Integrating Organizational Requirements and Socio-Intentional Architectural Styles

Lúcia R. D. Bastos [1], Jaelson F. B. Castro [1], John Mylopoulos [2]

[1] *Centro de Informática, Universidade Federal de Pernambuco, Av. Prof. Luiz Freire S/N, Recife PE, Brazil 50732-970, +1 5581*

{lrdb, jbc}@cin.ufpe.br

[2] *Dept. of Computer Science University of Toronto, 10 King's College Road Toronto M5S3G4, Canada, +1 416 978 5180*

jm@cs.toronto.edu

Abstract

Software systems of today are characterized by increasing size, complexity, distribution, heterogeneity, and lifespan. Understanding and supporting the interaction between software requirements and architectures remains one of the challenging problems in software engineering research. To address these challenges we are proposing an integration framework developed within the context of the Tropos project. The proposal aims at identifying the key architectural elements and the dependencies among those elements, based on the stated system requirements.

1. Introduction

Requirements Engineering and Software Architecture have become established areas of research, education and practice within the software engineering community.

Evolving and elaborating system requirements into a viable software architecture satisfying those requirements is still a difficult task, mainly based on intuition. It also remains a challenge to show that a given software architecture satisfies a set of functional and non-functional requirements. This is somewhat surprising, as software architecture has long been recognised to have a profound impact on the achievement of non-functional goals ("ilities") such as availability, reliability, maintainability, safety, confidentiality, evolvability, and so forth.

In this work we show an approach for this integration of systems requirements and software architectures within the context of the Tropos project, an information system development framework which is requirements-driven in the sense that it adopts concepts used during early requirements analysis. To model and understand issues of the application domain (the enterprise) we use the i* technique [2],[3], which allows a better description of the organizational relationships among the various agents of a system as well as an understanding of the rationale of the decisions taken. In the architectural design we use a catalogue of socio-intentional structures adopting a set of architectural styles for multi-agent systems motivated in organization theory and strategic alliances [4], [5], [6].

The paper is structured as follows. Section 2 presents the Tropos ontology, including a modeling framework for requirements analysis namely the *i** technique, and the organizational-inspired architectural styles. Section 3 emphasize the existence of conceptual differences between requirements and architecture. Section 4 introduces the baseline of our proposal to integrating organizational requirements and socio-intentional styles. Finally, Section 5 summarizes the related work, concludes the papers with contributions and points to further work.

2. The Tropos Methodology

The Tropos methodology adopts the view of information systems as social structures. By social structures, we mean a collection of social actors, human or software, which act as agents, positions, or roles and have social dependencies among them. Tropos is intended as a seamless methodology tailored to describe both the organizational environment of a system and the system itself in terms of the same concepts.

The Tropos ontology is described at three levels of granularity [1]. At the lowest (finest granularity) level, Tropos adopts concepts offered by the i* organizational modeling framework [2], [3], [4], such as actor, agent, position, role, and social dependency.

At a second, coarser-grain level the ontology includes possible social patterns, such as mediator, broker and embassy. At a third, more macroscopic level the ontology offers a set of organizational styles inspired by organization theory and strategic alliances literature. All three levels are defined in terms of the i* concepts.

Tropos methodology spans four phases:

- Early requirements - concerned with the understanding of a problem by studying an organizational setting; the output is an organizational model that includes relevant actors, their goals and dependencies.
- Late requirements - the system-to-be is described within its operational environment, along with relevant functions and qualities.
- Architectural design - the system's global architecture is defined in terms of subsystems, interconnected through data, control and dependencies.
- Detailed design - behavior of each architectural component is defined in further detail.

More details about *Tropos Methodology* can be found in [1].

2.1 Requirements in the I* framework

This section will review the main concepts of the i* technique [2], [4]. It is a framework, which focuses on the modeling of strategic actor relationships of a richer conceptual model of business processes in their organizational settings. The ontology of the i* technique [4] caters to some of these advanced concepts. It can be used for: (i) obtaining a better understanding of the Organizational relationships among the various system agents; (ii) understanding the rationale of the decisions taken; and (iii) illustrating the various characteristics found in the early phases of requirements specification. According to this technique, the participants of the organizational setting are actors with intentional properties, such as, goals, beliefs, abilities and compromises. These actors depend upon each other in order to fulfill their objectives and have their tasks performed.

The i* technique consists of two models: The Strategic Dependency Model (SD) and the Strategic Rationale Model (SR).

The Strategic Dependency Model (SD) consists of a set of nodes and links connecting them, where nodes represent actors and each link indicates a dependency between two actors. Hence, a model is described in terms of network of dependency relationships among various actors, capturing the motivation and why of activities. We can distinguish, four types of dependencies, three of them related to existing intentions – *goal dependency*, *resource dependency* and *task dependency* – while the fourth is associated with the notion of non-functional requirements, the so called *soft-goal dependency*. In the *goal dependency*, an agent depends on another one to provide the desired condition, and it does not worry about how this condition is achieved. In the *resource dependency*, the agent depends on the availability of physical resource or information. In the *task dependency*, the agent informs the other what (and how) should be done. The *soft-goal dependency* is similar to the *goal dependency*, except that the condition is not precisely defined at the start of the process, i.e., the goals in a sense involves subjective aspects, that gradually are clarified during the development process. This type of dependency provides an important link connecting two important aspects in software engineering: (i) the technical and (ii) managerial side. We still can identify different degrees of dependencies: open, committed and critical [5]. We can distinguish actors as agents, *roles* and *positions*. An agent is an actor with concrete physical manifestations. It is a person or artificial agents (hardware/software). A role is an abstract characterization of the behavior of a social actor within some specialized context, domain or endeavor. A position is a set of roles typically played by one agent. Moreover we can analyze opportunities and vulnerabilities of the chain dependency [3].

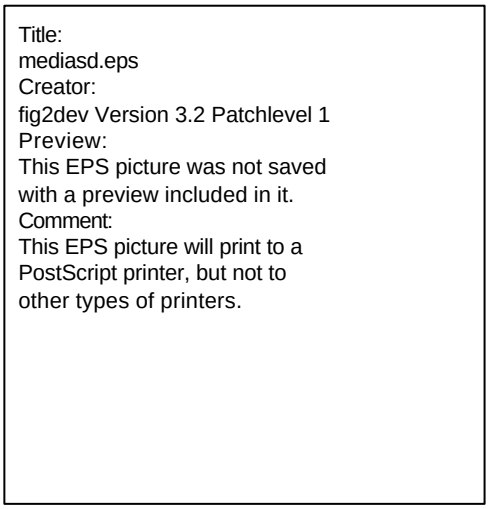

Figure 1 – SD model for Media Shop

In the Figure 1, we have the Strategic Dependency (SD) model of the e-commerce example. The *Media Shop* is a store selling and shipping different kinds of media items such as books, newspapers, magazines, audio CDs, videotapes, and the like [1]. To increase

market share, *Media Shop* has decided to open up a B2C retail sales front on the internet. With the new setup, a customer can order *Media Shop* items in person, by phone, or through the internet. The system has been named *Medi@* and is available on the world-wide-web using communication facilities provided by *Telecom Cpy*. It also uses financial services supplied by *Bank Cpy*, which specializes on on-line transactions. *Medi@* system is introduced as an actor in this strategic dependency model depicted.

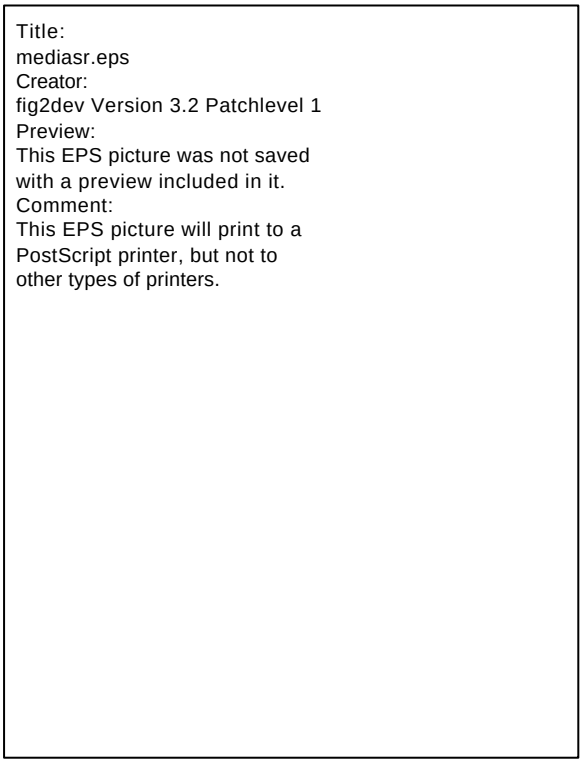

Figure 2 – SR model for Medi@

The second model of the technique i***** is the Strategic Rationale Model (SR). It is used to: (i) describe the interests, concerns and motivations of participants process; (ii) enable the assessment of the possible alternatives in the definition of the process; and (iii) research in more detail the existing reasons behind the dependencies between the various actors. Nodes and links also are part of this model. It includes the previous four types of nodes (present in the SD model): *goa*l, *tas*k, *resource* and *soft-goa*l. There are two new types of relationship, *means-end* that suggests that there may be other means of achieving the objective (alternatives) and *task-decomposition* that describes what should be done in order to perform a certain task.

The analysis in Figure 2 focuses on the software (Media), instead of an external stakeholder. The figure postulates a root task *Internet Shop Managed* providing sufficient support (++) [13] to the softgoal *Increase Market Share*. That task is firstly refined into goals *Internet Order Handled* and *Item Searching Handled*, softgoals *Attract New Customer*, *Secure* and *Usable* and tasks *Produce Statistics* and *Maintenance*. *Internet Order Handled* is achieved through the task *Shopping Cart*, which is decomposed into subtasks: *Select Item*, *Add Item*, Check *Out*, and *Get Identification Detail*. These are the main process activities required to design an operational on-line shopping cart. More details can be founded in [1].

In next section we will detail the organizational-inspired architectural styles *Tropos*, which consider information systems as social structures all along the development life cycle.

2.2. Socio-Intentional Architectural Styles

A system architecture constitutes a relatively small, intellectually manageable model of system structure, which describes how system components work together. Unfortunately, traditional architectural styles for e-business applications [12],[13] focus on web concepts, protocols and underlying technologies but not on business processes nor non functional requirements of the application. As a result, the organizational architecture styles are not described nor the conceptual high-level perspective of the e-business application.

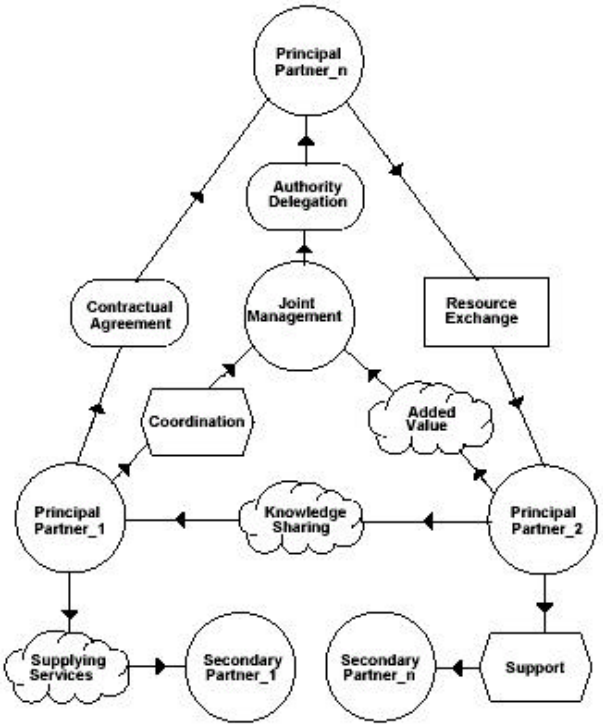

Figure 3 – The joint venture pattern

*Tropo*s has defined organizational architectural styles [1],[5],[6],[7] for agent, cooperative, dynamic

and distributed applications to guide the design of the system architecture. These architectural styles *(pyramid, joint venture* (Figure 3)*, structure in 5, takeover, arm's length, vertical integration, co-optation, bidding)* are based on concepts and design alternatives coming from research on organization management. The proposal is to use human organizations as a metaphor to suggest a set of generic styles for agent systems, with a preference for organizational design theories over social emergence theories.

For example, the joint venture architectural style in Figure 3. The joint venture style is a more decentralized style based on an agreement between two or more principal partners who benefit from operating at a larger scale and reuse the experience and knowledge of their partners. Each principal partner is autonomous on a local dimension and interacts directly with other principal partners to exchange services, data and knowledge. However, the strategic operation and coordination of the joint venture is delegated to a Joint Management actor, who coordinates tasks and manages the sharing of knowledge and resources. Outside the joint venture, secondary partners supply services or support tasks for the organization core.

The organizational architectural styles have been described in UML, in order to provide detailed representation in architectural phase of Tropos Methodology, as well as to represent the organizational styles into a industrial notation [16].

3. The Gap Between Requirements and Architectural Description

The inter-dependencies and constraints between requirements elements and architectural elements are thus not well-understood and subsequently only little guidance is available in bridging requirements and architecture. The semantic gap between requirements and software design is substantial [12].

Requirements Engineering is concerned with identifying the purpose of a software system, and the contexts in which it will be used. Software architecture is related to the principled study of large grained software components, including their properties, relationships, and pattern of combination [9]. In addition to specifying the structure and topology of the system, the architecture should show the intended correspondence between the system requirements and elements of the constructed system. It can additionally address system-level properties such as capacity, throughput, consistency, and component compatibility [14].

The existence of conceptual differences between what to do (requirements) versus how to do it (architecture, design and code) constitutes a semantic gap. Filling this gap requires better models and notations for the intermediate step. There are some critical challenges when trying to reconcile requirements and architectures [8]:

- Requirements are frequently captured informally in a natural language. On the other hand, entities in a software architecture specification are usually specified in a formal manner [11].
- System properties described in non-functional requirements are commonly hard to specify in an architectural model [11].
- Iterative, concurrent evolution of requirements and architectures demands that the development of an architecture be based on incomplete requirements. Also, certain requirements can only be understood after modeling and even partially implementing the system architecture [12].
- Mapping requirements into architectures and maintaining the consistency and traceability between the two is complicated since a single requirement may address multiple architectural concerns and a single architectural element may have numerous non-trivial relations to various requirements.
- Real-world, large-scale systems have to satisfy hundreds, possibly thousands of requirements. It is difficult to identify and refine the architecturally relevant information contained in the requirements due to this scale.
- Requirements and the software architecture emerge in a process involving heterogeneous stakeholders with conflicting goals, expectations, and terminology. Supporting the different stakeholders demands finding the right balance across these divergent interests.

The following section outlines the basis of our approach.

4. The Integrating Framework Proposal

This section describes an informal four-steps process to address the transition between requirements and architectural design. This proposal is a framework to identifying and mapping the architectural decision from a requirements specifications.

4.1. Mapping Architectural Elements from *i**

This proposal focuses on finding a systematic process to support the transition from requirements specification to architectural design.

As showed in Figure 4 the proposal are composed by two modules: *i* Architectural Extension and*

Integration Process. Our approach for integration process takes as input a goal oriented requirements specifications in i* technique and returns as output an architectural model. The main concerns are related to the identification, classification and support a variety of architectural elements from system requirements.

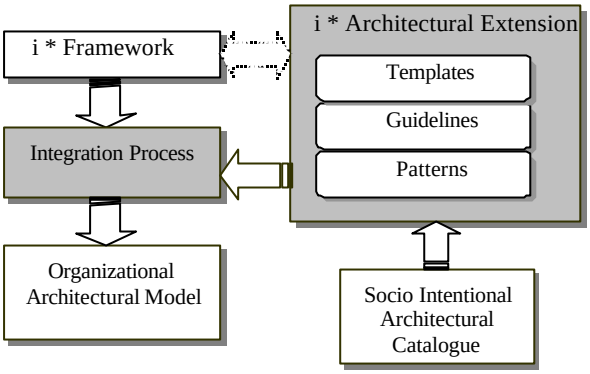

Figure 4 – i* Architectural extension

This extension includes:

– Templates – To extend and refine the properties from i* architectural elements (possibly actors, goals, softgoals resource, task, dependency and links). The identified architectural elements from i* framework are:
  1. Components - The computational elements (possibly systems actors) of the architecture bound together by connectors;
  2. Connections - The relations between components (possibly dependencies between actors or relationships to archive goals, like means-end or task decompositions);
  3. Constraints – assertions and constraints that apply to the entire system or components (possibly extracted from the non-functional requirements, goals, dependency sequences or architectural patterns);

– Guidelines – To support the mapping from SR description into organizational architectural styles elements.
– Architectural Patterns – Compositions or styles in which architectural elements are connected in a particular way. In this work we are using the architectural styles of the socio intentional catalogue (e.g., Joint Venture style).

Figure 5 shows the four-steps Integration Process to mapping and relating i* systems requirements and organizational architectural elements:

– Step 1: Capturing the architectural requirements. This step covers an analysis using as input the i* requirements model and architectural guidelines to identifying architectural elements and capture additional architecture-relevant information. As output we have some templates for architectural elements;
– Step 2: Applying the NFR Framework to select among the socio-intentional architectural style using the non-functional requirements;
– Step 3: Relating i* architectural requirements with the architectural elements from the socio-intentional catalogue applying the guidelines;
– Step 4: Generating the i* architectural model.

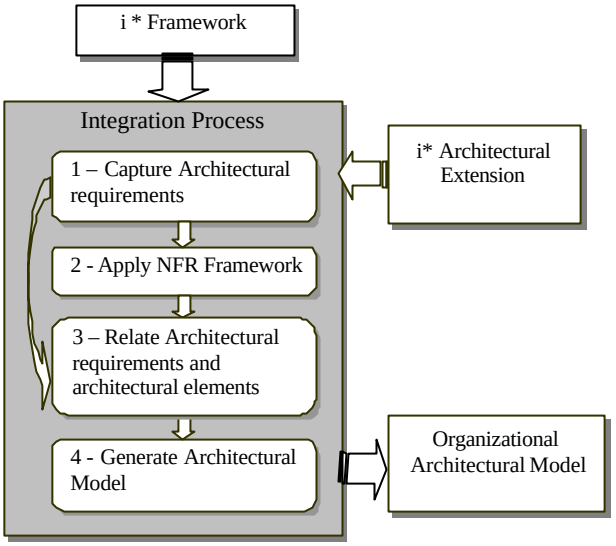

Figure 5 – Integration process

Capture Architectural Requirements - The primary activity is to identify an i* architectural elements composed by requirements elements, showing in Table 1, with complementary architectural definitions.

Table 1 – Mapping the i* architectural elements

I* Elements	**Architectural Elements**
Actor	*System component*
Task	*Responsibility*
Goal	*Responsibility/ constraint*
Soft-goal	*Constraint*
Dependency	*Connection/Relationship/constraints*
Resource	*System entity*
Link	*Connection*

In the sequence we show an architectural template example for the component Medi@ system showed in Figure 2.

The Table 2 shows the partial template definition of a component. The *Name* attribute is the i* specification from which the element (actor) is derived. In our example "Medi@" it is a system component. The *Responsibilities* attribute is a list of assignment of system responsibilities (tasks and goals), the sub-components that implement the component. The *Interface* attribute denotes the connectors (dependencies) between others components or sub components. The *Constraints* attribute denotes which

goals the sub-components satisfy, the soft-goals list and architectural style selected.

Table 2 – Architectural templates

Type: System Component
Name: Medi@
Responsibilities: {list of task and goal}
Interface: {list of dependencies)
Constraints: {Assertions in use, relationship};
…….
Architectural Pattern: {organizational style}
Composed of: {components}
{responsibilities}
……..

The organizational architectures offer a set of design parameters (such direct supervision, standardization of skills, outputs and work processes) that can influence the division of labor and the coordination mechanisms. This design parameters, include, among others task assignments. Tasks are partially ordered sequences of steps intended to accomplish some goal. Tasks can be decomposed into goals and/or subtasks, whose collective fulfillment completes the task. These decompositions also allow to identify actors that can accomplish a goal, carry out a task, or deliver some resource needed by another actor. Fulfillment of an actor's obligations can be accomplished through delegation and through decomposition of the actor into components actors.

To define the roles in the organizational architectures we propose an initial classification of the responsibilities (tasks and goals) as show in Table 3.

Table 3 – Task type

Basic	The input, processing and output associated with the running the organization
Manager	The coordination and managerial activities
Controller	Standardization of work process
Support	The non-operational services that are outside the basic flow of operational tasks.

Applying NFR Framework - An important task during architectural design is to select among alternative architectural styles using as criteria the desired qualities identified in the previous phase (Late Requirements). They will guide the selection process of the appropriate architectural style. The analysis involves refining these qualities, represented as softgoals, to sub-goals that are more specific and more precise and then evaluating alternative architectural styles against them, as showed in Figure 6.

The analysis resulting in a softgoal dependency graph is intended to make explicit the space of alternatives for fulfilling a top-level attribute. The organizational patterns are represented as operationalized attributes (saying, roughly, "fulfilled by the pattern structure-in-5/joint-venture") [7].

The evaluation results in contribution relationships from the social structures to the quality attributes, labeled "+", "++", "-", "--" that mean respectively *partiallysatisfie*d, *satisfie*d, *partially denied* and *denie*d. Design rationale is represented by claims drawn as dashed clouds. They make it possible for domain characteristics such as priorities to be considered and properly reflected into the decision making process. Exclamation marks are used to mark priority attributes while a check-mark " " indicates an accepted attribute and a cross "**Õ**" labels a denied attribute.

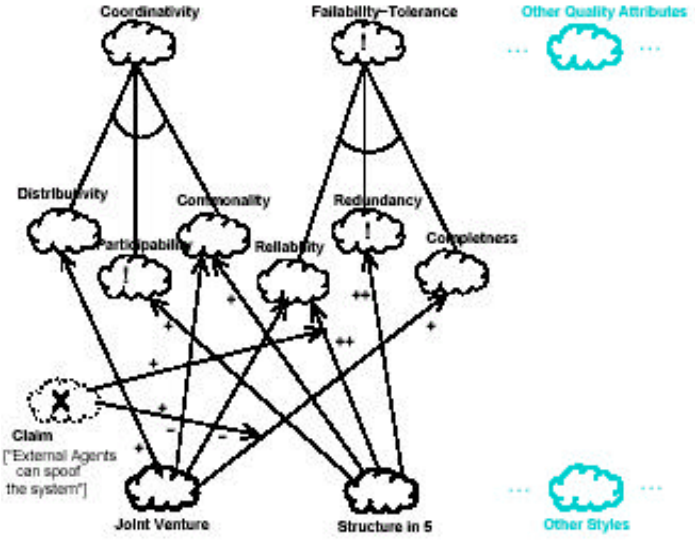

Figure 6 – Partial evaluation for selecting architectural styles

More details about the selection and non-functional requirements decomposition process can be found in [6],[7].

Relating the i* architectural elements and Socio Intentional elements - The architectural level of design requires a different form of abstraction to reveal high-level structure. In particular, should be possible to represent as first class abstractions new architectural patterns and new forms of interaction between architectural requirements elements, so that the distinct roles of each requirement elements in the structure are clearer.

The organizational pattern adopts the abstractions offered by organizational theory. The structure of an organization defines the roles of various intentional components (actor), their responsibilities, defined in

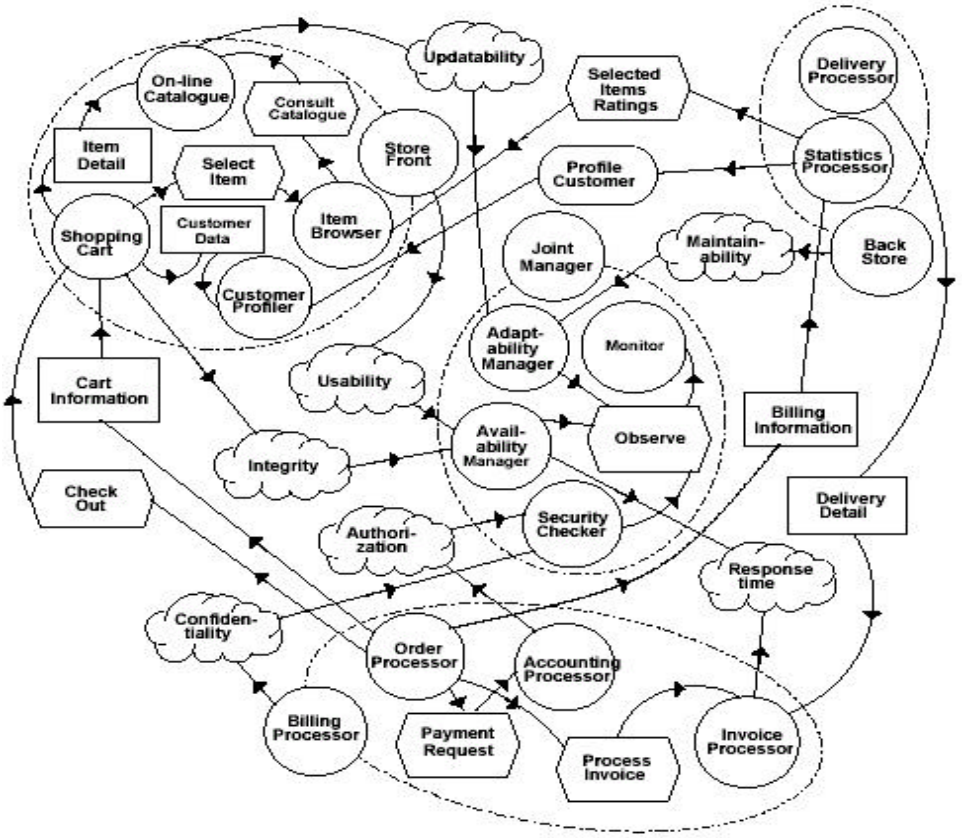

Figure 7 – Medi@ system as joint venture architecture

terms of tasks and goals they have assigned and resources they have been allocated.

A *role* is an abstract characterization of the behaviour of an actor within some specialized context, domain or endeavour. Its characteristics are easily transferable to other actors. Dependencies are associated with a role when these dependencies apply regardless of who plays the role. In order to describing this relationship it is necessary to analyse the responsibilities and roles in the system requirements.

Our work consists of extending the i* with guidelines to support the mapping of i* requirements elements to i* architectural elements.

Guideline 1.1: The *i* systems (or i* roles) can be mapped to a system component in architectural model.*

For instance, the Figure 7 suggest a possible assignment of system responsibilities for the business-to-consumer (B2C) part of Media System. Following the joint venture style, the architecture is decomposed into three principal partner actor (*Store Front, Billing Processor and Back Store*) .

Guideline 1.2**:** *The i* relationship between systems (or roles) can be mapped as interface in architectural model.*

The partners control themselves on a local dimension for exchanging, providing and receiving services, data and resources with each other. For instance, the *Store Front* interacts primarily with the customers and provides them with a usable front-end web application for consulting and shopping media items. See Figure 7.

Guideline 1.3**:** *The i* task (or goal decomposition into task) can be mapped as responsibility in architectural model.*

For instance, some of the responsibilities (see table 1) in Medi@ system are *"Internet Shop Managed"*, "*Secure Form Order*", "*Internet Orders Handled*", "*Maintenance*", as seen in Figure 2.

Guideline 1.4**:** *The i* tasks-type (or goal-type) defines the roles of various intentional architectural components (actor).*

For instance, *Billing Processor* is in charge for the secure management of orders and bills, and other financial data. And the *Joint Manage*r manages the system on a global dimension. See Figure 7.

Further guidelines are required to describe a complete mapping between requirements and architecture. Of course not all concepts captured in the requirements phase will correspond to architectural system models. The models do not have a one-one relationship; many elements of the organizational requirements model are not part of the architectural model, since not all of the organizational tasks require a software system. Many tasks contain activities that are performed outside the software system, and so do

not become part of the architectural system model. Likewise, many elements in the architectural model comprise detailed technical software solutions and constructs that are not part of the organizational model.

5. Conclusion

The relationship between requirements and architectures has received increased attention recently [15]. A number of goal-based requirements approaches, most notably KAOS [9] [10] and the NFR framework [13], have proposed the explicit use of the notion of 'goals' to structure system requirements and architecture. A proposal KAOS/APL presented in [15] has suggested the use of intermediate descriptions between requirements and architecture that they call 'architectural prescriptions', which describe the mappings relationship between requirements and architectures. The CBSP approach [8] explores the relationships between software requirements and architectures, and proposes a technique to reconciling mismatches between requirements terminology and concepts with those of architectures.

The purpose of this paper is to present our work on the development of a framework to complement the specification of architectural elements and mapping the relationship between requirements and architectural elements using a set of organizational styles.

Future research directions will extend the architectural catalogue with classical software pattern proposed in the literature (piper-and-filters, layers, event-based) .

6. References

[1] Castro, J., Kolp, M., Mylopoulos, J.: "Towards Requirements Driven Information Systems Engineering: The Tropos Project", In Information Systems, Vol. 27. Elsevier, Amsterdam, The Netherlands (2002) 365–389.

[2] Yu, E.: "Modelling Strategic Relationships for Process Reengineering". Ph.D. thesis, Department of Computer Science, University of Toronto, Canada (1995).

[3] Yu, E., Liu, L.: "Modelling Trust in the *i** Strategic Actors Framework". Proceedings of the 3rd Workshop on Deception, Fraud and Trust in Agent Societies. Barcelona, Spain (at Agents2000), June 3-4, 2000.

[4] Yu, E.: 'Agent Orientation as a Modelling Paradigm". Wirtschaftsinformatik. 43(2) April 2001. pp. 123-132.

[5] 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) 5–12.

[6] Kolp, M. and Giorgini. P.: "Information Systems Development through Social Structures". Submitted to the 14th International Conference on Advanced Information Systems Engineering (CAiSE'02), Toronto, Canada, May 2002.

[7] M. Kolp, P. Giorgini and J. Mylopoulos. "Organizational Patterns for Early Requirements Analysis". 15th International Conference on Advanced Information Systems Engineering (CAiSE'03), Velden, Austria, June 2003.

[8] Grünbacher, P., Egyed, A. and Medvidovic, N.: "Reconciling Software Requirements and Architecture: The CBSP Approach". Proceedings RE'01, 5th International Symposium on Requirements Engineering. Toronto, Canada. August 2001.

[9] Lamsweerde, A. van.: "Requirements Engineering in the Year 00: A Research Perspective". 22nd Proceedings of International Conference on Software Engineering, Limerick, Ireland. Jun. 2000.

[10] Lamsweerde, A. van.: "Goal-Oriented Requirements Engineering: A Guided Tour". Proceedings RE'01, 5th International Symposium on Requirements Engineering. Toronto, Canadá. August 2001, 249-263.

[11] Medvidovic N., Taylor R.N.: "A Classification and Comparison Framework for Software Architecture Description Languages". IEEE Transactions on Software Engineering, 26/1:70-93, 2000.

[12] Nuseibeh, B.: "Weaving the Software Development Process between Requirements and Architectures". First International Workshop From Software Requirements to Architectures (STRAW'01). May, 2001.

[13] Chung, L., Nixon, B. A., Yu, E. and Mylopoulos, J.: "Non-Functional Requirements in Software Engineering". Kluwer Publishing, 2000.

[14] Shaw, M.: "Abstraction for Software Architecture and Tools to Support Them". IEEE Transactions on Software Engineering, 21(4): pp.314-335, April 1995.

[15] STRAW'01. Proceedings of First International Workshop From Software Requirements to Architectures (STRAW'01), 2001. http://www.cin.ufpe.br/~straw01.

[16] Silva, C.T.L.L and Castro, J.F.B.: "Detailing Architectural Design in the Tropos Methodology". Proceedings of the 15th Conference on Advanced Information System Engineering – CAISE 03. Klagenfurt, Velden, Austria, 2003.