

*Faculty of Informatics*

*Faculty of Informatics - Papers*

---

*University of Wollongong*

*Year 2006*

---

Co-evolution of Agent Oriented  
Conceptual Models and Use Case  
Diagrams

M. M. Bhuiyan\*

M. M. Islam†

A. Krishna‡

A. Ghose\*\*

\*University of Wollongong

†University of Wollongong

‡University of Wollongong, aneesh@uow.edu.au

\*\*University of Wollongong, aditya@uow.edu.au

This paper was originally published as: Bhuiyan, MMR, Islam, MMZ, Krishna, A & Ghose, A, Co-evolution of Agent Oriented Conceptual Models and Use Case Diagrams, Sixth International Conference on Quality Software (QSIC 2006), Beijing, China, October 2006, 446-451. Copyright IEEE 2006.

This paper is posted at Research Online.

<http://ro.uow.edu.au/infopapers/492>

# Co-evolution of Agent Oriented Conceptual Models and Use Case Diagrams

Mohammad M R Bhuiyan, M.M.Zahidul Islam, Aneesh Krishna, Aditya Ghose  
Decision Systems Laboratory  
School of Information Technology and Computer Science (SITACS)  
University of Wollongong (UOW), Northfields Avenue, NSW 2522, Australia  
{mmrb95, mmzi44, aneesh, aditya}@uow.edu.au

## Abstract

*Agent-oriented conceptual modeling notations such as  $i^*$  represents an interesting approach for modeling early phase requirements which includes organizational contexts, stakeholder intentions and rationale. On the other hand Use Case diagram is used for capturing functional requirements of the system. The integration of  $i^*$  model and Use Case diagram closes the gap of capturing organizational requirements and system requirements. But in both contexts the requirements might change at any time. Any change made in one model must be reflected in the other. This paper proposes a methodology supporting the co-evolution of these two otherwise disparate approaches in a synergistic fashion.*

## 1. Introduction

Constructing a system that adhere to organizational environment and meet end users need require developing a clearly defined early stage functional requirements (such as determining the main goals of the intended system, relations and dependencies among stakeholders, alternatives in the early-stage requirements analysis etc.) [5]. The  $i^*$  modeling [5] framework is a semi-formal notation built on agent-oriented conceptual modeling is well-suited for answering these questions.

A number of proposals have been made for combining  $i^*$  modelling with late-phase requirements analysis and the downstream stages of the software life-cycle. The TROPOS project [1] uses the  $i^*$  notation to represent early- and late-phase requirements, architectures and detailed designs. However, the  $i^*$  notation in itself is not expressive enough to represent late-phase requirements, architectures and designs. To address this problem, a custom-designed formal language called FormalTropos [2] has been proposed. Proposals to integrate  $i^*$  with formal agent programming languages have also been reported in the literature [4]. This paper has similar objectives, but takes a somewhat different approach. We believe that the value of conceptual modeling in the  $i^*$  framework lies in its use as a notation complementary to existing specification languages, i.e., the expressive power of  $i^*$  complements that of existing notations. The use of  $i^*$  in this fashion requires that we define methodologies that support the co-evolution of  $i^*$  models with more traditional

specifications. We use the notion of co-evolution in a very specific sense to describe a class of methodologies that permit  $i^*$  modeling to proceed independently of specification in a distinct notation, while maintaining some modicum of loose coupling via consistency constraints. In the current instance, we examine how this might be done with formal Unified Modeling Languages (UML) [3]. Our aim, then, is to support the modeling of organizational contexts, intentions and rationale in  $i^*$ , while traditional specifications of functionality and design proceeds in the formal Use Case [3] notation of UML. More generally, this research suggests how diagrammatic notations for modeling early-phase requirements, organization contexts and rationale can be used in a complementary manner with more traditional specification notations like UML.

In Section 2 & 3, below, we present  $i^*$  modeling framework and UML Use Case diagrams with an example. Section 4 discusses about some benefits of the co-evolution of the two notations. Section 5 introduces the mapping methodology between  $i^*$  models and Use Case diagrams based on [15, 16]. Section 6 discusses a methodology for supporting the co-evolution of  $i^*$  models and Use Case diagrams. Finally, Section 6 presents some concluding remarks.

## 2. The $i^*$ Modeling Framework:

The central concept in  $i^*$  is that of intentional actor. Intentional properties of an agent such as goals, beliefs, abilities and commitments are used in modelling requirements [5] [6]. The actor or agent construct is used to identify the intentional characteristics represented as dependencies involving goals to be achieved, tasks to be performed, resources to be furnished or softgoals (optimization objectives or preferences) to be satisfied. The  $i^*$  framework also supports the modelling of rationale by representing key internal intentional characteristics of actors/agents. The  $i^*$  framework consists of two modelling components [5]: Strategic Dependency (SD) Models and Strategic Rationale (SR) Models.

The SD and SR models are graphical representations that describe the world in a manner closer to the users perceptions. The SD model consists of a set of nodes and links. Each node represents an "actor", and each link between the two actors indicates that one actor depends on the other for something in order that the former may attain

some goal. The depending actor is known as *dependor*, while the actor depended upon is known as the *dependee*. The object around which the dependency relationship centres is called the *dependum*. The SD model represents the goals, task, resource, and softgoal dependencies between actors/agents. In a goal-dependency, the *dependor* depends on the *dependee* to bring about a certain state in the world. In a task-dependency, the *dependor* depends on the *dependee* to carry out an activity. In a resource-dependency, one actor depends on the other for the availability of a resource. In a softgoal-dependency, a *dependor* depends on the *dependee* to perform certain goals or task that would enhance the performance. The notion of a softgoal derives from the Non-Functional Requirements (NFR) framework [7] and is commonly used to represent optimization objectives, preferences or specifications of desirable (but not necessarily essential) states of affairs.

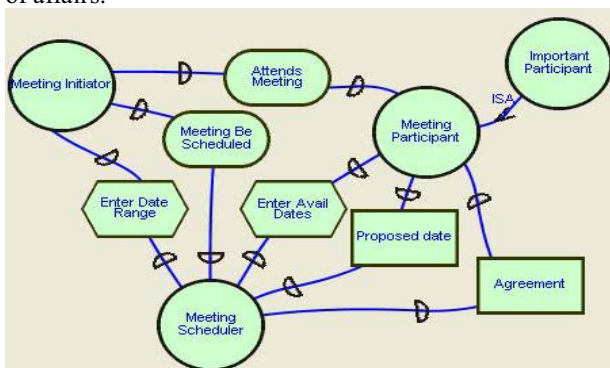


Figure 1: SD model of the Meeting Scheduling System.

We consider a Meeting Scheduling System for our methodology explanation. The *i\** model in figure-1, 2, involves a MeetingInitiator actor which depends on MeetingParticipant actor to achieve AttendMeeting goal.

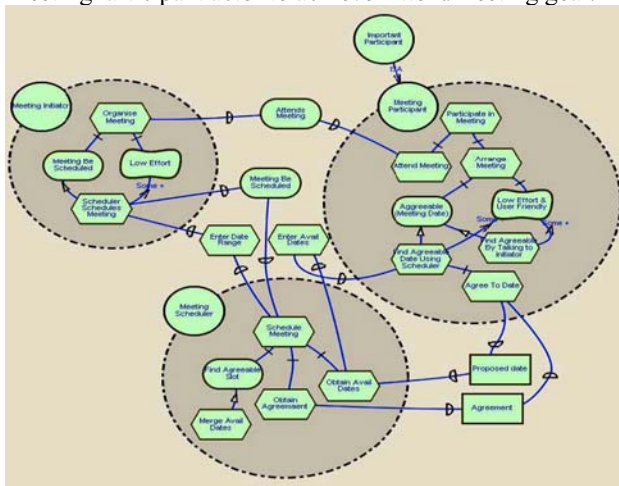


Figure 2: SR model of the Meeting Scheduling System.

An SR model supplies more detailed information of an actor's inside model such as goals, tasks, resources and

softgoals which emerge as both internal and external dependencies in a SR model. As a result of this, an SR model provides resources for modeling stakeholder interests and how they might be fulfilled.

### 3. Use Case Diagram:

A Use Case specifies the behavior of the system or a part of the system. It is a description of a set of sequence of actions, including the variants that a system performs to generate a visible result of value to an actor. Use Case diagrams are central to modeling the behavior of the system or a sub system. Each one of these shows represents a set of Use Cases, actors and their relationship [3]. This scenario based technique has become very popular to understand, model and validate user and system requirements [10] [11]. The Use Case diagram describes the use of a system by the actors related to it. These actors are any external elements that interact with the system. The interactions between the system and various actors provide a way for the developers to come to a common understanding with the systems' end users and domain experts [3]. Use Cases also help to validate the proposed system architecture and to verify the system as it evolves during development.

### 4. Benefits of the Co-evolution of *i\** and Use Case models:

Designing and constructing a good quality system that produces high efficiency in operation and manages the organization's requirements through its activity needs significant effort in requirement engineering and precise mapping of the two complementary notations. Collecting organizational requirements through *i\** framework produces rich information for the system/software to be constructed. The usefulness and effectiveness of *i\** can be enhanced by using it with an industry standard specification language such as Unified Modeling Language (UML). Our vision is that the Use Case notation and the *i\** modeling framework can function in a complementary and synergistic fashion.

- There is a need to map both SD and SR models into late phase requirements specifications; UML can be used successfully to realize these goals.
- We cannot represent softgoals in UML. On the other hand, the *i\** notation allows us to represent and reason with softgoals (representations of nonfunctional requirements or objectives).
- Co-evolution of *i\** and Use Case models allows observation and assessment of the impact of changes into the functional and non functional requirements of the future system.
- The Use Case is written from the actor's point of view, not from the system's point of view. Use Cases

derived from *i\** actor dependencies can be defined clearly in the Use Case diagrams. The co-evolution of these two models helps analysts to identify and understand important Use Cases for the planned system which allows them to avoid too many Use Cases descriptions.

### 5. Derivation of Use Case Model from Organizational Modeling:

We can derive a Use Case model from an *i\** diagram following the guidelines proposed by Victor F. A. Santander and Jaelson F. B. Castro [8] [9]. Some steps of the guidelines will be discussed briefly as we start mapping the SD and SR model of the Meeting Scheduling System to Use Cases diagrams. The following steps will generate a Use Cases diagram for the Meeting Scheduling System.

#### Discovering System Actors:

The first step for the mapping includes discovering appropriate actors from the SD model. The actors in figure 1 are: Meeting Initiator (MI), Meeting Scheduler (MS), Meeting Participant (MP) and Important Participant (IP). According to guideline 2 [9], MS actor can not be taken as a Use Case actor. The other actors, MI, MP, and IP are considered as candidate actors for the Use Case Diagram, as they will interact with the intended Meeting Scheduling System. The IP actor is related to MP actor by an IS-A relationship. According to guideline 4, IP actor will be mapped individually for actors in Use Cases and will be related in the diagram through a <<generalized>> relationship. IP is therefore considered as a specialization of MP actor.

#### Discovering Use Cases for the Actors:

Guideline 5 [9] suggests that for each candidate actor, we should observe all its dependencies in which the actor is a *dependee*, for discovering the Use Case of the System.

**Table-1: Use Case Discovery**

Actor	Dependency	Type of Dependency
MI	EnterDateRange	Task
MP	AttendsMeeting	Goal
MP	EnterAvailableDates	Task
MP	Agreement	Resource

Now we will look for the special situations where the system itself is a *dependee*. The goal dependency, MeetingBeScheduled between MI and the system, requires some interaction. This dependency represents the use of the system by the MI actor. So MeetingBeScheduled is considered as a Use Case that describes the details of the scheduling process. In this case the *dependor* itself is the Use Case actor.

#### Discovering and Describing Use Case Scenario:

In this step SR model of the Meeting Scheduling system is used as source of information for the scenario description and Use Cases relationships.

**Table-2: Use Case Goal Classification**

Actor	Use Case Goal	Goal Classification
MI	EnterDateRange	Subfunction
MI	MeetingBeScheduled	Summary
MP	AttendsMeeting	User Goal
MP	EnterAvailableDates	Subfunction
MP	Agreement	Subfunction

The Use Case MeetingBeScheduled is classified as summary goal that contains all the necessary steps to schedule the meeting. Scenarios for the Use Cases are also derived from the SR model. The scenario for this Use Case is described below:

**Table-3: Use Case Scenario Discovery**

Use Case: **MeetingBeScheduled**

**Actor:** MI

**Goal:** Schedule to Meeting

**Scenarios:**

1. The MI actor initiates the Use Case by supplying a date range for the meeting. So the EnterDateRange Use Case is included <<include>> in this step.
2. Based on the proposed dates by the MI the system then asks the MP to provide their available dates. For this reason the Use Case EnterAvailableDates has also been included <<include>>.
3. The system then look for a consensus date list from the proposed dates of the MI and MP.
4. Based on this list, the system proposes a date for the meeting to be scheduled.
5. The system then request agreement for a scheduled meeting date. At this stage Agreement Use Case is included <<include>>.

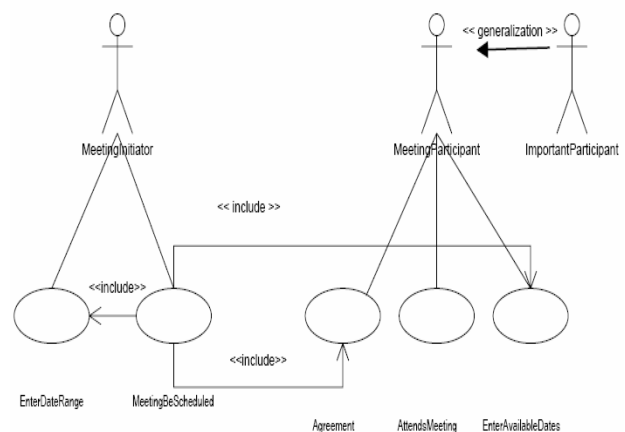


Figure 3: Use Case diagram derived from the SD and SR models.

Although Castro, J. [8] [9] has proposed this methodology based on goal-oriented analysis, it does not reflect the softgoals of the *i\** model in corresponding Use Case diagram. Our methodology for co-evolution will also add a

new feature to include these non functional requirements of the system.

## 6. Methodology Supporting the Co-evolution of $i^*$ and Use Case Model:

In this section we will propose a methodology for the co-evolution of  $i^*$  and Use Case model. Our strategy is to localize the changes. This co-evolution process involves two aspects; One is reflecting changes of  $i^*$  model on Use Case diagram and the other is reflecting changes of Use Case diagram on  $i^*$  model

There are sixteen categories of changes that may occur to an  $i^*$  model [20, 21, 22]. These are addition and deletion, respectively, of the following eight elements: Dependencies, Tasks, Goals, Resources, Softgoals, Means-end links, Task-decomposition links and Actors [12]. In our methodology we will focus all of them. Our illustration will show the possible changes that may occur in the the  $i^*$  model of the Meeting Scheduling system (figure-2, 4), and how these changes affect the Use Case diagram (figure-3, 5).

### Guideline-1: Addition/ deletion of an actor to an existing $i^*$ diagram:

Adding an Actor to the  $i^*$  model can make two possible changes to the Use Case scenario:

1. It may introduce a new Use Case actor and
2. Creates dependencies among actors, which in turn creates new Use Case.

With the addition of a new actor the SD and SR model is extended. The SR model is further decomposed to outline the goals, tasks, resources and soft goals of the new  $i^*$  actor and also their interactions to the system or with other actors. In this case dependencies among the actors need to be identified, and guideline-2 needs to be followed to reflect the changes in the Use Case model.

If the new actor in  $i^*$ , is related through the IS-A mechanisms and mapped individually for actors in Use Cases, it will be related in the Use Case diagrams through the <<generalization>> relationship. If an actor of the  $i^*$  model is deleted, goals, tasks, resources and soft goal dependencies related to the actor will be removed from both the SD and SR model. That means its association and interactions with any other actors will be removed. In this case, the Use Case actor will be removed and the scenario will be updated to reflect the change.

For example, if we add a new actor Meeting Place Coordinator (MPC) (figure-4), it will introduce a new Use Case actor. It will create dependencies as well which in turn introduce new Use Cases. At this stage, we will add a new actor in the Use Case Discovery table and follow guideline-2 for other related changes. From figure-1 we can observe that the actor IP is related to MP with an IS-A relationship. IP actor is thus represented in the Use Case with <<generalization>> relationship. Suppose MI is

deleted from the  $i^*$  model. Then all the dependencies associated to this actor i.e. AttendsMeeting, EnterAvailableDates will be removed, Agreement Use Cases will be removed from the Use Case diagram. Use Case Scenario Discovery table needs to be updated as well. In this case scenario-4 of the Use Case Scenario Discovery table will be removed.

### Guideline-2: Addition/ deletion of a dependency to an existing SD model:

Addition of a dependency may lead to the creation of new Use Case depending on the type of dependency, *depender* actor and *dependee* actor. For addition of goal/ task/ resource dependency we should consider two situations. Firstly, if none of the actors involved in the new goal/ task/ resource dependency is system actor, then we should observe which actor is the *dependee* actor. The dependency will be allocated to the *dependee* actor (Use Case Discovery table) and a new Use Case will be created for this actor. Secondly, if the system actor itself acts as a *dependee*, then special situation should be considered. In this case the interaction of the actors needs to be monitored. If these interactions directly relates to the operation of the system a new Use Case will be introduced. But in this situation the *depender* actor will be the Use Case actor.

If a dependency is deleted from an  $i^*$  model, then all the task, goals, resource and softgoal associated with it and corresponding Use Case are also removed. If it does not have a corresponding Use Case, the modified SR model need to be checked to find what impact or changes it is making to the existing Use Case scenario.

For example, the addition of the new actor MPC has introduced two dependencies, EnterMeetingDate task dependency and ConfirmMeetingLocation resource dependency with the MS actor (figure-4). From observation we can see that MPC is the *dependee* actor for ConfirmMeetingLocation dependency. So it will be added in the Use Case Discovery Table and a new Use Case ConfirmMeetingLocation will be introduced. We should consider a special situation for the task dependency EnterMeetingDate between MPC and MS. In this case the system itself is the *dependee* actor. From observations we can conclude that this task dependency requires some interaction between MPC and MS actors which represents EnterMeetingDate as a new Use Case. But in this situation the *depender* actor MPC is the Use Case actor.

If any dependency is removed from the actors, then all the interactions associated with it will also be removed. Both Use Case Discovery and Use Case Scenario Discovery tables need to be updated to reflect the changes.

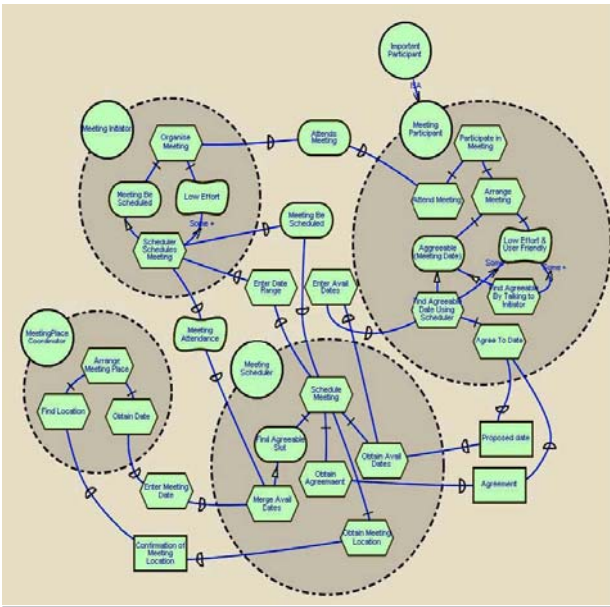


Figure 4: SR model of the Meeting Scheduling System with an illustration of possible changes.

**Guideline-3: Addition/ deletion of a task to an existing SR model:**

Addition of a new task creates a new dependency between actors directly or via links. Generally a new task in the existing SR model is related to a goal via means-ends link or to an existing soft goal via contribution link. It may also necessarily be associated with other tasks or resources via task decomposition link. If the added new task delineates a dependency then it will be added in the Dependency column of the Use Case Discovery table. This dependency can generate a new Use Case depending on its interaction behavior with the other actors. On the other hand if the goals, soft goals, tasks and resources associated with the new task contribute directly or through link to actor/system goal, needs to be included as <<include>> to the Actor's Goal Use Case from the derived Use Case of the new task.

For example, adding a new task ArrangeMeetingPlace in the SR model for MPC actor (figure-4) introduce a dependency. In this case, we should follow guideline-2.

**Guideline-4: Addition/ deletion of a goal/ resource/ to an existing SR model:**

Generally *i\** goal can be recorded as a Use Case goal. If the new goal to the SR model creates a new dependency it may create a new Use Case. If a new resource to the SR model produce a dependency between actors and if it creates a direct or via link which elicits a goal to be achieved by the resource receiving actor, it will essentially be a Use Case of the system.

Deletion of a goal/resource removes the links associated with it. This modification needs to be adjusted in the

Discovering Use Case actors table and it will lead to a modification in the Use Case scenario describing table.

Guideline-1 is to be followed if a new dependency is introduced by the addition of a goal/ resource/ task to an existing SR model. If addition or deletion of a goal/ resource to the SR model in figure-2, 4 results some new dependencies, guideline-2 and 4 should be followed. In case of softgoal we should associate it with the non-functional requirements of Use Cases.

**Guideline-5: Addition/ deletion of a task decomposition link to an existing SR model:**

Task decomposition link describes the breakdown of acts that an actor holds into it in the SR model. It can essentially be illustrated as a collection of sub-tasks, sub-goals, resources etc. Addition of a task decomposition link generates these collections under the parent which they are linked to through the decomposition link. Removing a task decomposition link deletes those collections of decomposed acts where they are singly linked directly or via linked to that removing decomposition link. In both addition and deletion of decomposition links Use Case Scenario needs to be modified.

For example, if we add two tasks ObtainDate and Find-Location by the task decomposition link in the SR model of MPC actor, it will generate two dependencies, EnterMeetingDate and ConfirmLocation. These dependencies introduce two Use Cases. By following guideline-2 we can make the necessary changes to our Use Cases

**Guideline-6: Addition/ deletion of a softgoal/ softgoal dependency to an existing SR model:**

Before we go to the discussion of changes in softgoals in *i\** models, we propose an extension of [8] [9]. We view softgoals as optimization goals where there is no way of actually specifying whether the softgoal was achieved completely. But, softgoals have a positive or negative contribution for achieving, accomplishing a goal, task, resource [5]. So when mapping from *i\** model to Use Case Diagram it is necessary to reflect the softgoals directly. It can be mapped as a non-functional requirement associated with a specific Use Case. We propose that the softgoal/ softgoal dependency will have a new Use Case that will be connected with the original Use Case by <<extends>> relationship. This ensures that the Use Case can contribute to "satisfice" the non functional requirement.

For example the MeetingAttendance softgoal dependency in figure-4 can be mapped as MeetingAttendance Use Case which will have an <<extends>> relationship with the original Use Case MeetingBeScheduled. Softgoal in SR model is mapped similarly. In the MeetingParticipant actor of *i\** diagram the functional goal Agreeable-MeetingDate is satisfied by participants through FindAgreeableDateUsingScheduler resource. But all participants may not find this approach of the system convenient

always. In order to make the system more convenient there is LowEffort&UserFriendly softgoal that will find dates through the FindAgreeableByTalkingToInitiator resource so that the participants have an alternative way to find a date for the meeting. In this case, LowEffort&UserFriendly Use Case will be created and extended under the MeetingParticipant actor.

In case of addition/ deletion of softgoal/ softgoal dependency in the *i\** model the corresponding Use Case Diagram will be changed as there will be creation/ deletion of new Use Cases.

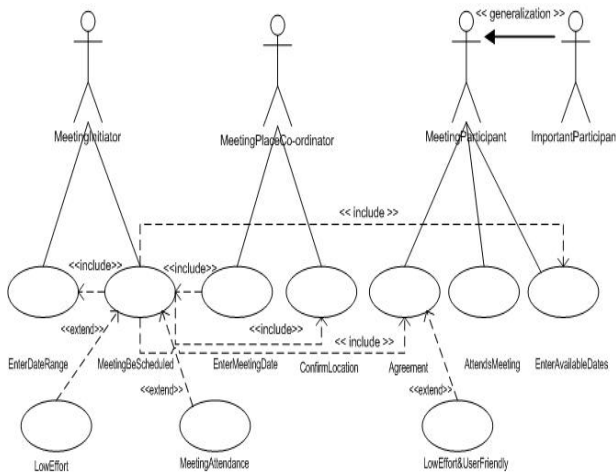


Figure 5: Use Case Diagram reflecting the changes in the *i\** diagram.

## 7. Conclusion and Further Work:

In this paper we have presented a relatively simple methodology to support the co-evolution of an early-phase requirements modeling notation *i\** with the Use Case notation of UML. We can analyze the system behavior using real life examples which is otherwise not possible by only looking at the *i\** model and Use Case model separately.

When proposing the co-evolution of two otherwise disparate approaches for requirements engineering, we need to maintain consistency between the two approaches. The mapping rules [6] [7] can be viewed as providing informal semantics to the *i\** diagrams. We believe that these semantics are largely consistent with the somewhat implicit semantics for *i\**. The proposed set of guidelines in our paper constrains the analyst to map the elements of the *i\** model to appropriate Use Case models and ensures that the two models are consistent. This allows us to trace corresponding elements in the two models when changes are made.

We note that some reverse procedures can be followed to reflect the changes of Use Case diagram on *i\** model given some assumptions that i) The Use Case diagrams

were obtained from an initial *i\** model via mapping following the guidelines described above. ii) The prior *i\** model is available for reference. Given these assumptions it is simple to identify the changes in Use Case diagram and thus reconstruct the corresponding *i\** model without loss of information.

We have not however investigated the possibility of articulating semantic consistency constraints between *i\** models and Use Case models. We have not focused on the reflection of changes of Use Case diagram on *i\** model. These two issues will be discussed in our future work.

## Reference:

- [1] J. Castro, M. Kolp and J. Mylopoulos. Towards Requirements-Driven Information Systems Engineering: The Tropos Project. Information Systems, Elsevier, Amsterdam, The Netherlands, 2002.
- [2] Fuxman, A., Pistore, M., Mylopoulos, J., Traverso, P. Model checking early requirements specifications in Tropos. *Proceedings of Fifth IEEE International Symposium on RE*, Toronto, Canada, August 27-31, 2001, pp. 174-181.
- [3] G. Booch, I. Jacobson, and J. Rumbaugh, The Unified Modeling Language User Guide, Addison-Wesley, 1999.
- [4] Wang, X., Lesprance, Y. Agent-Oriented Requirements Engineering Using ConGolog and *i\**, *AOIS-2001*, Berlin, Germany, 2001, pp. 59-78.
- [5] Yu, E. Modeling Strategic Relationships for Process Reengineering. PhD Thesis, Graduate Department of Computer Science, University of Toronto, Toronto, Canada, 1995, pp. 124
- [6] Yu, E. Agent Orientation as a Modelling Paradigm, *Wirtschaftsinformatik*, 2001, Vol. 43, No. 2, pp. 123-132.
- [7] Chung, L. Representing and Using Non - Functional Requirements for Information System Development. A Process-Oriented Approach. PhD Thesis, Graduate Department of Computer Science, Toronto, University of Toronto, 1993.
- [8] Santander, V. F. A., Castro, J. F. B. Deriving Use Cases From Organizational Modeling In: *IEEE Joint Conference On Requirements Engineering - Re02*, 2002, Essen, Germany
- [9] Santander, V. F. A., Castro, J. F. B. Deriving Use Cases From Organizational Modeling. Los Alamitos, California, USA: IEEE, 2002. V.1. P.32 - 39
- [10] C. Rolland, C. Souveyet, and C.B. Achour, "Guiding Goal Modeling Using Scenarios", *IEEE Transactions on Software Engineering*, Vol 24, No 12, Special Issue on Scenario Management, December 1998.
- [11] I. Jacobson, *Object Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, 1995.
- [12] A. Krishna, A. Ghose, and S. Vilkomir. Co-Evolution of Complementary Formal and Informal Requirements, *IWPSE'04*, September 06 - 07, 2004, Kyoto, Japan, pp. 159-164.