Support for Requirement Traceability: The Tropos Case

Rosa Candida Pinto, Carla Silva and Jaelson Castro

Universidade Federal de Pernambuco – Centro de Informática
Recife (PE) – Brazil - 50732-970
{rccp, ctlls, jbc}@cin.ufpe.br

Abstract. *For many years, the research and business communities have agreed that the traceability is very important in the software development process. It helps the impact analysis of required changes, improving the efficient management of software projects and hopefully improving the software quality. Software development methodologies supporting requirement traceability can develop and maintain higher quality software with less cost. Our research aims to support traceability through the agent-oriented software lifecycle. In particular, this paper proposes a requirement traceability process to be used in the requirements specifications and system architecture models of the Tropos framework. An e-commerce example is used to demonstrate the applicability of the proposed approach.*

1 The Introduction

If we are to be successful in the development of the next generation of agent-oriented software systems we must deal with the critical issue of requirement traceability [Castor, Pinto, Castro and Silva 2004]. Requirement Traceability refers to the ability to ensure continued alignment between stakeholders' requirements and various outputs of the system development process. A requirements traceability process describes and follows the life of a requirement, in both a forward and backward direction (i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases) [Gotel 1996]. Failure to do so will imply in higher costs for maintaining software systems. However, empirical studies of traceability needs and practices in industrial organizations have indicated that traceability support is not always satisfactory. As a result, traceability is rarely established in existing industrial setting [Zisman, Spanoudakis, Pérez-Miñana and Krause 2003].

This paper proposes a requirement traceability process [Pinto, Silva and Castro 2005] to be used in the requirements specifications and system architecture models of the Tropos framework [Castro, Kolp and Mylopoulos 2002] [Bresciani, Giorgini, Giunchiglia, Mylopoulos and Perini 2004] [Giorgini, Kolp, Mylopoulos, Castro 2005]. In particular, we present a general traceability framework [Toranzo and Castro 1999] [Toranzo 2002] applied in the context of agent-oriented development [Wooldridge 1999] [Wooldridge 2002] to enhance the *Tropos*[1] framework in order to support traceability through the agent-oriented software lifecycle.

The structure of this paper is as follows: Section 2 describes the Tropos approach for agent-oriented development in Section 3 presents the meta-models that are required to support traceability. In Section 4, we defined a process that can be used to

[1] For further detail and information about *Tropos* project, see http://www.troposproject.org

derive traceability information in the context of the Tropos approach. In Section 5, we apply Tropos to a case study and show all phases of the proposed requirement traceability process. Section 6 describes related work and finally Section 7 concludes the paper

2 TROPOS

Tropos is a requirements-driven framework in the sense that it proposes to use the concepts used during early requirements analysis at various stages of the software development lifecycle. Tropos rests on the idea of using requirements modeling concepts to build a model of the system-to-be within its operational environment [Castro, Kolp and Mylopoulos 2002]. This model is incrementally refined and extended, providing a common interface to the various software development activities. The models also serve as a basis for documentation and evolution of the software system. Tropos [Bresciani, Giorgini, Giunchiglia, Mylopoulos and Perini 2004; Giorgini, Kolp, Mylopoulos, Castro 2005] spans four phases: *Early requirements*, Late *requirements*, *Architectural design*, Detailed *design*.

Tropos adopts the concepts and models offered by *i** framework [Yu 1995] which offers concepts such as *actor* (actors can be *agents*, *positions* or *roles*), as well as social dependencies among actors including *goal*, *softgoal*, *task* and *resource* dependencies.

The i* framework also includes the Strategic Dependency (SD) model for describing the network of relationships among actors, as well as the Strategic Rationale (SR) model for describing and supporting the reasoning that each actor goes through concerning its relationships with other actors by using a means-ends analysis. Due to space limitation, we do not explain the phases of Tropos methodology in detail. However, an overview of them is presented in the case study (section 4). An interested reader can find a full description of the Tropos' phases in [Castro, Kolp and Mylopoulos 2002; Bresciani, Giorgini, Giunchiglia, Mylopoulos and Perini 2004].

In the sequel we sketch the models supporting requirement traceability that will be used in an e-commerce case study.

3 Requirements traceability Meta-Model

The requirement engineering process supports the understanding of the stakeholders' goals, as well as the refinement of these goals into requirements. An important task of this process is keeping track of bi-directional relationships between requirements and stakeholders' motivations as well as between requirements and development process artifacts in order to facilitate the maintenance and verification of the system [Ramesh and Jarke 2001] [Gotel 1996]. Systems evolution requires a better understanding of the requirements, which can only be achieved by the agility to trace back to their sources. Moreover, test procedures, if traceable to requirements or design, can be modified when errors are discovered.

As a consequence of these different uses and perspectives on traceability, there are wide variations on the format and content of traceability information across different system development efforts. Thus, a reference model is needed to facilitate the construction of a requirement traceability scheme [Toranzo and Castro 1999].

In this paper requirement traceability is defined as the ability to describe and follow the life of a requirement, in both forward and backward direction. This process happens within the context of four composite, interrelated and parallel information layers:

External Layer represents elements (such as constraints) on the universe where the organization is inserted.

Organizational Layer represents an element (with goals and decisions) of the universe.

Managerial Layer is related to activities that can be performed by an organization (such as management of people, budge and contracts).

Development Layer is related to artifacts produced by a development process [Toranzo 2002].

These layers define classes that composes the traceability reference model. They are related to each other by means of associations named *satisfy*, *resource*, *responsibility*, *represents* and *allocated_to*. *Satisfy* specifies that an activity must be realized in the target element in order to fulfill the needs of the source elements connected to it. *Resource* represents information needed to understand a specific element. *Responsibility* associates stakeholders with the software elements, decisions or activities. *Represents* captures the requirements mapping into other languages, such as modeling or programming. A*llocated_to* defines which component will take care of what requirement. The notation used to represent the proposed associations is based on UML (Unified Modeling Language) [Booch, Rumbauch and Jacobson 1999] stereotypes. Moreover, the reference model is divided into two sub-models for clarity.

Requirement Management sub-model (Figure 1) helps requirements understanding, capture, tracking, validation and verification. We can also represent the CONSTRAINTs imposed by the environment on the system.

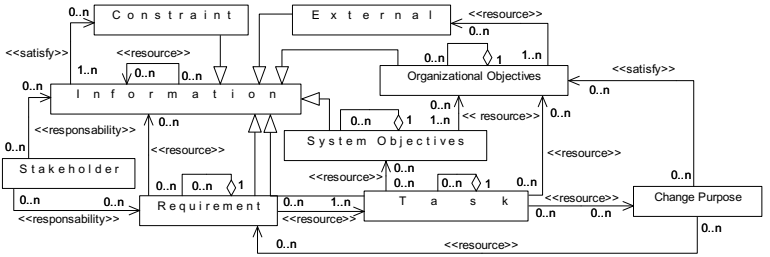

Figure 1. Requirements Management Sub-model

SYSTEM OBJECTIVES represents goals to be achieved by the software system. The TASK class represents the management activities to be performed by the project manager. To be effective, requirements have to be associated to the stakeholders that propose them. This is illustrated in the model by the STAKEHOLDER class and its associations to the REQUIREMENT class. To control the changes in a system, the model includes the CHANGE PURPOSE class associated to TASK, REQUIREMENT and ORGANIZATIONAL OBJECTIVES classes.

Design sub-model is used to refer to any activity that creates artifacts, including implementation (Figure 2).

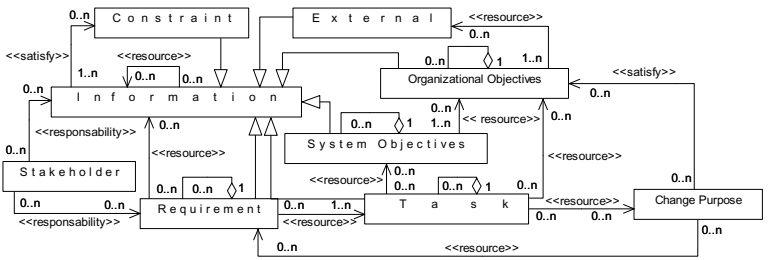

Figure 2. Design Sub-model

This sub-model indicates that the DESIGN ELEMENT class is the root of DIAGRAM, PROGRAM and SUBSYSTEM classes. PROGRAMs *represent* REQUIREMENTs that are *resources* for TEST. Due to various reasons, changes may be required and should be recorded by the CHANGE PURPOSE and its association to DESIGN ELEMENT indicates that it has a *resource* dependency with some design elements. TASK is always related to project management, i.e., tasks to be performed by the manager to support some organizational need, implementation of a requirement or a diagram. The *satisfy* association between the TASK and DESIGN ELEMENT expresses that the satisfaction of the tasks depends on some design elements. The *allocated_to* association between REQUIREMENT and SUBSYSTEM classes expresses that requirements are assigned to one or more subsystems.

In addition to these sub-models, Toranzo (2002) presents a *Rational model* for identification and structure of the problems and decisions made (reasoning) during the software development (Figure 3).

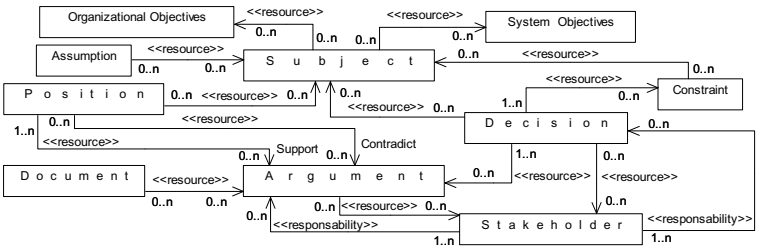

Figure 3. The Rational model

In the *Rational model*, the SUBJECT can be any problem that requires a discussion or resolution. The problem does not need to be related to a requirement as it could be related to ORGANIZATIONAL or SYSTEM OBJECTIVES. Various POSITIONs or alternatives that address the resolution should be considered and for each one of them there will be one or more ARGUMENTs that will support or contradict it. The relationships *support* and *contradict* indicate that a position can have, as a resource, none or many arguments that support or contradict it. A DOCUMENT can be a *resource* of information used as basis to some ARGUMENT. The *resource* associations between CONSTRAINT/ASSUMPTION and SUBJECT expresses that the formers can be resources of information to understand the SUBJECT. The same happens with the association between CONSTRAINT and DECISION. CONSTRAINT can also be resources of information to understand a DECISION. There are some

stakeholders who provide an argument and stakeholders who are responsible for the argument that generates the decision so we have two associations between the classes ARGUMENT and STAKEHOLDER. The associations are of type *resource* and *responsibility*.

As emphasized before, the traceability reference model is a general purpose one. If we aim to use it in connection with a certain software development approach, e.g. Tropos, we must then describe a process for guiding the creation of the traceability matrixes. This process is described in the next section.

4 The Requirements Traceability Process

The main contribution of this paper is to apply this process to the following Tropos phases: late requirements and architectural design.

In this section we sketch a process which includes three stages as follows:

1.Information Gathering (IG): we identify the information related to the four layers (external, organizational, management and development) previously described.

2.Information Structuring (ST): consists of two activities. First, we remove the instances that represent irrelevant information, as well as select instances with the same meaning. Then, we determine the relationship among the instances.

3.Definition of the Traceability Matrixes (TM): Last but not least, we define the matrixes that capture and store the relationships among the instances of the classes.

In the sequel, nine guidelines are defined. The first four ones (IG1-IG4) are related to information gathering. The proper structuring of this collected information is achieved by means of guidelines ST1 and ST2. The set of valid values for association instances are defined in ST3. The construction of the appropriate traceability matrixes is guided by TM1 and TM2.

In this work, we consider that the organizational setting was understood, during the early requirements phase and that it was decides to develop a software system. In late requirements phase we extend the conceptual model developed during early requirements to include the system-to-be. The system is described within its operational environment, along with relevant functions and qualities. The artifacts produced by this phase are the Strategic Dependency (SD) and Strategic Rationale (SR) models for the actor representing the system. During architectural design phase the system's global architecture is defined in terms of subsystems, interconnected through data, control and other dependencies. The artifact produced by this phase is the architectural design model.

As Tropos is still evolving its detailed design and implementation phases, in this paper we do not apply the new requirement traceability process to these phases. It is expected that new guidelines are to be included when these phases is properly addressed. Now, we can introduce the process guidelines in detail:

Guideline IG1. Appropriate for finding the instances of the Requirements Management sub-model classes (Figure 1) from the SD diagram of the actor representing the system. We have the following rules: (**1**) The actor which has some dependency relationship with the actor representing the system represents an instance of the STAKEHOLDER class; (**2**) If the actor representing the system is the dependee of a softgoal, resource or task dependency, the dependum is an instance of the REQUIREMENT class; (**3**) If the actor representing the system is the dependee of a goal dependency of

the actor representing the organization, then the goal is an instance of the ORGANIZATIONAL OBJECTIVES class; (**4**) If the actor representing the system is the depender of a goal dependency of the actor does not represent the organization, then the goal is an instance of the SYSTEM OBJECTIVES class; (**5**) If the actor representing the system is the depender of a (goal, softgoal, resource or task) dependency, the dependum is an instance of the EXTERNAL class.

Guideline IG2. Appropriate for finding the instances of the Requirements Management sub-model classes (Figure1) from the SR diagram of the actor representing the system. During the means-ends analysis of the actor representing the system, the following rules apply: (**1**) Each goal depicted represents an instance of the SYSTEM OBJECTIVES class; (**2**) Each task depicted represents an instance of the REQUIREMENT class; (**3**) Each softgoal depicted is a non-functional requirement and therefore represents an instance of the REQUIREMENT class; (**4**) Each resource depicted is the result of some functionality associated to a functional requirement which represents an instance of the REQUIREMENT class.

Guideline IG3. Appropriate for finding the instances of the Rational model classes (Figure 3) from the process for selecting the proper architectural style. We have the following rules: (**1**) An instance of the SUBJECT class represents an issue on which a decision must be taken; (**2**) Instances of the POSITION class represent the alternative solutions for the SUBJECT; (**3**) An instance of the ARGUMENT class represents some criteria used for choosing the proper solution; (**4**) Instances of the ASSUMPTION class represent facts that must be taken into account for choosing the proper solution; (**5**) Instances of the CONSTRAINT class represent limitations/restrictions that must be taken into account for deciding the proper solution; (**6**) An instance of the DOCUMENT class represents some information used as reference for choosing the proper solution.

Guideline IG4. Appropriate for finding the instances of the Design sub-model classes (Figure 2) from the architectural design model of the system under development. We have the following rules: (**1**) Each architectural component represents an instance of the SUBSYSTEM class.

Having gathered the relevant information, we can now proceed to the next stage of the requirement traceability process that has to do with structuring the information (ST):

Guideline ST1. Given a set of instantiated classes of the reference model, we have to structure them. Hence, we can remove those unnecessary ones. Instances with the same meaning can also be deleted.

Guideline ST2. For each pair of associated classes in the reference model, we have to instantiate the association to be later used in the correspondent traceability matrix. For example if we want to create a traceability matrix to relate REQUIREMENT instances with ORGANIZATIONAL OBJECTIVES instances we have to instantiate the <<resource>> association between them (Figure 1).

Guideline ST3. For each instance created in the ST2, we define the set of values assigned to it. For example, the dependency degree between organizational information and functional requirements can be evaluated as <H> (High), <M> (Medium) or <L> (Low).

The last stage of the requirement traceability process is the definition of the traceability matrixes (TM).

Guideline TM1. For each pair of instantiated classes which are associated in a reference model, we can create a traceability matrix.

Guideline TM2. For each created matrix, we have to analyze the system artifacts that are related to the matrix and fill the association which has been instantiated in a previous stage of the process.

In the sequel we outline the Tropos' phases through an e-commerce example and make some remarks on how the traceability issues can be addressed.

5 Case Study

Media Shop is a store selling and shipping different kinds of media items such as books, newspapers, magazines, audio CDs, videotapes, and the like. Media Shop customers (on-site or remote) can use a periodically updated catalogue describing available media items to specify their order. To increase market share, Media Shop has decided to open up a B2C retail sales front on the Internet. The system has been 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. The basic objective for the new system is to allow an on-line customer to examine the items in the Medi@ Internet catalogue, and place orders.

In the next sections we describe how the requirement traceability process previously outlined can be used in conjunction with the Tropos phases. After applying the proposed process to this example, we will be able to justify the existence of each requirement in the Medi@ system, as well as, the selection of a specific architectural style applied in the system architecture design

5.1 Applying guidelines for Information Gathering (IG)

The description provided in the previous section is sufficient for producing a model of an organizational environment. For details, see [Giorgini, Kolp, Mylopoulos, Castro 2005]. Having understood the organizational setting one can now decide to develop a software system to support it. As shown in Figure 4, actors are represented as circles; dependums -- goals, softgoals, tasks and resources -- are respectively represented as ovals, clouds, hexagons and rectangles; and dependencies have the form depender $\Rightarrow$ dependum $\Rightarrow$ dependee.

In late requirements phase we extend the conceptual model developed during early requirements to include the system-to-be, i.e., the Media@. As late requirements analysis proceeds, *Medi@* is given additional responsibilities, and ends up as the dependee of several dependencies including *Availability*, *Security* and *Adaptability softgoals* (Figure 4). For more details about the evolution of the models provided by Tropos see [Giorgini, Kolp, Mylopoulos, Castro 2005].

According to the guidelines presented in previous section, we begin to perform the traceability process from the late requirements phase. Applying **Guideline IG1**, we conclude that all the actors depending on (or depended upon) the actor representing the system (Medi@ actor in Figure 4) corresponds to stakeholders, information to be regarded in the traceability process, since they will use the system and/or be used by the

system. Thus, *Media Shop*, *Customer Media Supplier, Telecom Cpy* and *Bank Cpy* are instances of the STAKEHOLDER class. This association is extremely important in the requirement traceability process because it stores information about the stakeholders and their contributions to the system. When a change is required, the correspondent stakeholders can be questioned about possible doubts as well as conflicts can be resolved. The incoming softgoal, resource or task dependencies of the actor representing the system (Medi@ actor in Figure 4) correspond to requirements, i.e. they are needs/requests to the system. Thus, *Availability, Adaptability* and *Security* softgoals, *Browse Catalogue*, *Keyword Search* and *Place Order* tasks (Figure 6) are instances of the REQUIREMENT class.

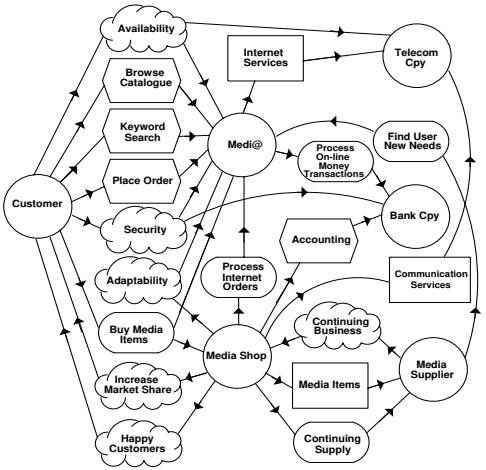

Figure 4.Strategic Dependency Diagram for Medi@ System

The incoming goal dependencies of the actor representing the system (Medi@ actor in Figure 4) from the actor representing the organization (Media actor in Figure 4) correspond to organization objectives, i.e. they are needs/requests to the system. Thus, *Process Internet Orders* (Figure 4) is an instance of the ORGANIZATION OBJECTIVE class. The incoming goal dependencies of the actor representing the system (Medi@ actor in Figure 4) from the actor not representing the organization (Media Supplier actor in Figure 4) correspond to system objectives, they are needs/requests to the system. Thus, *Find New User Needs* goal (Figure 4) is an instance of the SYSTEM OBJECTIVE class.

All the outcoming dependencies of the actor representing the system (Medi@ actor in Figure 4) correspond to external information, i.e. they are needs/requests from the system to the environment. Thus, *Internet Services* and *Process On-line Money Transactions* are instances of the EXTERNAL class.

After a means-ends analysis of the Medi@ actor, we define the Strategic Rationale (SR) model (Figure 5). Now, we introduce softgoal contributions to model sufficient/partial positive (respectively ++ and +) or negative (respectively -- and -) support to *Security*, *Availability*, *Adaptability*, *Attract New Customers* and *Increase Market Share* softgoals.

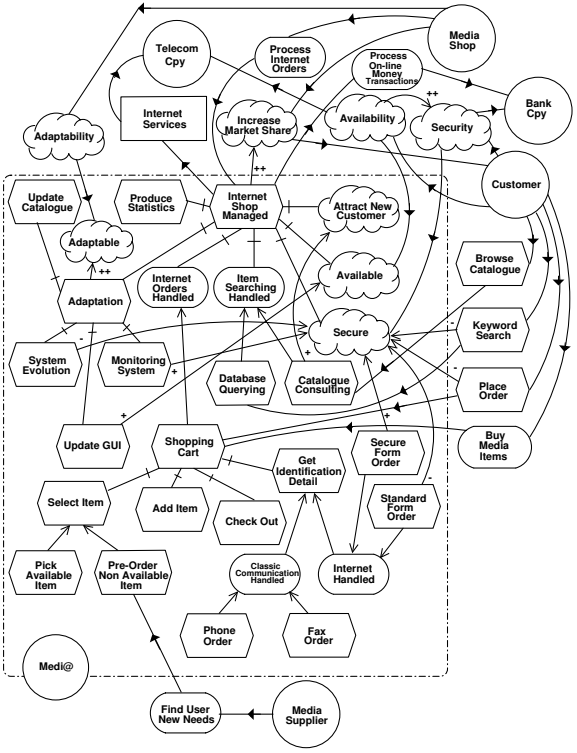

Figure 5. Strategic Rationale diagram for Medi@

Applying **Guideline IG2** of the proposed process, we find that all the goals resulting from means-ends analysis of the Medi@ actor (Figure5) correspond to system goals, i.e. they are the state of affairs the system aims to achieve through its functionalities. Thus, *Internet Orders Handled, Item Searching handled, Classic Communication Handled and Internet Handled* goals (Figure 5) are instances of the SYSTEM OBJECTIVES class. All softgoals resulting from means-ends analysis of the Medi@ actor (Figure 5) correspond to requirements, they are non-functional requirements that the system must to satisfy. Thus, *Adaptable, Attract New Customer, Available* and *Secure* softgoals (Figure 5) are instances of the REQUIREMENT class. All the tasks resulting from means-ends analysis of the Medi@ actor (Figure5) correspond to requirements, they are operations that the system should able to perform. Thus, *Update Catalogue, Produce Statistics, Internet Shop Managed, Database Querying, Catalogue*

Consulting, Secure Form Order, Standard Form Order, Get Identification Detail, Check Out, Add Item, Select Item, Adaptation, System Evolution, Monitoring System, Update GUI, Shopping Cart, Phone Order, Fax Order and Pre-Order Non Available Item tasks (Figure 5) are instances of the REQUIREMENT class. The details about the traceability of means-ends analysis of the system actor Medi@ (Figure 5) is described in [Pinto, Silva and Castro 2005].

Now we can design the proper system architecture aiming to meet the non-functional requirements previously defined. Hence, we present the requirement traceability process applied on the Tropos architectural phase in order to show how the design information and management decisions can be traced.

The software quality attributes (Availability, Security, and Adaptability), which we have highlighted in the SD diagram of the Late Requirements phase (Figure 6), will guide the selection process of the appropriate architectural style [Kolp, Giorgini and Mylopoulos 2002]. The Rational model (Figure 3) captures this information. It will be useful to justify the decision taken.

To cope with non-functional requirements (NFRs) (software quality attributes or softgoals) and select the architectural style for the system, we go through a means-ends analysis (see [Kolp, Giorgini and Mylopoulos 2002] for more details) using the NFR framework [Chung, Nixon, Yu and Mylopoulos 2000]. We refine the identified non-functional requirements to sub-requirements that are more precise and evaluate alternative organizational styles against them. The result is a Correlation Catalogue [Kolp, Giorgini and Mylopoulos 2002] (using the notation of NFR framework (++, +, --, -)) which demonstrates the suitability or not of certain architectural styles for the required NFRs.

Accordingly to the Correlation Catalogue (Table 1) provided in [Kolp, Giorgini and Mylopoulos 2002], the lines are the softgoals (i.e. non-functional requirements – NFR) to be fulfilled and the columns are some architectural organizational styles.

Table 1.Correlation Catalogue

	Structure-in-5	Pyramid	Joint Venture	Bidding
Predictability	+	++	+	--
Security	+	++	+	--
Adaptability		+	++	++
Cooperativity	+	++	+	-
Competitivity	-	-	-	++
Availability	+	+	++	--
Failability-Tolerance		--		++
Modularity	++	-	+	+
Aggregability	++		++	

For simplicity we assume that the architectural style selection was based only on the use of the softgoalsand the Correlation Catalogue. We can use the **Guideline IG3** to record the decision taken. Of course a richer set of information could equally be used/recorded to select the architectural style.

Applying **Guideline IG3** of the proposed process, we find that the issue on which a decision must be made in architectural design phase is the selection of the proper architectural style to be used. Thus, this selection process is an instance of the SUBJECT class in the Rational model (Figure3). The alternative solutions for the selection process are the architectural styles available in the literature. Hence, each architectural style is an instance of the POSITION class in the Rational model. The criteria used for deciding the proper solution is the non-functional requirements have been found in Late Requirements phase of Tropos. Thus, each non-functional requirement is an instance of the ARGUMENT class in the Rational model. Choosing the proper architectural style based on organizational approach and not based on traditional architectural styles is a fact taken into account in the selection process and therefore represents an instance of the ASSUMPTION class in the Rational model. The information taken as reference for choosing the proper organizational architectural style is the Correlation Catalogue which represents an instance of the DOCUMENT class in the Rational model. Now we can apply the architectural style Joint Venture in the system and find the architectural model depicted in Figure 6.

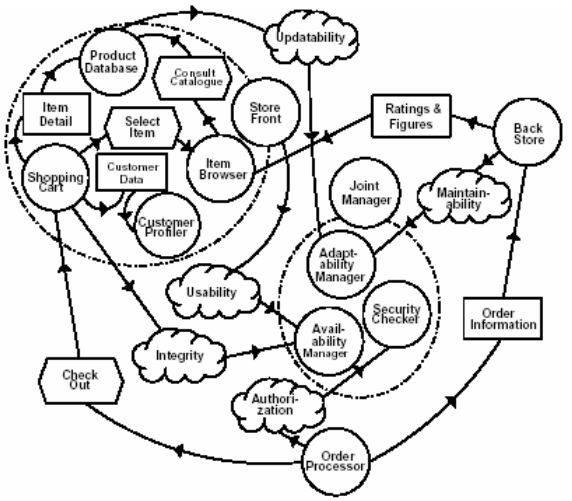

Figure 6. Architectural model for Medi@

Applying **Guideline IG4** of the proposed process, we conclude that each architectural components of Medi@ (Store Front, Back Store, Joint Manager and Order Processor) is an instance of the SUBSYSTEM class of the Design sub-model (Figure2).

5.2 Applying guidelines for Information Structuring (ST)

Having gathered all the relevant information, we can now structure it according to the second stage of the requirement traceability process. For simplicity we will not explicitly show the deletion or review some gathered instances performed according to **Guideline ST1**. Applying **Guideline ST2**, we can, for example, define an instance of the <resource> relationship between REQUIREMENT and ORGANIZATIONAL OBJECTIVES classes

in the Requirements Management model (Figure 1). The influence between organizational information and functional requirements can be evaluated as <H> (High), <M> (Medium) or <L> (Low).

We also can define an instance of the <resource> association between POSITION and ARGUMENT classes in the Rational model (Figure 3). The relationship used in the NFR framework (++, +, --, -) can be mapped to the <resource> association (Table 2), as <support, S>, <support, P>, <contradict, P>, <contradict, S>, respectively (read S as sufficient and P as partial).

Table 2. Instances of the association between positions and arguments

++ (*make)*	<support, S>
+ (*help)*	<support, P>
- (*hurt)*	<contradict, P>
-- (*break)*	<contradict, S>

Similarly we can define an instance of the <responsibility> association between REQUIREMENT and SUBSYSTEM classes in the Design sub-model (see Figure 2). The association between functional requirements and architectural components can be instantiated as <RB> (read RB as *Realized By*).

5.3 Applying guidelines for defining the Traceability Matrixes (TM)

Having structured all the gathered information, we can now create traceability matrixes according to the third stage of the requirement traceability process. Applying the **Guidelines TM1 and TM2**, we can, for example, create a traceability matrix between instances of the REQUIREMENT and ORGANIZATION classes (Table 3), of the Requirement Management sub-model (Figure 1).

Table 3. Traceability matrix between functional requirements and organizational information

<resource> →	[ORG1] Increase Market Share	[ORG2] Happy Customers	[ORG3] Continuous Supply	[ORG4] Media Items	[ORG5] Running Shop	[ORG6] Run Shop	[ORG7] Improve Service	[ORG8] Handle Billing	[ORG9] Handle Customer Orders	[ORG10] Order by Internet	[ORG11] Enhance Catalogue	[ORG12] Etc...
[RF1] Browse Catalogue		M			M	M	M					
[RF2] Keyword Search		M			M	M	M					
[RF3] Place Order		M				M	M					
[RF4] Process Internet Orders		H			H	H	H					
[RF5] Etc...	L	L			L	L	L					

By analyzing that matrix, we can conclude that the number of relationships between one requirement and all organizational information determine the most critical requirements. We can also conclude that the organizational information not related to the requirements in the traceability matrix can be disregarded. Moreover, if some organizational information is changed, the impact in the system requirements can be analyzed.

Table 4 presents a traceability matrix between instances of the POSITION and ARGUMENT classes of the Rational model (Figure 3), created according to the third stage of the requirement traceability process (section 3).

This matrix shows that the Joint Venture style fully supports, Availability and Adaptability, whereas the NFR Security is only addressed in a moderate fashion. By analyzing that matrix, we can conclude that non-functional requirements have a strong influence in the proper choice of the architectural style. Therefore, if some non-functional requirement is added or excluded in a newer version of the Medi@ system, it may impact in the fulfillment of the system's quality requirements hence, a new selection of an architectural style may be required in order to better satisfy the non-functional requirements. This kind of impact analysis is very important to be conducted.

Table 4. Traceability matrix between positions and arguments

<resource> →	[POS1]Pyramid	[POS2] Joint Venture	[POS3] Co-optation
[ARG1] Availably	<support, P>	<support, S>	<contradict, P>
[ARG2] Security	<support, P>	<support, P>	<contradict, P>
[ARG3] Adaptability	<support, P>	<support, S>	<support, S>

Table 5 presents a traceability matrix between instances of the REQUIREMENT and SUBSYSTEM classes of the Design sub-model (Figure 2).

Table 5. Traceability matrix between functional requirements and architectural components

<responsibility> →	[SUB1] Front Store	[SUB2] Back Store	[SUB3] Joint Manager	[SUB4] Order Processor
[RF1] Database Querying	RB			
[RF2] System Evolution			RB	
[RF3] Get Identification Detail				RB
[RF4] Add Item	RB			
[RF5] Etc…		RB		

Through that matrix, we can easily find the subsystem where a specific requirement is implemented. Thus, if some requirement is to be changed, the software engineers will spend less time looking for the subsystem(s) to be changed. They can query the traceability matrix, find the target subsystem(s) and perform the costs analysis of the required change before taking the decision for implementing it.

The SR model describes the intentional relationships that are "internal" to actors, such as means-ends relationship and task decomposition [Yu 1995]. By using traceability matrixes one can easily find these means-ands relationship and task decomposition. For details see [Pinto, Silva and Castro 2005].

In the sequel we present some related work as well as a comparison with our approach.

6 Related Work

Ramesh [Ramesh and Jarke 2001] introduces the reference model to trace requirements. This model enables the user to extract and adapt their elements to construct his/her own requirement model for a specific project. Our work considers the social aspects and includes new concepts such as task, and resource relationship which improves the semantic of the model. Gotel [Gotel 1996] presents one result of the empiric work related to the identification and understanding of the problems and practices associated with the requirements traceability. She divided the traditional requirement into pre-requirement and pos-requirement traceability. Our proposal presents a reference-model which aids of the identification, discussion and building of the traceability model. Besides this, it expresses explicitly the external and organizational aspects. Toranzo [Toranzo 2002] introduces one set of types relationship and structures the information that will be traceable in levels (external, organizational and managerial) to improve the semantic of requirement traceability. Our work goes beyond Toranzo's work to include a process to be followed during the development of the traceability model. Cysneiro [Cysneiro Filho, Zisman and Spanoudakis 2003] presents an approach that can be used to generate traceability relations between organizational models specified in i^* and software systems models represented in UML. Our work considers a reference model which supports building traceability matrixes for agent based system. Haumer [Haumer 2000] extends the type of Pre-requirements Traceability defined by Gotel [Gotel 1996] by recording concrete system usage scenarios using rich media (e.g. video, speech, graphic) and interrelating the recorded observations with the conceptual models. Besides the elicitation and validation phase, we consider the other phases of the software development lifecycle, e.g. late requirement and architectural design phases.

All the above works contributed to improve the requirements traceability in some aspects. Our proposal outlines a process to help the software engineer to find and follow the necessary information to be traceable in a specific project using the Tropos methodology. By using this process we can register the whole "history" of a requirement in an agent-oriented system, from the motivations for the requirement existence until its implementation and test routines.

7 Conclusions

It is well known that software traceability is a significant factor of efficient software project management and software systems quality [Castor, Pinto, Castro and Silva 2004]. In this work we outline a requirement traceability process to be used in the context of Tropos.

In this paper, we have proposed an approach which aims to support traceability through requirements specifications, system architecture models, static and dynamic software design models and implementation artifacts of agent-oriented software systems. Our purpose is to be able to perform an impact analysis before the implementation of a requested change. This is possible because the requirements impacted by the change can be detected since the links between these requirements and other system's artifacts, such as design and implementation ones, can be traced. Hence, software quality can be improved since we can check if all stakeholders' requirements are addressed by the system. Moreover, estimating change and effort become more accurate and consequently we can minimize the time and cost of software maintenance.

Nowadays, we are working on the traceability of the means-ends analysis used in the Tropos approach. Our requirement traceability process is still evolving and further guidelines to support both the detailed design and implementation phases need to be defined. Instantiating all the classes of the three reference models (Requirement Management and Design sub-models and Rational model) for each phase of Tropos is also required. To support this requirement traceability process, a proper CASE tool still needs to be developed.

8 References

Booch, G., Rumbauch, J. and Jacobson, I. (1999) *The Inified Modeling Language User Guide*. Addison-Wesley: Reading, MA.

Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J. and Perini, A. (2004) "Tropos: An Agent -Oriented Software Development Methodology"', in *Autonomous Agents and Multi –Agent Systems* 8 (3): 203-236, May 2004.

Castor, A., Pinto, R., Castro, J. and Silva, C. (2004) *"Towards Requirement Traceability in TROPOS"*, in Proc. of the VII Workshop on Requirements Engineering (WER'04), Tandil, Argentina.

Castro, J., Kolp, M. and Mylopoulos, J. (2002) "*Towards Requirements-Driven Information Systems Engineering: The Tropos Project"*. Information Systems Journal, Elsevier, Vol 27, pp. 365-89.

Chung, L. K., Nixon, B. A., Yu, E. and Mylopoulos, J. (2000) "*Non-Functional Requirements in Software Engineering"*, Kluwer Publishing.

Cysneiros Filho, G., Zisman, A. and Spanoudakis, G. (2003) *"Traceability Approach for I* and UML Models"*, in Proceedings of 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'03), Portland, May 2003.

Giorgini, P., Kolp, M., Mylopoulos, J. and Castro, J. (2005) Tropos: "A Requirements-Driven Methodology for Agent-Oriented Software". Book Chapter. In *Agent-Oriented Methodologies*. ed.: Idea Group, p. 20-45.

Gotel, O. (1996) "*Contribution Structures for Requirements Engineering"*. PhD Thesis. Department of Computing, Imperial College of Science, Technology, and Medicine, London, U.K.

Haumer, P. (2000) *A Framework to Improve Requirements Traceability.* Ph.D thesis, Informatik V. RWTH Auchen, Auchen, Germanyl, Octuber 2000.

Kolp, M., Giorgini, P. and Mylopoulos, J.(2002) "Information Systems Development through Social Structures", in *Proc. of the 14th Int. Conf. on Software Engineering and Knowledge Engineering* (SEKE'02), Ishia, Italy.

Pinheiro, F. A. C. (2003) "Requirements Traceability", Chapter of the *Book Perspectives On Software Requirements*. Kluwer Academic Publishers.

Pinto, R., Silva, C. and Castro, J. (2005) "A process for Requirement Traceability in Agent Oriented Development" in *Proceedings of WER 2005*. Workshop on Requirements Engineering, Porto, Portugal, pp 221-232

Ramesh, B. and Jarke, M. (2001) "*Towards Reference Models For Requirements Traceability"*. IEEE Transactions on Software Engineering, vol. 27, pp. 58-93, January 2001.

Toranzo, M. and Castro, J. (1999) "A Comprehensive Traceability Model to Support the Design of Interactive Systems", in *WISDOM´99 - International Workshop on Interactive system Development and Object Models*, 1999, Lisboa. 1999. Also described in NUNES, N., et all, Interactive System Design and Object Models In: International Workshop on Interactive System Development and Object Models, 1999, Lisboa. ECOOP'99 - Workshop Reader. London: Springer verlag - Lecture Notes in Computer Science LNCS, 1999. v.1743. p.267 – 287

Toranzo, M. (2002) "*A Framework to Improve Requirements Traceability"* (in Portuguese: Um Framework para Melhorar o Rastreamento de Requisitos). Ph.D thesis, Universidade Federal de Pernambuco, Centro de Informática, Brazil, December 2002.

Wooldridge, M. (1999) *Intelligent Agents*, in G. Weiss, editor. Multiagent Systems, the MIT Press, April 1999.

Wooldridge, M. J. (2002) *Introduction to Multiagent Systems*. John Wiley and Sons, New York, NY, USA.

Yu, E. (1995) "*Modelling Strategic Relationships for Process Reengineering"*. Ph.D thesis, University of Toronto, Department of Computer Science.

Zisman, A., Spanoudakis, G., Pérez-Miñana, E. and Krause, P. (2003) "Tracing Software Requirements Artefacts", in The *2003 International Conference on Software Engineering Research and Practice (SERP 2003)* in conjunction with The International Multiconference in Computer Science and Computer Engineering, Las Vegas, June 2003