A Process for Requirement Traceability in Agent Oriented Development

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. *Requirement traceability is intended to ensure continued alignment between stakeholders' requirements and various outputs of the system development process. Therefore a process for requirement traceability is a significant factor on efficient software project management. Failure to do so will imply in higher costs for maintaining software systems. Methodologies supporting requirement traceability can develop higher quality software with fewer costs. This paper presents an innovative research that aims to support traceability through requirements specifications, system architecture models, static and dynamic software design models and implementation artifacts of agent-oriented software systems. In this work we outline a process that can be used to extend Tropos to support traceability. An e-commerce example is used to demonstrate the applicability of the proposed approach.*

1 The Introduction

Researchers and stakeholders agree about the importance of the requirements traceability. In complex systems there are quite complex web of relationships, hence requirement tracing is inevitable [1]. In the last years, a large number of approaches and techniques to address various aspects of traceability have being developed for the software and systems engineering.

Requirement Traceability refers to the ability to describe and follow 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) [2]. Traceability improves quality of software system. It facilitates the verification and validation analysis, control of changes, as well as reuse of software systems components, and so on.

This paper presents an innovative research that aims to support traceability through requirements specifications, system architecture models, static and dynamic software design models and implementation artifacts of agent-oriented software systems. In particular we present a general framework [3] applied in the context of agent-oriented development [4]. We sketch the approach to enhance the *Tropos*[1] framework [5, 6, 7]

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

to support traceability. 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 [6, 7] spans four phases, as follows:

- *Early requirements*: concerned with the understanding of a problem by studying an organizational setting.
- *Late requirements*: where the system-to-be is described within its operational environment, along with relevant functions and qualities.
- *Architectural design*: where the system's global architecture is defined in terms of subsystems, interconnected through data, control and other dependencies.
- *Detailed design*: where the behavior of each architectural component is further refined.

The structure of this paper is follows: Section 2 presents the meta-models that are required to support traceability. In Section 3, we define a process that can be used to derive traceability information in the context of the Tropos approach. In Section 4, we apply Tropos to a case study and show all phases of the proposed requirement traceability process. Section 5 describes related work and finally Section 6 concludes the paper.

2 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 [9, 2].

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 [3].

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. The reference model used in our approach is based on Toranzo [3]. It defines classes that represent the information to be traced. These classes are related to each other by means of associations named *satisfy*, *resource*, *responsibility* and *represents*. The matricial representation of an association is a tuple whose structure varies according to the association type [3]. For instance, the association resource has two components (<DepDegree; Tree>). The first component, DepDegree, express the degree of dependency in a qualitative way (e.g. <H: High, M: Medium or L: Low> or <S: sufficient or P: partial>) or quantitative way (values between 1 and 10). The second component, Tree, represents the type of the logic tree which will relate the elements into a decomposition. This component can assume the values <A> (read A as AND) or <O> (read O as OR). The notation used to represent the proposed associations is based on UML (Unified Modeling Language) 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.

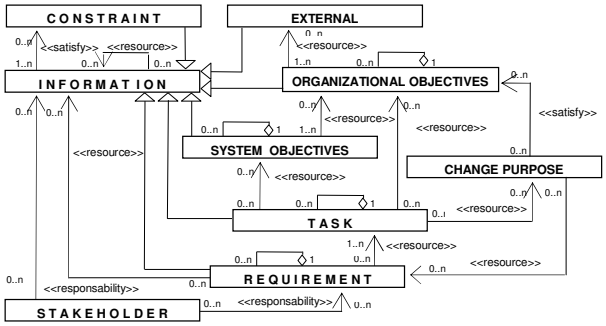

Figure 1. Requirements Management Sub-model

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

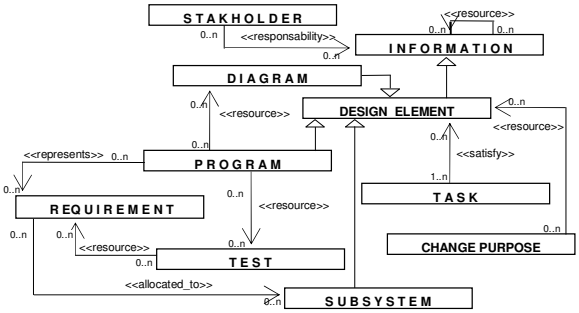

Figure 2. Design Sub-model

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

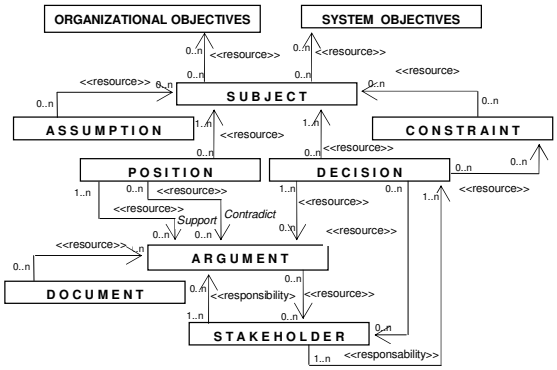

Figure 3. The Rational model

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.

3 The Requirements Traceability Process

The main contribution of this paper is to define a 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 to be traced.

2. Information Structuring (ST): consists of three activities. First, we remove the instances that represent irrelevant information, as well as delete the instances with the same meaning. Then, we determine the association among the instances, as well as their values.

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 phase, in this paper we do not apply the new requirement traceability process to this phase. It is expected that new guidelines are to be included when the Detailed Design phase is properly addressed. Now, we can introduce the traceability 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 which 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 which 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.

4 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.

On 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, the change impact assessment in the Medi@ system, as well as, to determine the requirements which satisfy a specific softgoal required for the Medi@ system. We could also trace the means-ends analysis used in the Tropos methodology.

4.1 Applying guidelines for Information Gathering (IG)

The description provided previously is sufficient for producing a model of an organizational environment. For details, see [7]. Having understood the organizational setting, one can now decide to develop a software system to support it (Figure 4). 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 see [7].

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.

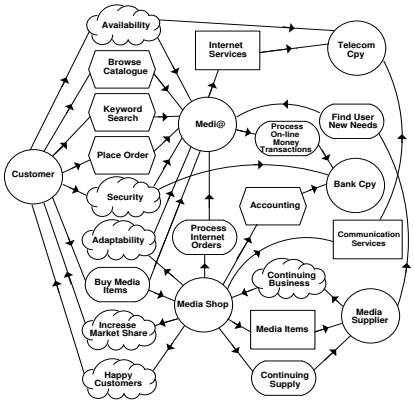

Figure 4. Strategic Dependency Diagram for Medi@ System

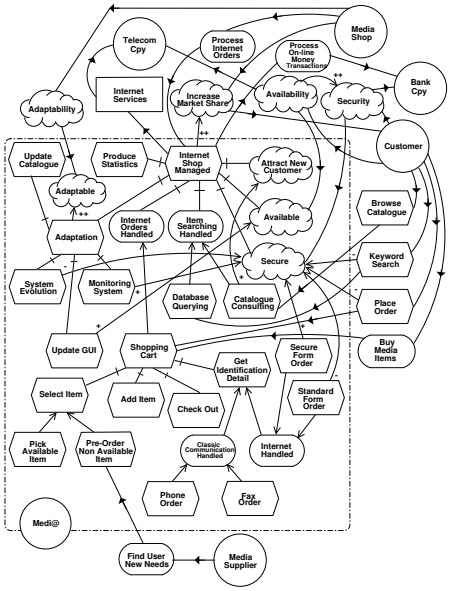

Figure 5. Strategic Rationale diagram for Medi@

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.

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.

Now we can design the proper system architecture aiming to meet the non-functional requirements previously defined. Hence, we can apply the requirement traceability process to the Tropos architectural design phase in order to show how the design information and management decisions can be traced. The Rational model (Figure 3) captures this information. However, the use of guideline IG3 is not shown in this work. An interested reader can find an example of its use in [4].

After selecting the proper architectural style, we can apply it to the system and find the system architectural model (See [5], for further details). For the sake of space, we will not apply the proposed process to perform traceability in architectural design phase. We could, for example, create a traceability matrix between REQUIREMENT and SUBSYSTEM instances.

Next section shows how the information gathered can be structured, as well as how the relationships between them can be instantiated.

4.2 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 revision of gathered instances performed according to **Guideline ST1**. For the sake of space, we could only show the creation of the traceability matrix capturing the mean-ends analysis of Medi@ system depicted in Figure 5.

In this paper, we highlight the traceability of means-ends analysis of the system actor Medi@ (Figure 5). Applying **Guideline ST2**, we can, for example, define the instances of the <<resource>> relationship between REQUIREMENT and SYSTEM OBJECTIVES classes and between REQUIREMENT and CHANGE PURPOSE classes in the Requirements Management model (Figure 1) and call them Decomposition_obj and Depend_on respectively. We can also define several instances of the <<resource>> association between instances of the REQUIREMENT class and call them Realized_by, Decomposition_req, Support_by and Contradict_by. One instance of the <<resource>> association between REQUIREMENTS and SYSTEM OBJECTIVES classes can be called Decomposition_obj is needed. Similarly an instance of the <<allocated_to>> association

between REQUIREMENT and SUBSYSTEM classes in the Design sub-model can be defined (see Figure 2).

Applying **Guideline ST3** to each instance defined in **ST2**, we can define the correspondent set of valid values. The influence between system objectives and functional requirements can be evaluated as <H> (High), <M> (Medium) or <L> (Low) and corresponds to the first component of the tuple which composes the association Decomposition_obj. The second component of the tuple is always <A> (read A as AND). The means-ends analysis could be mapped using Realized_by, Decomposition_obj, Decomposition_req, Support_by and Contradict_by relationship. For example, the task decomposition link can be instantiated as <A> (read A as AND), while the means-ends link can be instantiated as <O> (read O as OR) corresponding to the second component of the tuple which composes the association. The first component of the tuple is always H (High). The relationship used in the NFR framework (++, +, --, -) can be mapped in the following away, the positive and negative influence can be mapped to the Support_by and Contradict_by association respectively. The non-functional requirement is supported or contradicted in a sufficient (S) or partial (P) way which are the values of the first component of the association tuple. The second component is <A> (read A as AND) for all the instances.

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 to the instances of the <<resource>> association between REQUIREMENTS called Realized_by, Decomposition_req, Support_by and Contradict_by (Table 2, 3, 4, and 5). An other example is to create a traceability matrix between instances of the REQUIREMENT and SYSTEM OBJECTIVE classes, called Decomposition_obj (Table 6), of the Requirement Management sub-model (Figure 1).

The SR model describes the intentional relationships that are "internal" to actors, such as means-ends relationship and task decomposition [8]. The components that participated in the means-ends relationship and task decomposition are mapped as follows: the tasks and softgoals elements in the Decomposition_req matrix (Table 3) and the tasks and goals elements in the Decomposition_obj matrix (Table 6).

For example, Table 2 indicates that the placement of an order (RF03) requires the use (realized by) of a Shopping Cart (RF13). According to Table 3 the shopping cart involves (i.e. it is AND decomposed into) the selection of item (RF14), its addition to the cart (RF15), proceeding to check it out (RF16) and providing the relevant identification details (RF17). In turn, Table 6, indicates that there are two means (OR - alternatives) of getting the identification information (RF17), either using the internet (SO06) or using classic means such as sending a fax (SO05).

The matrixes are also useful to find which functional requirements are related some non-functional requirement. For this purpose, we use Support_by and Contradict_by matrixes. Table 4 shows that the security non-functional requirement (RN07) is partially supported by the use of monitoring systems (RF18) and secure forms (RF18). Unfortunately this security constraint may partially hinder System Evolution

(RF08). Moreover, the use of Standard Form is (RF19) is not adequate (RF19) as far as security is concerned. The matrixes could also be used to estimate the impact of a change. For example, suppose that the following request is made: the customer may wish to be informed of items which might be related to the ones he has ordered (see Table 7). After we find the initial requirements, we use the Realized_by link to follow the trace of the other requirements that could be influenced by this change. Hence we use the Decomposition_obj and Decomposition_req matrixes to find the requirements that are part of the decompositions created during the means-ends analysis.

Table 2. Realized_by Matrix

Realized_by <<resource>> ←	[RF01] Browse Catalogue	[RF03] Place Order	[RN02] Security
[RF12] Catalogue Consulting	<H,A>		
[RF13] Shopping Cart		<H,A>	
[RN07] Secure			<H,A>

Table 3. Decomposition_req Matrix

Decomposition_req <<resource>> ←	[RF13] Shopping Cart	[RF14] Select Item	[RF05] Internet Shop Managed
[RF13] Shopping Cart			
[RF14] Select Item	<H,A>		
[RF15] Add Item	<H,A>		
[RF16] Check Out	<H,A>		
[RF17] Get Identification Detail	<H,A>		
[RF20] Pick Available Item		<H,A>	
[RF21] Pre-order Non Available Item		<H,A>	
[RF06] Produce Statistics			<H,A>
[RF07] Adaptation			<H,A>

Table 4. Support_by Matrix

Support_by <<resource>> ←	[RN07] Secure
[RF09] Monitoring System	<P,A>
[RF18] Secure Form Order	<P,A>

Table 5. Contradict_by Matrix

Contradict_by <<resource>> ←	[RN07] Secure
[RF08] System Evolution	<P,A>
[RF19] Standard Form Order	<P,A>

Table 6. Decomposition_obj Matrix

Decomposition_obj <<resource>> ←	[RF05] Internet Shop Managed	[RF17] Get Identification Detail
[SO03] Internet Orders Handled	<H,A>	
[SO04] Item Searching Handled	<H,A>	
[SO05] Classic communication Handled		<H,O>
[SO06] Internet Handled		<H,O>

Table 7. Depend_on Matrix

Depend_of <<resource>> ←	**[CP01]** The system should suggest items which might be related to the ones the customer has ordered.
[RF01] Browse Catalogue	<H,A>
[RF03] Place Order	<H,A>

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

6 Related Work

Ramesh [9] 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 includes new concepts and relationships types, such as the task concept and the resource relationship, improving the semantic of the reference model. It also considers the rationale model. Gotel in [2] presents one result of the empiric work related with 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 explicitly addresses external and organizational aspects, as well as the system ones. Toranzo in [3] introduces a set of types of relationships and structure the traceable information in levels (external, organizational and management) to improve the semantic of requirement traceability. Our work extends Toranzo's work but includes a process to be followed during the development of the traceability model. Cysneiro [10] 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 systems. Haumer [11] extends the type of pre-requirements traceability defined by Gotel [2]. He, for instance, uses goal *attainment* and *failure* pre-traceability relations between goal-oriented requirement models and collections of observed cases of system usage encoded in multimedia (e.g., video and audio), in order to inform review activities which are concerned with the assessment of adequacy of these models, and he also provides method and tool support to use them in a reference base to support explanation, review, and negotiation of the conceptual models. Our proposal, beyond consider a requirements elicitation and validation phase, it encloses other phases of software development lifecycle, such as early requirements and architectural design phases.

All the works above contributed to improve the requirements traceability in some aspects. Our proposal outlines a process to help the software engineer to find and structure the necessary information to perform traceability 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

Requirement traceability has been recognized by many as an important pre-requisite for developing and maintaining high quality software. In this work, we outline a process that can be used to extend Tropos to address requirements traceability.

We intend to develop a complete and usable requirement traceability process for Tropos aiming to ensure the quality improvement of both the methodology and the software developed with it.

The benefits of requirements traceability are manifold: software quality can be improved since we can check if all stakeholders' requirements are addressed by the system. Similarly, an impact analysis can also be performed 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, estimating change and effort become more accurate and consequently we can minimize the time and cost of software maintenance.

Our requirement traceability process is still evolving and further guidelines for instantiating all the classes of the three reference models (Requirement Management and Design sub-models and Rational model) for each phase of Tropos may be required. In particular, we need to support both the detailed design and implementation phases. Proper tool supporting this requirement traceability process is also another topic that needs to be addressed.

8 References

1. Pinheiro, F. A. C. (2003) "Requirements Traceability", Chapter of the *Book Perspectives On Software Requirements*. Kluwer Academic Publishers.
2. Gotel, O. (1996) "*Contribution Structures for Requirements Engineering"*. PhD Thesis. Department of Computing, Imperial College of Science, Technology, and Medicine, London, U.K.3. 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
3. 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.Ramesh, B. and Jarke, M. (2001) "*Towards Reference Models For Requirements Traceability"*. IEEE Transactions on Software Engineering, vol. 27, pp. 58-93, January 2001.
4. 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.
5. 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.
6. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A. (2004) "Tropos: An Agent - Oriented Software Development Methodology"', in *Autonomous Agents and Multi –Agent Systems* 8 (3): 203-236, May 2004.
7. 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.
8. Yu, E. (1995) "*Modelling Strategic Relationships for Process Reengineering*". Ph.D thesis, University of Toronto, Department of Computer Science.
9. Ramesh, B. and Jarke, M. (2001) "*Towards Reference Models For Requirements Traceability"*. IEEE Transactions on Software Engineering, vol. 27, pp. 5-93, January 2001.
10. 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.
11. Haumer, P. (2000) *A Framework to Improve Requirements Traceability.* Ph.D thesis, Informatik V. RWTH Auchen, Auchen, Germanyl, Octuber 2000.