layer architectural pattern

## 4.7 BTW architecture of SCORE project

Figure 4-21 presents the BTW architecture produced by UFPE SCORE team described in Acme. In this architectural model, five different components are identified: *Advice Manager*, *Profile manager*, *Advice Publisher*, *Advices recommender* and *Maps Manager*. *Profile Manager* is responsible for maintaining profiles of users of the system. The *Advice Manager* is responsible for maintaining of advices. *Advice Publisher* is responsible for the publication of advice in accordance with the profiles of users using the resources of graphic presentation of maps. *Advices Recommend* is responsible for reviewing the profiles and advice and recommendations to suggest routes or areas to be visited by travelers. *Maps manager* is responsible for maintenance of maps depending on external maps management components.

In addition, this solution architectural reflects the real implementation of the BTW system presented in the SCORE contest.

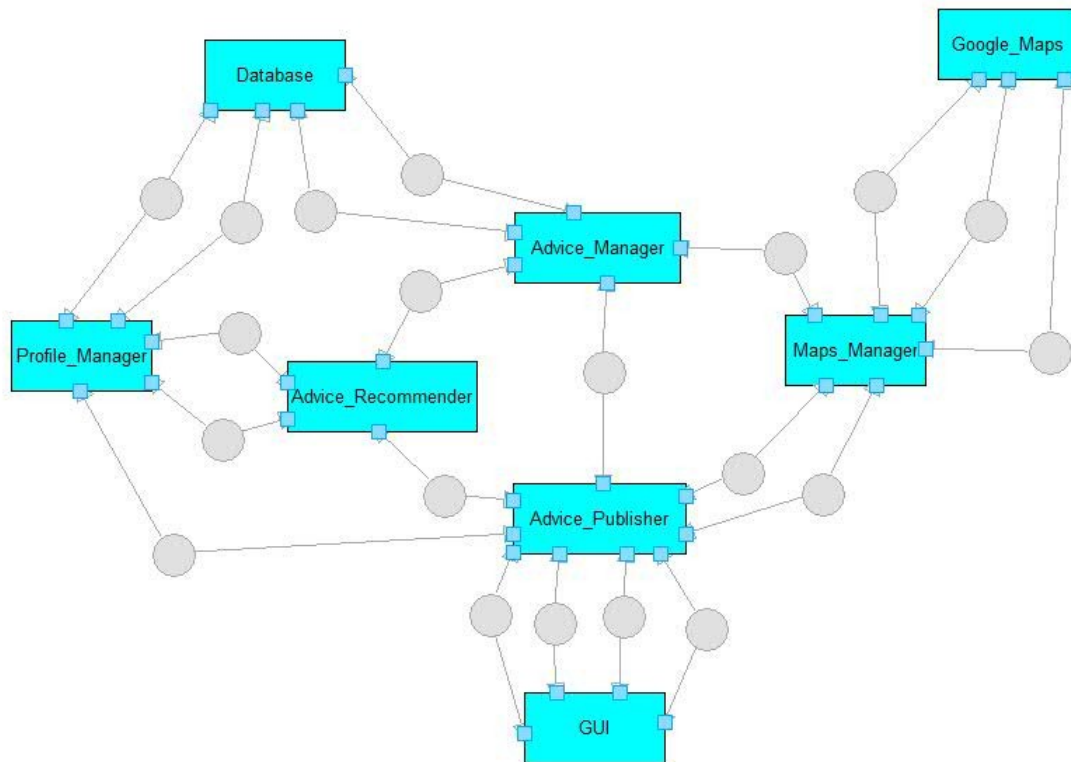


Figure 4-22 BTW architecture of SCORE project

## 4.8 Evaluating Architectural Models

In order to evaluate the architectural solutions produced by STREAM approach and by approach of BTW UFPE team, we used a set of modularity metrics proposed by [Sant'Anna et al, 2007]. We chosen this set of metrics since they are used to architectural model, such as Acme, and use the same properties used by evaluation of requirements models of our approach: separation of concerns, coupling and cohesion. Table 4-3 shows the suite of modularity architectural metrics used. The metrics that we used in this thesis were: CDAC, CDAI, CIBC, IIBC, AC, EC and LCC.

**Table 4-3 Suite of modularity architectural Metrics [Sant'Anna et al, 2007]**

| Attribute                                      | Metric                                                 | Definition                                                                                                  |
|------------------------------------------------|--------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|
| <b>Concern Diffusion</b>                       | Concern Diffusion over Architectural Components (CDAC) | It counts the number of architectural components which contributes to the realization of a certain concern. |
|                                                | Concern Diffusion over Architectural Interfaces (CDAI) | It counts the number of interfaces which contributes to the realization of a certain concern.               |
|                                                | Concern Diffusion over Architectural Operations (CDAO) | It counts the number of operations which contributes to the realization of a certain concern.               |
| <b>Coupling Between Architectural Concerns</b> | Component-level Interlacing Between Concerns (CIBC)    | It counts the number of other concerns with which the assessed concerns share at least a component.         |
|                                                | Interface-level Interlacing Between Concerns (IIBC)    | It counts the number of other concerns with which the assessed concerns share at least an interface.        |
|                                                | Operation-level Overlapping Between Concerns (OIBC)    | It counts the number of other concerns with which the assessed concerns share at least an operation.        |
| <b>Coupling Between Components</b>             | Afferent Coupling Between Components (AC)              | It counts the number of components which require service from the assessed component.                       |
|                                                | Efferent Coupling Between Components (EC)              | It counts the number of components from which the assessed component requires service.                      |
| <b>Component Cohesion</b>                      | Lack of Concern-based Cohesion (LCC)                   | It counts the number of concerns addressed by the assessed component.                                       |
| <b>Interface Complexity</b>                    | Number of Interfaces                                   | It counts the number of interfaces of each component.                                                       |
|                                                | Number of Operations                                   | It counts the number of operations in the interfaces of each component.                                     |

The following steps were used to apply this metrics framework:

1. Define a set of concerns that will drive the evaluation;
2. List all components and interfaces (ports in the Acme notation);
3. Label all components and interfaces with the concerns that will be evaluated, the components and interfaces can present more than one label;
4. Collect metrics;
5. Evaluate the results according to proprieties of separation of concerns, coupling and cohesion.

To apply this metrics framework some considerations were taken into account:

- (i) the architectural models should be described in the same notation;
- (ii) the interfaces (or ports) that are used by two or more components can be counted several times (one time by each component that access it);
- (iii) interfaces (or ports) with the same name between different components will be considered as different interfaces;
- (iv) since the number of components are different in both models, evaluating metrics also needs to consider the average value of metrics to obtain a fair comparison.

The concerns used are the same used by requirements model evaluating (sub-activity Evaluate i\* models): Recommendation (A), Access Control (B), Mapping Handle (C), Information in maps (D).

Three versions of BTW's architecture were compared using as comparison criteria the modularity properties that are: separation of concerns, coupling and cohesion. The

evaluation included the original version that competed in the SCORE (BTW-SCORE), the initial architectural solutions produced by the STREAM approach (BTW-STREAM) and a refined version of the architectural solution of the STREAM approach after applying the layers architectural pattern (BTW-STREAM-Layers). Using the modularity evaluating metrics for the attributes of coupling, cohesion and separation of concerns, we obtained the results shown in Table 4-4. It is expected that modular architectures have high cohesion, low coupling and high separation of concerns.

**Table 4-4 Results after applying architectural metrics**

| Attribute | Metrics                                                | BTW-SCORE | BTW-STREAM | BTW-STREAM layers |
|-----------|--------------------------------------------------------|-----------|------------|-------------------|
| Cohesion  | Lack of Concern based Cohesion (LCC)                   | 12        | 9          | 9                 |
| SoC       | Concern Diffusion over Architectural Components (CDAC) | 12        | 8          | 9                 |
|           | Concern Diffusion over Architectural Interfaces (CAI)  | 20        | 15         | 13                |
| Coupling  | Afferent Coupling Between Components (AC)              | 15        | 11         | 3                 |
|           | Efferent Coupling Between Components (EC)              | 15        | 9          | 3                 |
|           | Component-level Interlacing Between Concerns (CIBC)    | 6         | 2          | 12                |
|           | Interface-level Interlacing Between Concerns (IIBC)    | 4         | 2          | 2                 |

For the cohesion attribute, we used the Lack of Concern-based Cohesion metric (LCC). This metric evaluates how cohesive are the components according to how the concerns are distributed. In the case of a component with high cohesion, it is expected that it deals with a few concerns. The BTW-SCORE architecture presents 12 for LCC to eight components, while, the BTW-STREAM architecture presents 9 for LCC to seven components, while the BTW-STREAM-layers architecture presents 9 for LCC to three components. The results show that the BTW-STREAM architecture (LCC = 9 for seven components) has a better cohesion of components compared with the other architectures, since, it has a smaller number of concerns being addressed by each module. Furthermore,

the BTW-STREAM-layers architecture has a low cohesion of components, since it has the same LCC that BTW-STREAM, but has fewer components. This is because many of the concerns that were separated into components are now grouped into one layer in the BTW-STREAM-layers architecture.

The attribute of separation of concerns was evaluated using the metrics of Concern Diffusion over Architectural Components (CDAC) and Concern Diffusion over Architectural Interfaces (CDAI). The metrics offers an indication of the spread of concerns in view of the level of components (CDAC) and the level of ports (CDAI). CDAC evaluates how many components contribute to the achievement of a concern, the lower the value of the metric, the lower the spread. This indicates a high separation of concerns that are affecting specific parts of the architecture. CDAI evaluates how many interfaces contribute to the achievement of a concern, the lower the value, the lower the diffusion and consequently the greater the separation of concerns. The evaluation showed to BTW-SCORE values CDAC = 12 and CDAI = 20 for BTW-STREAM values were CDAC = 8 and CDAI = 15, and BTW-STREAM-layers CDAC = 9 and CDAI = 13. According to the results of these metrics, we can conclude that among the three architectures evaluated BTW-STREAM and BTW-STREAM-Layers showed similar results and better than BTW-SCORE. However, BTW-STREAM-Layers has fewer components. This shows that despite obtaining values close to-STREAM BTW it is not as good a solution for this attribute.

The attribute of coupling was evaluated at two levels: at the level of components, we used Afferent Coupling between Components (AC) and Efferent Coupling between Components (EC) metrics; and at the level of concerns, we used Component-level Interlacing between Concerns (CIBC) and Interface-level Interlacing between Concerns (IIBC) metrics. The component-oriented metrics (AC and EC) measure the interaction between the components, the higher the values, the greater the dependency between the modules. With fewer components, the architecture BTW-STREAM-layers (AC = 3, EC = 3) has lower value than the other architectures, which shows that the application layer has a lower coupling between the components than BTW-SCORE (AC = 15, EC = 15) and BTW-STREAM (AC = 11, EC = 9). In the case of concern-level metrics, the coupling shows how much the concerns share components (CIBC) and interfaces (IIBC). For CIBC, the more concerns share the same component, the greater the value of the metric and the greater the coupling between concerns at the component level. The same goes for IIBC, the more concerns an interface share the higher the level of coupling in Interfaces. The values obtained for the metrics CIBC and IIBC: were BTW-SCORE, CIBC = 6 and

IIBC = 4; BTW-STREAM, CIBC = 2 and IIBC = 2; BTW-STREAM-layers, CIBC = 12 and IIBC = 2. CIBC and IIBC metrics show that the BTW-STREAM architecture has less coupling between concerns, considering both components and interfaces. BTW-STREAM-layers, which had presented the best result in the metrics AC and EC, has a high value for CIBC, comparing with other architectures. This is because there are several concerns sharing just one component of the architecture.

In general, the BTW-STREAM architecture shows the best result when considering the three attributes compared. This evaluation shows that the application of the STREAM approach to produce the architecture model showed a better outcome when compared to the original architecture produced for the competition. However, the application of the layered model failed to produce a better result than the others, because it grouped different concerns in a single layer.

## **4.9 Conclusions**

Our approach generated an initial architectural model described in Acme, from i\* requirements models for the BTW system. To achieve this, it was necessary to balance the responsibilities of a system actor, delegating them to other new system actors. In order to do this a set of horizontal rules were applied in BTW SR i\* model produce a modular i\*model correspondent. In sequence, a set of vertical rules were applied in this modular i\* model producing architectural solutions described in Acme. One of these solutions were selected considering softgoals from i\* models and was refined using layers architectural pattern.

At the end of this chapter the BTW SCORE and BTW-STREAM architectures were evaluated using modularity architectural metrics.

# 5 Conclusions

---

In this chapter, some conclusions, considerations and future works are presented.

---

## 5.1 Introduction

The objective of this thesis was to integrate requirements specifications with architectural descriptions, specifically  $i^*$  requirements models and Acme architectural models. To do so, we proposed STREAM (a Strategy for Transition between Requirements Models and Architectural Models). STREAM defines a systematic strategy, based on MDD, to integrate requirements and architecture. The main characteristics of STREAM is that it smoothly and gradually transforms  $i^*$  models into Acme description by applying a set of transformation rules.

In addition to the STREAM approach, this thesis contributes with the following contributions:

- (i) a meta-model for  $i^*$  [Lucena, 2008];
- (ii) a set of transformation rules that modularize  $i^*$  models [Lucena, 2009b];
- (iii) a set of transformation rules that transforms  $i^*$  models into Acme models;
- (iv) use of metrics to evaluate the degree of modularity and complexity of  $i^*$  models and use of modularity metrics to evaluate our approach;
- (v) use of the non-functional requirements to choose one architectural solution among alternatives and use of architectural patterns to refine the initial architectural solution.

We runned some examples to demonstrate our approach. We selected two case studies from different application domains and with different degrees of complexity: an e-commerce system [Castro et. al., 2002] and a web-based recommendation system [Pimentel et al., 2009].

## 5.2 Future Works

For future works, the following activities should be undertaken:

1. Define precisely the horizontal rules proposed in this work:

Currently we are also defining precisely our horizontal transformation rules using OCL language to ensure that the resulting models will be well formed. The use of OCL will enable us to define the transformation rules and verify their soundness. However, to implement our proposal we rely on a specific model transformation language, namely ATL.

2. Implementation of the proposed horizontal and vertical rules in a i\* toolkit based in MDD.

This work includes automating this approach through the implementation of our transformation rules in the Istar Tool [Campos, 2008][Paes Jr et al, 2009], a tool based on MDD and Eclipse Platform.

3. Use more examples that allows analyzing different non-functional requirements and that generates more than one architectural solution.
4. Incorporate other architectural views in our approach;
5. Use evaluating methods to architectural models.

### **5.3 Summary of the publications**

The following publications had a relevant influence in this thesis:

1-. SILVA, C. ; ARAUJO, J. ; CASTRO, J. ; MOREIRA, A. ; **Lucena Márcia**, SARMENTO, L.. A Modeling Language for Advanced Separation of Concerns in Multi-Agent Systems. In: IDEAS, 2008, Recife. 11th Iberoamerican Workshop on Requirements Engineering and Software Environments. Recife : Fundacao Antonio dos Santos Abranches, 2008. p. 267-280.

2- ALENCAR, F. ; SILVA, C. ; **Lucena, Márcia** ; CASTRO, J. ; BATISTA, E. ; Ramos, R. A. . Improving the Understandability of i\* Models. In: 10th International Conference on Enterprise Information Systems, 2008, Barcelona. Proceeding of 10th International Conference on Enterprise Information Systems, 2008.

3- **Lucena, Márcia**; BATISTA, E. ; SILVA, C. ; ALENCAR, F. ; SILVA, M. J. ; CASTRO, J. . Towards a Unified Metamodel for i\*. In: Research Challenges in Information Science, 2008, Marrakech. IEEE International Conference on Research Challenges in Information Science, 2008.

4- SILVA, C.; **Lucena, M.** ; CASTRO, J. ; ARAUJO, J. ; MOREIRA, A. ; ALENCAR, F. . Support for Aspectual Modeling to Multiagent System Architecture. In: Early Aspects at ICSE, 2009, Vancouver. Aspect-Oriented Requirements Engineering and Architecture Design, 2009.

5- **Lucena, M.**, Silva, C., Santos, E., Alencar, F., Castro, J.: Applying Transformation Rules to Improve i\* Models. In: SEKE 09, pp 43-48 USA, 2009

6- **Lucena, M.**; SILVA, Carla T L L ; SANTOS, E. ; Alencar, F. ; CASTRO, J. F. B. . Modularizando Modelos i\*: uma Abordagem baseada em Transformação de Modelos. In: Workshop on Requirements Engineering, 2009, Valparaíso, Chile. Proceedings of the WER'09, 2009.

7- **M. Lucena**, J. Castro, C. Silva, F. Alencar, E. Santos, and J. Pimentel, A Model Transformation Approach to Derive Architectural Models from Goal-Oriented Requirements Models, LNCS. Berlin, Heidelberg: Springer, 2009, vol. 5872, pp. 370-380.

# References

- [Alencar et al, 2008] ALENCAR, F. ; SILVA, C. ; Lucena, M. ; CASTRO, J. ; BATISTA, E. ; Ramos, R. A. . Improving the Understandability of i\* Models. In: 10th International Conference on Enterprise Information Systems, 2008, Barcelona. Proceeding of 10th International Conference on Enterprise Information Systems, 2008
- [Avgeriou, 2008] Avgeriou, P., Kruchten, P., Lago, P. Third Workshop on SHARing and Reusing architectural Knowledge (SHARK 2008) 30th Int. Conf. on Software Engineering (ICSE 2008) ICSE 2008 Leipzig, Germany, 10 - 18 May 2008.
- [Bass et al, 2003] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice, Second Edition. Addison-Wesley Professional, April 2003.
- [Bastos ET al, 2005] Bastos, L., Castro, J.: From requirements to multi-agent architecture using organisational concepts. ACM SIGSOFT Software Engineering Notes 30(4): 1-7 . 2005.
- [Berry, 2003] Berry, D. M. Kazman, R., Wieringa, R.: 2nd Intl Ws on From Software Requirements to Architectures (STRAW'03) at ICSE'03, Portland, USA, 2003.
- [Boehm, 2007] B. Boehm, "Software risk management: principles and practices," Software Engineering: Barry W. Boehm's Lifetime Contributions to Software Development, Management, and Research, vol. 8, no. 1, p. 387, 2007.
- [Bresciani, 2004] P. Bresciani et al., "Tropos: An Agent-Oriented Software Development Methodology", Journal of Autonomous Agents and Multi-Agent Systems, 8(3), pp. 203-236, 2004.
- [Campos, 2008] Campos, B. Istar.-Tool: ferramenta para modelos i\*. dissertação de mestrado, Centro de Informática, 2008.

- [Castro et al, 2003] Castro, J., Silva, C., Mylopoulos, J.: Modeling Organizational Architectural Styles in UML. CAiSE 2003: 111-126, (2003)
- [Castro, 2001] Castro, J., Kramer, J.: 1st Intl. Workshop on From Software Requirements to Architectures (STRAW'01) at ICSE'01, Toronto, Canadá, 2001.
- [Castro et al, 2002] J. Castro, et al., "Towards Requirements-Driven Information Systems Engineering: The Tropos Project", Information Systems Journal, 27(6), pp. 365-389, 2002.
- [Chung, 2000] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, Non-Functional Requirements in Software Engineering, 1st ed. Springer, October 2000.
- [Clements, 2002] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, Documenting Software Architectures: Views and Beyond. Addison-Wesley Professional, October 2002
- [Clements, 2007].
- [Dardenne, 1993] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," Science of Computer Programming, vol. 20, no. 1-2, pp. 3-50, April 1993. DOI: 10.1016/0167-6423(93)90021-G
- [Deboer et al., 2008], Deboer, R., Vanvliet, H.: On the similarity between requirements and architecture, Journal of Systems and Software, 2008.
- [Estrada et al, 2006] Estrada, H., Rebollar, A. M., Pastor, O., Mylopoulos, J. An Empirical Evaluation of the i\* Framework in a Model-Based Software Generation Environment. In CAiSE'06. LNCS 4001, Springer-Verlag , 513-527. 2006.
- [Garlan et al, 1997] Garlan, D., Monroe, R. Wile, D.: ACME: An Architecture Description Interchange Language". Proc. of the CASCON 97, (1997)

- [Garlan et al., 1996] Garlan, D., Shaw, M. An Introduction to Software Architecture. Advances in Software Engineering and Knowledge Engineering, Volume I, edited by V.Ambriola and G.Tortora, World Scientific Publishing Company, New Jersey, 1993.
- [Grau and Franch, 2007] G. Grau and X. Franch, "On the adequacy of i\* models for representing and analyzing software architectures," 2007, pp. 296-305.
- [Grau et al, 2005a] GRAU, G.; FRANCH, X.; MAYOL, E.; AYALA, C.; CARES, C.; HAYA, M.; NAVARRETE, F.; BOTELLA, P.; QUER, C. "RiSD: A Methodology for Building i\* Strategic Dependency Models". In 17th International Conference on Software Engineering and Knowledge Engineering. SEKE'05. Howard International House, Taipei, Taiwan. 2005.
- [Grau et al, 2005b] GRAU, G.; FRANCH, X.; MAIDEN, N. "A Goal-Based Round-Trip Method for System Development". In 11th International Workshop on Requirements Engineering: Foundations for Software Quality (REFSQ'05). ISBN:3-922602-98-3. 2005.
- [Grau et al, 2006] GRAU, G.; FRANCH, X.; AVILA, S. "J-PRiM: A Java Tool for a Process Reengineering i\* Methodology". In 14th IEEE International Requirements Engineering Conference (RE'06). 2006.
- [Grau et al, 2007] Grau, G., Franch, X.: On the Adequacy of i\* Models for Representing and Analyzing Software Architectures. ER Workshops 2007: 296-305.
- [Grau et al, 2008] Grau, G. et al., 2008. i\* Wiki Home, In <http://istar.rwth-aachen.de/>, 02/04/09.
- [Grunbacher et al, 2004]. P. Grünbacher, A. Egyed, N. Medvidovic, "Reconciling Software Requirements and Architectures with Intermediate Models", Software and System Modeling (SoSyM), Vol. 3, no. 3, Springer, pp. 235--253, 2004.

- [Harel, 2004] Harel, D., Rumpe, B., Meaningful Modeling: What's the Semantics of 'Semantics'?, *Computer* 37:10, IEEE Press, 2004, pp 64-72
- [Horkoff, 2007] Horkoff, J., Using i\* modeling for evaluation, Master thesis, Department of Computer Science, University of Toronto, Canada, 2007
- [IEEE, 2000] IEEE Computer Society (2000) IEEE Recommended Practice for Architectural Description of Software-Intensive Systems: IEEE Std 1471.
- [Kleppe, 2003] Kleppe, A., Warmer, J., Bast, W.: MDA Explained - The Model Driven Architecture: Practice and Promise. Addison-Wesley. (2003)
- [Kolp et al., 2001] Kolp, M., Castro, J., Mylopoulos, J. A social organization perspective on software architectures. In Proc. of the 1st Int. Workshop From Software Requirements to Architectures. STRAW'01, Toronto, Canada, 2001. pp.: 5-12.
- [Kolp et al., 2006] M. Kolp, P. Giorgini and J. Mylopoulos, "Multi-Agents Architectures as Organizational Structures". In *Journal of Autonomous Agents and Multi-Agent (JAAMAS)*, 13(1):3-25, Springer, 2006.
- [Lago, 2009] Lago, P., Avgeriou, P., Kruchten, P. Fourth Workshop on SHARing and Reusing architectural Knowledge (SHARK 2009) 31st Int. Conf. on Software Engineering (ICSE 2009), ICSE 2009, Vancouver, Canada, May 16-24, 2009.
- [Lamsweerde, 2000]. Lamsweerde, A., 2000. Requirements Engineering in the Year 00: A Research Perspective. In Keynote paper, Proc. ICSE'2000 - 22nd Intl. Conference on Software Engineering. IEEE Computer Society Press
- [Loucopoulos, 1995].
- [Lamsweerde, 2001] A.van Lamsweerde, "Goal-oriented requirements engineering: A guided tour," in *RE '01: Proceedings of the*

- 5th IEEE International Symposium on Requirements Engineering. Washington, DC, USA: IEEE Computer Society, 2001.
- [Lamsweerde, 2003] A. van Lamsweerde. From system goals to software architecture. In 3rd Interl School on Formal Methods for the Design of Computer, Communication and Soft Systems, 2003.
- [Lamsweerde, 2005]
- [Lamsweerde, 2009] A. van Lamsweerde, Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley, March 2009.
- [Lucena et al, 2008] Lucena, M.; BATISTA, E. ; SILVA, C. ; ALENCAR, F. ; SILVA, M. J. ; CASTRO, J. . Towards a Unified Metamodel for i\*. In: Research Challenges in Information Science, 2008, Marrakech. IEEE International Conference on Research Challenges in Information Science, 2008.
- [Lucena et al, 2009a] Lucena, M., Silva, C., Santos, E., Alencar, F., Castro, J.: Applying Transformation Rules to Improve i\* Models. In: SEKE 09, pp 43-48 USA, 2009
- [Lucena et al, 2009b] Lucena, M.; SILVA, Carla T L L ; SANTOS, E. ; Alencar, F. ; CASTRO, J. F. B. . Modularizando Modelos i\*: uma Abordagem baseada em Transformação de Modelos. In: Workshop on Requirements Engineering, 2009, Valparaíso, Chile. Proceedings of the WER'09, 2009.
- [Lucena et al, 2009c] M. Lucena, J. Castro, C. Silva, F. Alencar, E. Santos, and J. Pimentel, A Model Transformation Approach to Derive Architectural Models from Goal-Oriented Requirements Models, ser. LNCS. Berlin, Heidelberg: Springer, 2009, vol. 5872, pp. 370-380.
- [Medvidovic and Taylor, 2000] Medvidovic, N., Taylor, R. N.: A Classification and Comparison Framework for Software Architecture Description Languages. Software Engineering, IEEE Transactions on 26(1):70-93 (2000)

- [Nuseibeh, 2001] Nuseibeh, B.: Weaving Together Requirements and Architectures. *IEEE Computer* 34(3), 115-117 (2001)
- [OMG, 2006] Object Management Group (OMG), 2006. Object Constraint Language (OCL) Specification, 2.0. Available at: <http://www.omg.org/cgi-bin/apps/doc?formaV06-05-01.pdf>. Last access: 11/2007
- [Pimentel et al, 2009] Pimentel, J., Borba, C, Xavier, L.: BTW: if you go, my advice to you Project. <https://jaqueira.cin.ufpe.br/jhcp/docs/>. July, 2009 SCORE world finalist teams
- [Rapanotti, 2002] Lucia Rapanotti, Jon G. Hall, Michael Jackson, Bashar Nuseibeh J. Hall, M. Jackson, R. Laney, B. Nuseibeh, and L. Rapanotti. Relating software requirements and architectures using problem frames. pages 137–144, 2002
- [Sant’Anna ET AL, 2003] Sant’Anna C. N. et al. On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework. *Proc. of SBES’03, Brazil, SBC*, 19–34. 2003.
- [Sant’Anna ET AL, 2007] C. Sant’Anna, E. Figueiredo, A. Garcia, and C. Lucena, "On the modularity of software architectures: A concern-driven measurement framework," in *Software Architecture*, ser. LNCS, 2007, vol. 4758, pp. 207-224. doi:10.1007/978-3-540-75132-8\_17
- [SEI, 2009] Software Engineering Institute web site, Carnegie Mellon University. <http://www.sei.cmu.edu/>
- [Shaw et al, 1996] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, April 1996.
- [Silva et al, 2007] Silva, L., Batista, T., Medeiros, A., Garcia, A., , Thais Batista, Ana Luisa Medeiros, Alessandro Garcia, Leonardo Minora.: On the Symbiosis of Aspect-Oriented Requirements and Architectural Descriptions. 10th International Workshop On Early Aspects (EA-AOSD) Co-located with AOSD 2007..

- [Silva et al, 2008] SILVA, C. ; ARAUJO, J. ; CASTRO, J. ; MOREIRA, A. ; Lucena M., SARMENTO, L.. A Modeling Language for Advanced Separation of Concerns in Multi-Agent Systems. In: IDEAS, 2008, Recife. 11th Iberoamerican WER 2008,. p. 267-280.
- [Silva et al, 2009] SILVA, C. ; Lucena, M. ; CASTRO, J. ; ARAUJO, J. ; MOREIRA, A. ; ALENCAR, F. . Support for Aspectual Modeling to Multiagent System Architecture. In: Early Aspects at ICSE, 2009, Vancouver. Aspect-Oriented Requirements Engineering and Architecture Design, 2009.
- [SCORE, 2009] The SCORE 2009 Contest. <http://score.elet.polimi.it/>
- [SPEM, 2006] SPEM - Object Management Group: Software Process Engineering Metamodel (SPEM) Specification, Version 1.1. <http://www.omg.org/docs/formal/05-01-06.pdf>
- [Taylor, 2009] Taylor, R. N., Medvidovi, N., Dashofy, I. E.: Software Architecture: Foundations, Theory, and Practice. John Wiley & Sons, 2009.
- [Travassos et al, 2002] Travassos, G., Gurov, D., Amaral, E.: Introdução à Engenharia de Software Experimental. Relatório Técnico, RTES-590/02, Rio de Janeiro, 2002.
- [Yu et al, 2004] Y. Yu, Julio, and J. Mylopoulos, "From goals to aspects: Discovering aspects from requirements goal models," in RE '04: Proceedings of the Requirements Engineering Conference, 12th IEEE International (RE'04). Washington, DC, USA: IEEE Computer Society, 2004, pp. 38-47. DOI:10.1109/RE.2004.23.
- [Yu et al, 2008] Yu, E., Castro, J. & Perini, A., 2008. Strategic Actors Modeling with i\*. In Tutorial Notes, 16th Intl. Conf. on Requirements Engineering. Spain: IEEE Computer Society, pp. 01-60. 2008.
- [Yu, 1995] Yu, E., "Modeling Strategic Relationships for Process Reengineering", Ph.D. thesis, Department of Computer Science, University of Toronto, Canada, 1995.



## A Appendix

---

This Appendix presents the textual representation of Acme models described in the Thesis.

---

## A.1 Textual representation of the Medi@ System (Chapter 3)

```
import families/Family1.acme;

System System2 : Family1 = new Family1 extended with {

  Component Produce_Statistics = {
    Port produce_statistics = {

    }

  }

  Component Adaptation = {
    Port adaptation = {

      Property provided : boolean = true;

    }

    Port availability = {

      Property provided : boolean = true;

      Property softgoal : SoftgoalType = unknown;

    }

    Port security = {

      Property provided : boolean = true;

      Property goal : SoftgoalType = unknown;

    }

  }

  Component Media = {
    Port pMedia = {

      Property required : boolean = true;

    }

    Port query_database = {
```

```

    Property required : boolean = true;
  }
  Port catalog_consulting = {
    Property required : boolean = true;

  }
}
Component Query_Database = {
  Port query_database = {
    Property provided : boolean = true;
  }
  Port item_to_be_selected = {

    Property provided : boolean = false;

    Property goal : boolean = false;

  }
  Port item_to_be_dealt = {

  }

}
Component Media_Shop = {
  Port processInternetOrders = {
    Property required : boolean = true;
  }
  Port adaptability = {
    Property required : boolean = true;
  }
}
Connector c1 : ConnDependency = new ConnDependency extended with {
}
Connector c2 : ConnDependency = new ConnDependency extended with {
}
Connector c3 : ConnDependency = new ConnDependency extended with {
}
Connector c4 : ConnDependency = new ConnDependency extended with {
}
Connector c5 : ConnDependency = new ConnDependency extended with {
}

```

```

Connector c6 : ConnDependency = new ConnDependency extended with {
}
Connector c7 : ConnDependency = new ConnDependency extended with {
}
Connector c15: ConnDependency = new ConnDependency extended with {
}
Connector c17: ConnDependency = new ConnDependency extended with {
}
Attachment Produce_Statistics.produce_statistics to c1.dependee;
Attachment Adaptation.adaptation to c2.dependee;
Attachment Query_Database.query_database to c3.dependee;
Attachment Query_Database.item_to_be_selected to c4.dependee;
Attachment Media.query_database to c3.depender;
Attachment Media.catalog_consulting to c4.depender;
Attachment Query_Database.item_to_be_dealt to c5.dependee;
Attachment Media.catalog_consulting to c5.depender;
Attachment Adaptation.availability to c6.dependee;
Attachment Media.pMedia to c2.depender;
Attachment Media.pMedia to c6.depender;
Attachment Media.pMedia to c1.depender;
Attachment Adaptation.security to c7.dependee;
Attachment Media.pMedia to c7.depender;
Attachment Media_Shop.processInternetOrders to c15.dependee;
Attachment Media.pMedia to c15.depender;
Attachment Media_Shop.adaptability to c17.dependee;
Attachment Adaptation.adaptation to c17.depender;
}

Family Family1 = {

Property Type SoftgoalType = Enum {make, somePos, help, unknown,
break, somNeg, hurt};
Connector Type ConnDependency = {
    Role dependee = {

    }
    Role depender = {

    }
}
}
}

```

## A.2 Textual representation of the BTW System (Chapter 4)

```
import families/Family1.acme;

System btw : Family1 = new Family1 extended with {

  Component BTW = {
    Port providing_advice_service = {

      Property required : boolean = true;

    }
    Port precise_advice = {

      Property required : boolean = true;

    }
    Port add_advice = {

      Property required : boolean = true;

    }
    Port add_advice_content = {

      Property required : boolean = true;

    }
    Port relevance = {

      Property provided : boolean = true;

      Property softgoal : Family1.SoftgoalType = unknown;

    }
    Port require_explicit_feedback = {

      Property provided : boolean = true;

    }
    Port filter_advice = {
```

```

        Property provided : boolean = true;

    }

    Port show_advices = {

        Property provided : boolean = true;

        Property goal : boolean = false;

    }

}

Component User_Access_Control = {
    Port security = {

        Property provided : boolean = true;

        Property softgoal : Family1.SoftgoalType = unknown;

    }

}

Component Mapping_Handler = {
    Port select_placemark = {

        Property provided : boolean = true;

    }

}

Component Map_Information_Publisher = {
    Port precise_advices = {
        Property provided : boolean = true;
        Property softgoal : Family1.SoftgoalType = unknown;

    }

    Port information_to_map = {

        Property provided : boolean = true;

        Property goal : boolean = false;

    }

}

```

```

    }

}

Component Advice_Receiver = {
    Port relevant_advice = {

        Property required : boolean = true;

    }

    Port furnish_feedback = {

        Property required : boolean = true;

    }

    Port trace_route = {

        Property required : boolean = true;

    }

    Port advice_to_be_received = {

        Property required : boolean = true;

    }

}

Connector c1 : ConnDependency = new ConnDependency extended with {

}

Connector c2 : ConnDependency = new ConnDependency extended with {
    Role depender = {

        Property required : boolean;

    }

}

Connector c3 : ConnDependency = new ConnDependency extended with {

}

Connector c4 : ConnDependency = new ConnDependency extended with {

```

```

}
Connector c5 : ConnDependency = new ConnDependency extended with {

}
Connector c6 : ConnDependency = new ConnDependency extended with {

}
Connector c7 : ConnDependency = new ConnDependency extended with {

}
Connector c8 : ConnDependency = new ConnDependency extended with {

}
Attachment User_Access_Control.security to c1.dependee;
Attachment BTW.providing_advice_service to c1.depender;
Attachment BTW.precise_advice to c2.depender;
Attachment      Map_Information_Publisher.precise_advices      to
c2.dependee;
Attachment BTW.add_advice to c3.depender;
Attachment      Map_Information_Publisher.information_to_map      to
c3.dependee;
Attachment Mapping_Handler.select_placemark to c4.dependee;
Attachment BTW.add_advice_content to c4.depender;
Attachment BTW.relevance to c5.dependee;
Attachment Advice_Receiver.relevant_advice to c5.depender;
Attachment Advice_Receiver.fournish_feedback to c6.depender;
Attachment BTW.require_explicit_feedback to c6.dependee;
Attachment BTW.filter_advice to c7.dependee;
Attachment Advice_Receiver.trace_route to c7.depender;
Attachment Advice_Receiver.advice_to_be_received to c8.depender;
Attachment BTW.show_advices to c8.dependee;
Attachment BTW.precise_advice to c2.depender;
}

```