Integration of Organisational Models and UML Use Case Diagrams

Moshiur Bhuiyan[1*], Farzana Haque[1], Luba Shabnam[2]

[1] Service Consulting, Enterprise Cloud Systems Pty Limited, Sydney, NSW 2560, Australia.
[2] School of Chemical and Biomolecular Engineering, University of Sydney, Sydney, NSW 2006, Australia.

* Corresponding author. Tel.: (0061) 425217587; email: moshiurb@ecloudsys.com
Manuscript submitted April, 4, 2016; accepted December 28, 2016.
doi: 10.17706/jcp.13.1.1-17

Abstract: The fundamental step in systems implementation is requirements gathering. This step is categorized into early phase requirements elicitation and systems modelling. Requirement elicitation involves gathering stakeholders' intentions, goals, rationale and organizational context that can be modelled with agent-oriented conceptual modelling such as *i**. On the other hand, systems' modelling is portraying abstract version of interaction among inter-related components within a whole system via UML languages such as Use Case diagrams. The co-evolution of Agent-Oriented conceptual model such as *i** and Use Case Diagrams aims to close the gap of capturing organizational requirements and system requirements. However, changes made in *i** must be reflected in the Use Case diagram to ensure consistency in requirements. Therefore, this paper proposes a methodology supporting the co-evolution of these two otherwise disparate approaches in a synergistic fashion.

Key words: Agent-oriented conceptual modelling, organisational modeling, use case, *i** framework, SD model, SR model, actor.

1. Introduction

Software development and system re-engineering requires developing 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.) [1], [2]. The *i** modelling framework is a semi-formal notation built on agent-oriented conceptual modelling (AOM) that is used to analyse organizational goals by capturing socio-technical scenarios [3].

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 and has been applied to many areas such as risk management, information security, enterprise architecture and business process management [2]. The TROPOS project [4] uses the *i** notation to represent early- and late-phase requirements, architectures and detailed designs. However, the disadvantages of *i** modelling language are the challenges of portraying the level of abstraction, goals may be shared among actors, hence it may further complicate the mapping process [5]. Adding to that, the *i** notation in itself is not expressive enough to represent late-phase requirements, architectures and designs.

To address this issue, several custom-designed formal languages were introduced including Tropos, Formal Tropos, Prometheus, AOR, BPM and Gaia [6], [7] has been proposed. Proposals to integrate *i** with formal agent programming languages have also been reported in the literature [2], [8]. This paper has similar

objectives, but takes a somewhat different approach. We believe that the value of conceptual modelling 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** modelling 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 Modelling Languages (UML) specifically through Use Cases [9]. Use cases describes the interaction among uses and systems in natural language [10]. Our aim, then, is to support the modelling of organizational contexts, intentions and rationale in *i**, while traditional specifications of functionality and design proceeds in the formal Use Case [9] notation of UML. Our proposed method can also ensure that documented requirements are coherent, consistent and reviewable throughout the software development cycle [7], [11]. More generally, this re-search 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, we present formal basis for co-evolution of models. In Sections 3 & 4, below, we present *i** modeling framework and UML Use Case diagrams with examples. Section 5 discusses some benefits of the co-evolution of the two notations. Section 6 introduces the mapping methodology between *i** models and Use Case diagrams based on [12], [13]. Section 7 discusses a methodology for supporting the co-evolution of *i** models and Use Case diagrams. Finally, Section 8 presents some concluding remarks.

2. A Formal Basis for Co-evolution of Models

A key concern of this research is the definition of methodologies that permit models in distinct notations (ideally with complementary representational capabilities) to co-evolve. Informally, models in distinct notations are said to co-evolve if the following are true:

- These models are independently maintained and updated, possibly by distinct sets of stakeholders.
- At any given point in time, these models satisfy a set of formal or informal inter-model consistency constraints. In other words, the models in distinct notations must not represent contradictory descriptions of the same reality.

The key to formalizing this notion is to understand inter-model consistency constraints. These are, in general, hard to obtain and must be hand-crafted for every pair of notations of interest. A more achievable approach involves, for a pair of models in distinct notations:

- Mapping each model into a model in the other notation.
- Using the consistency rules or semantics intrinsic to each notation to determine if the models are consistent. These consistency rules or semantics are usually easier to come by and even when they are not formally specified, consistency is relatively easy to manually verify.

Formally, this can be achieved by defining a mapping function in the following manner. Let $N1$ and $N2$ be two distinct modeling notations. Let $f_{N1, N2}$: $M_{N1} \rightarrow M_{N2}$ where M_{N1} and M_{N2} are the sets of all possible models expressible in N_1 and N_2, respectively, be a function that maps a model in N_1 to a model in N_2. Ideally, such a function must generate an N_2 model that expresses as much of the input model (in N_1) as can be expressed in N_2. $f_{N1,N2}$ is similarly defined. Co-evolution of models in N_1 and N_2 can then be defined as follows:

If M_1 and M_2 are the current models in N_1 and N_2, respectively, then it must be the case that $f_{N1, N2}$ (M_1) is consistent with M_2 and $f_{N2, N1}$ (M_2) is consistent with M_1.

A key question is to understand the properties of these mapping functions. Intuitively, one would expect these functions to satisfy the properties of *soundness* and *completeness* in a specialized sense. Soundness in

this context requires that only information included in the input model to the mapping function be represented in the output model. Completeness requires that as much of the information included in the input model as can be represented in the notation of the output model is indeed included in the output model.

These properties are difficult to prove formally for such mapping functions. Realizing probably sound and complete mapping functions and formally grounded machinery to support co-evolution represents a major research agenda. We present preliminary progress in that direction in this paper. To be precise (in the context of this paper), the goal of this exercise is not to guarantee consistency, but to detect inconsistencies.

3. The *i** Modelling 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 [2], [14]. 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 soft goals (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 [15]: Strategic Dependency (SD) Models and Strategic Rationale (SR) Models.

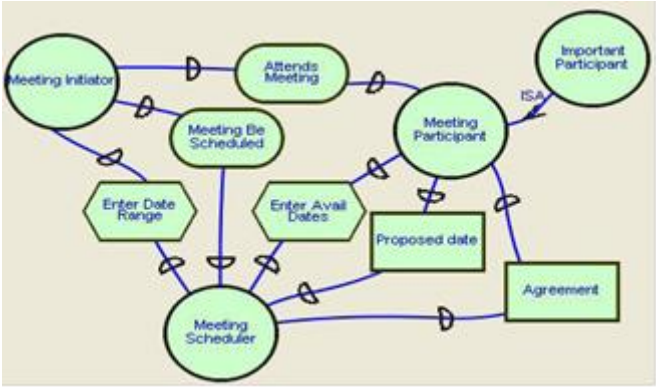

Fig. 1. SD model of the meeting scheduling system.

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 depender, 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 depender depends on the dependee to bring about a certain state in the world. The dependee is given the freedom to choose how to achieve it. In a task-dependency, the depender depends on the dependee to carry out an activity. Task- and goal-dependencies may often appear interchangeable. One way to understand the distinction is to view goals as more coarse-grained, abstract entities and tasks as more fine-grained, specific entities (while recognizing that goals can always be reformulated as tasks and vice versa). Another dimension to this distinction is the relative autonomy of the dependee in deciding how a goal is achieved, while in a task the depender and dependee must coordinate in a far more tightly-coupled fashion. In a resource-dependency, one actor (the depender) depends on the other (the dependee) for the availability of a resource. In each of the above kinds of dependencies, the depender becomes vulnerable in situations where the dependee fails to achieve a goal, perform a task or make a resource available. In a softgoal-dependency, a depender 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 [15]-[17] and is commonly used to represent optimization objectives, preferences or specifications of desirable (but not necessarily essential) states of affairs. Considering a Meeting Scheduling system which concentrates on scheduling meeting dates, the SD model in Fig. 1, and SR model in Fig. 2, involves a MeetingInitiator actor which is depends on MeetingParticipant actor to achieve AttendMeeting goal. 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.

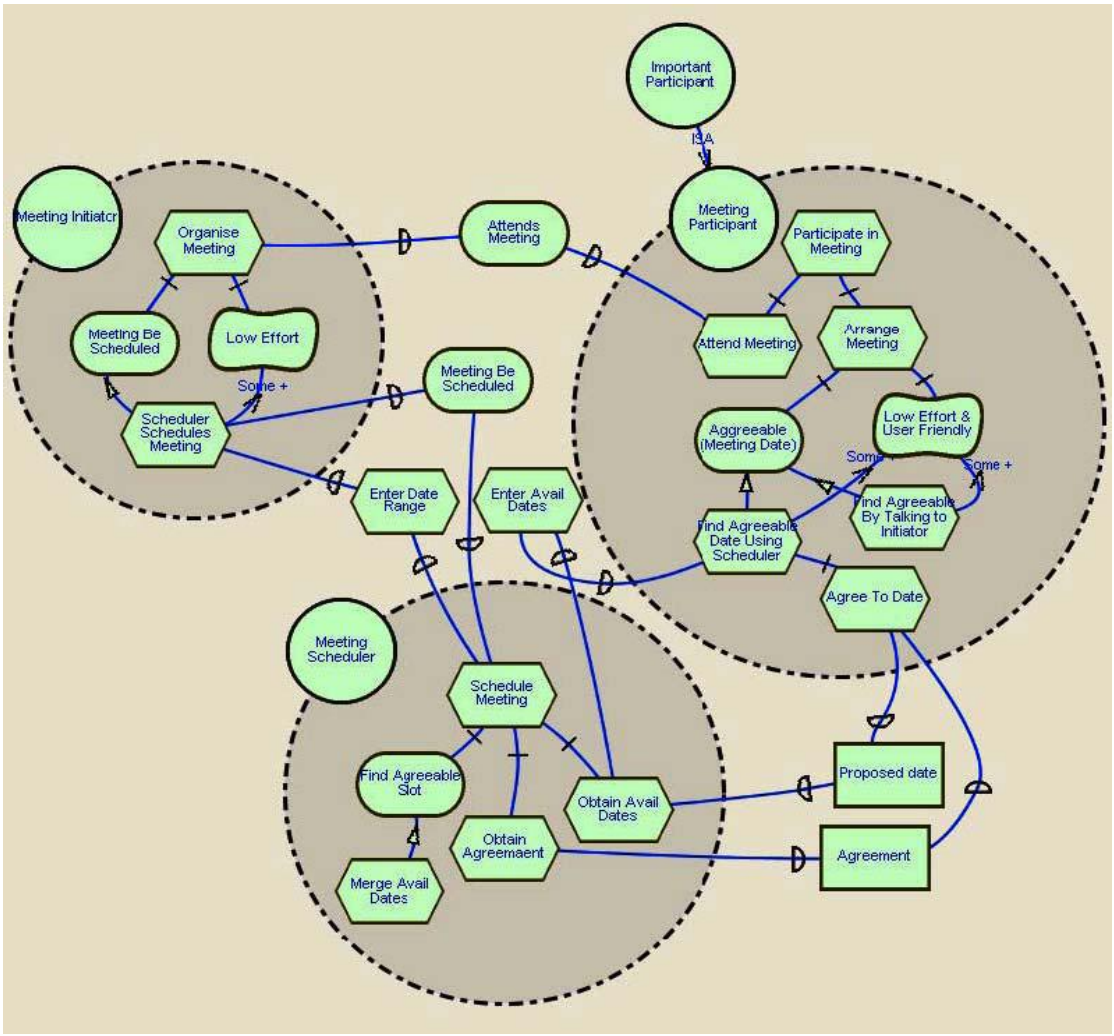

Fig. 2. SR model of the meeting scheduling system.

4. Use Case Diagrams

Use Cases are powerful modeling notation that specifies the functional requirements and behavior of the system or a part of the system [11]. It originated as a requirements modelling technique and by the mid-1990s, this modelling technique was designated as a part of the Unified Modelling Language (UML) specification. Use cases describe 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 modelling the behavior of the system or a sub-system. Each one of these represents a set of Use Cases, actors and their relationship]. This scenario based techniques has become very popular and easy to understand, model and validate user

and system requirements [10], [14]. However, specifying scenarios via Use Cases may not be sufficient for requirements specifications due to the usage of natural language. Therefore, the proposed remedy is to combine scenarios with a set of conceptual models [18]. According to the UML 2.0, a use case is "the specification of a set of actions performed by a system, which yields an observable result that is typically, of value for one or more actors or other stakeholders of the system". The Use Case diagram provides a visual view of sequence steps to achieve a task and describes the use of a system by the actors related to it [10]. 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. Use Cases also help to validate the proposed system architecture and to verify the system as it evolves during development.

5. 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 phase of software engineering. One of the benefits of this proposal is to ensure that system components and source codes are reproducible, traceable and reportable [19]. There is a need of performing precise mapping of the two complementary notations (as mentioned earlier in formal section). 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** modelling 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.
- UML notation does not support the representation of softgoals, whereas the *i** notation allows the designer to represent and reason with softgoals (representations of non-functional 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 in avoiding too many Use Case descriptions.

6. Deriving Use Case Models from Organizational Models

Use Case models can be derived from a given *i** diagram by following the guidelines proposed in [4]. 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 result of this mapping renders Use Case diagrams for the intended system and presents scenario textual descriptions for each Use Case suggesting a heuristic for their development from organizational model. The following steps will generate a Use Cases diagram for the Meeting Scheduling System (illustrated in Fig. 1 and 2).

6.1. Discovering System Actors

The first step in the mapping includes discovering appropriate actors from the SD model. Fig.-1 illustrates *i** actors that represents the organizational model. The actors in this model are: Meeting Initiator (MI), Meeting Scheduler (MS), Meeting Participant (MP) and Important Participant (IP). Meeting Scheduler (MS) actor is the system itself to be developed. According to the guideline 2 in [12], this *i** actor cannot be

taken as a Use Case actor. The other actors, MI, MP, and IP can be considered as valid and candidate actors for the Use Case Diagram, as they will interact with the intended Meeting Scheduling system. The actor Important Participant (IP) is related to Meeting Participant (MP) actor by an IS-A relationship. So, 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.

6.2. Discovering Use Cases for Actors

Guideline 5 in [12] suggests that for each candidate actor of the system, we should observe all its dependencies in which the actor is a dependee in order to discover the Use Cases of the System. The following table shows the candidate actors of the system and their dependencies, and type of dependency from the point of view of a dependee:

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 depender itself is the Use Case actor.

6.3. 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. According to guideline 7 in [12], we have the following table:

Table 2. Use Case Goal Classification

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

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 the 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 requests for agreement of meeting date. At this stage Agreement Use Case is included as <<include>>.

Although in [4] the authors have proposed this preliminary work based on goal-oriented analysis, it does not reflect the softgoals of the *i** model in corresponding Use Case diagrams. Our proposed methodology for co-evolution would propose the opportunity for an analyst/ designer to include non-functional requirements of the system. By following the guidelines we get the following Use Case Diagram:

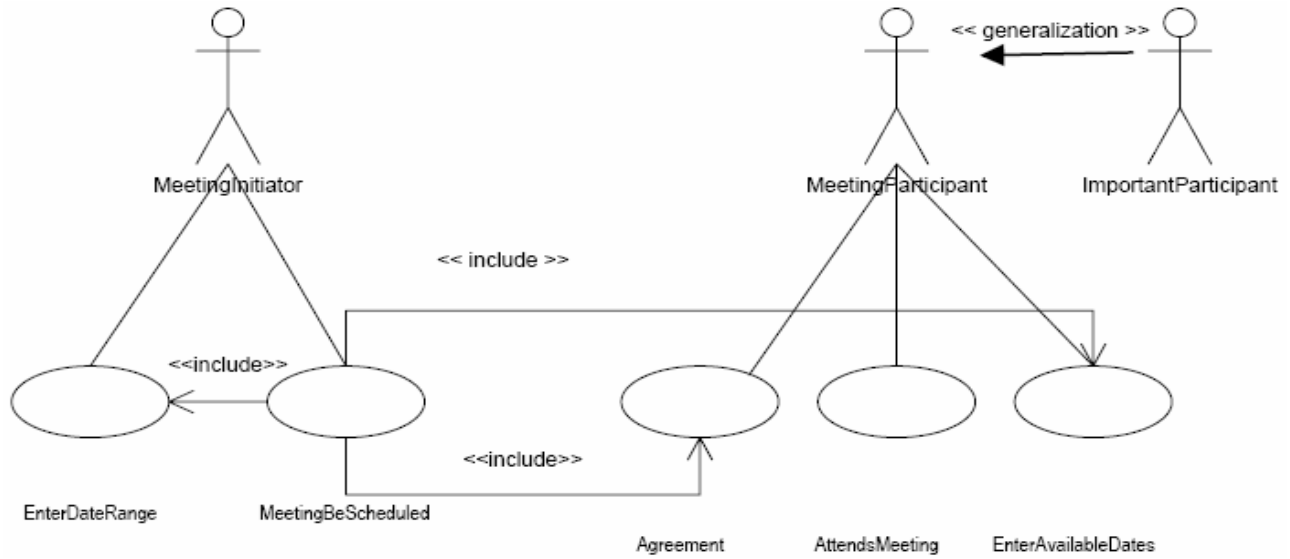

Fig. 3. Use case diagram derived from the SD and SR models of the meeting scheduling system.

7. Methodology Supporting the Co-evolution of *i** and Use Case Model

In this section we propose a methodology for the co-evolution of *i** and Use Case model with a strategy of localizing the changes. This co-evolution process involves two aspects:

1. Reflect the changes of *i** model on Use Case diagram
2. Reflect the changes of Use Case diagram on *i** model

There are sixteen categories of changes that may occur to an *i** model [20]-[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 [20]. We demonstrate our proposed methodology for the co-evolution of *i** models and Use Case diagrams by focusing on all these categories. In the following sections we illustrate the possible changes that may occur in the *i** model of the Meeting Scheduling system (Fig. 1, 2), and how these changes affect the Use Case diagram (Fig. 4, 5).

7.1. 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. Create dependencies among actors, which in turn may create 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 mechanism in the organizational model and mapped individually for the 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 with its associated Use Case will be removed and the scenario will be updated to reflect the change.

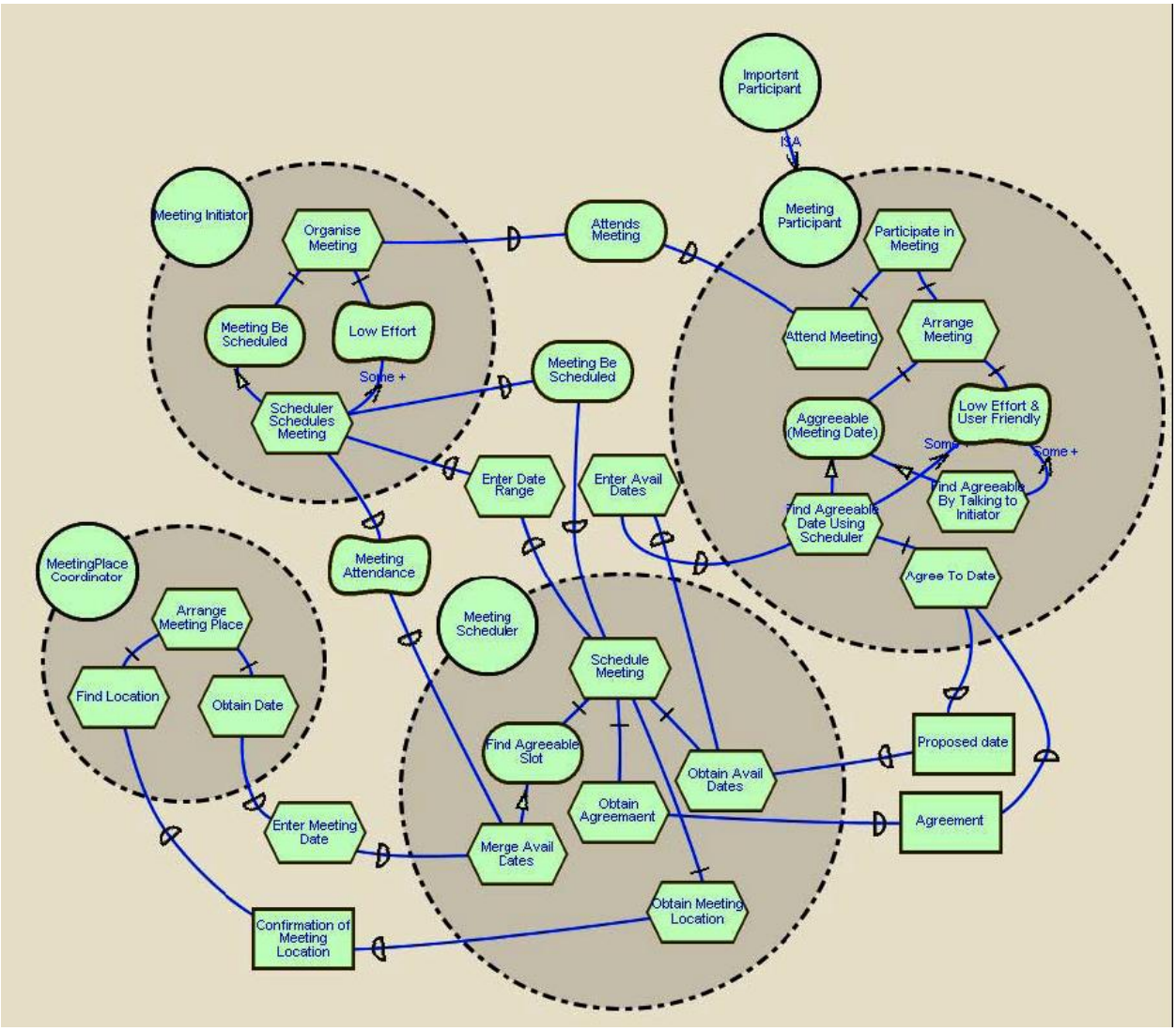

Fig. 4. SR model of the meeting scheduling system with an illustration of possible changes.

For example, if we add a new actor called Meeting Place Coordinator (MPC) (Fig. 4), it would introduce a new Use Case actor. It creates dependencies 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 Fig. 1 we observe that the actor Important Participant (IP) is related to Meeting Participant with an IS-A relationship. If any actor is added as an IS-A relationship, we represent it in the Use Case as <<generalization>> relationship.

Consider a scenario where actor MI is deleted from the *i** model. As a result, all the dependencies associated to this actor i.e. AttendsMeeting, EnterAvailableDates would be removed; hence Agreement Use Cases would be removed from the Use Case diagram. Use Case Scenario Discovery table needs to be updated to reflect these changes, in this case scenario-4 of the Use Case Scenario Discovery table will be removed.

7.2. 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 are also removed. If that dependency has a corresponding Use Case then it will be removed as well. If it does not have a corresponding Use Case, the modified SR model needs 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 and ConfirmMeetingLocation resource dependencies with the MS actor (Fig. 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.

7.3. Guideline-3: Addition/ Deletion of a Task to an Existing SR Model

A new task to an existing SR model will create a new dependency between actors directly or via links. Generally a new task in the existing SR model 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 relation where an actor depends on another actor for the accomplishment of this task 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/systems 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, a new task ArrangeMeetingPlace is added in the SR model for MPC actor (Fig. 4). This task has introduced a dependency as it delineates a relation where an actor depends on another actor for the accomplishment of the task. In this case, we should follow guideline-2 for the necessary changes in the Use Case scenario and diagram.

7.4. 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 in terms of receiving resources from one actor to another 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 which are associated with it. This modification needs to be adjusted in the discovering Use Case actors table and it will lead to modification in the Use Case scenario

describing the table too. 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 Fig. 2, 4 results into 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.

7.5. 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 includes 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 de-composed acts where they are singly linked directly or linked via 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 FindLocation 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 guidenline-2 we can make the necessary changes to our Use Cases.

7.6. 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 to earlier work [12], [13]. We view softgoals as optimization goals where there is no way of actually specifying whether the softgoal has been achieved completely or not. But, softgoals have a positive or negative contribution for achieving, accomplishing a goal, task, resource [12]. So when proposing an approach of 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.

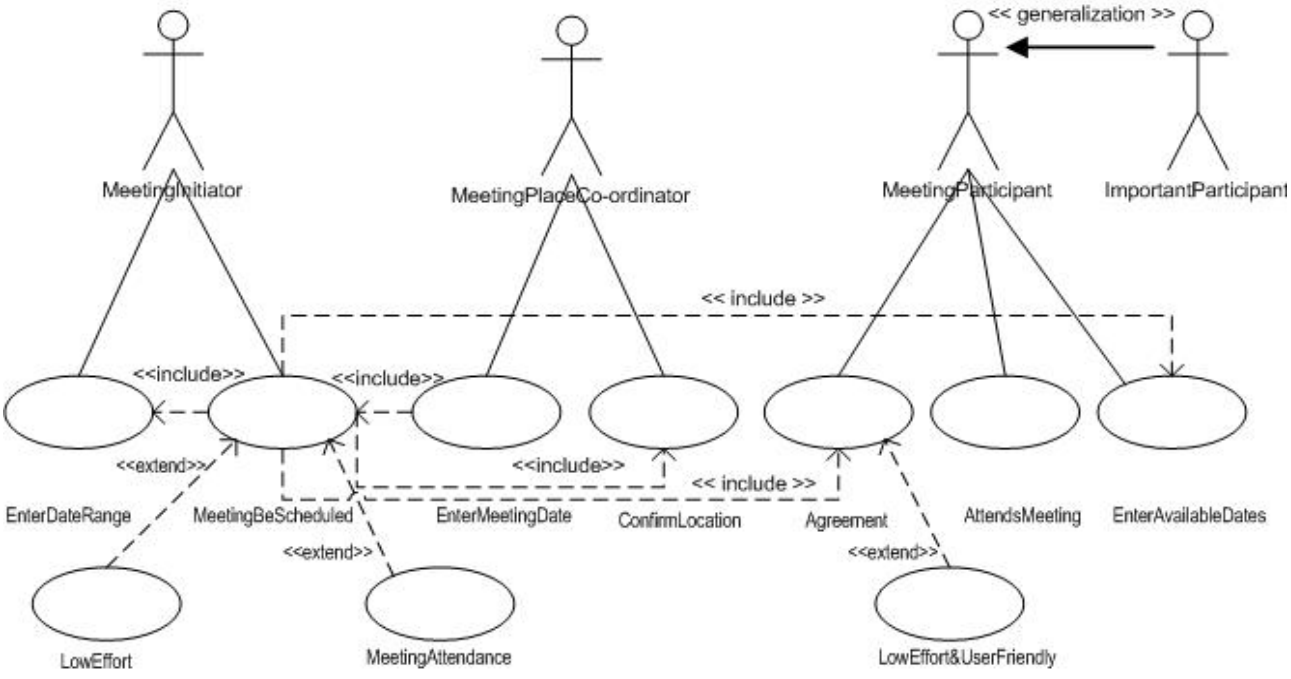

Fig. 5. Use Case Diagram reflecting the changes in *i** diagram.

This ensures that the Use Case can contribute to "satisfies" the non-functional requirement. For example the MeetingAttendance softgoal dependency in Fig. 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 AgreeableMeetingDate is satisfied by the 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.

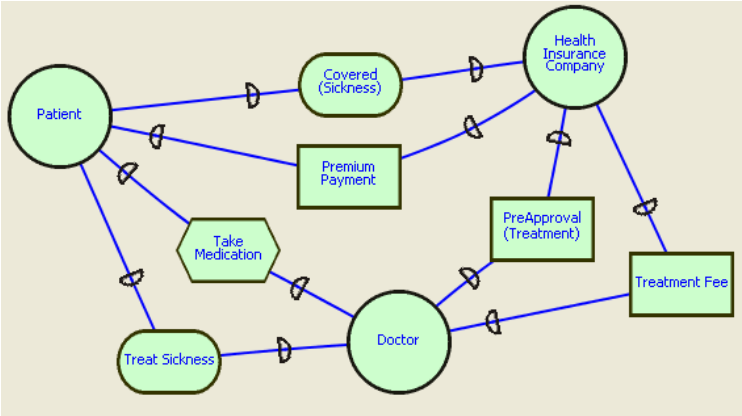

Fig. 6. SD model of the *i** Framework.

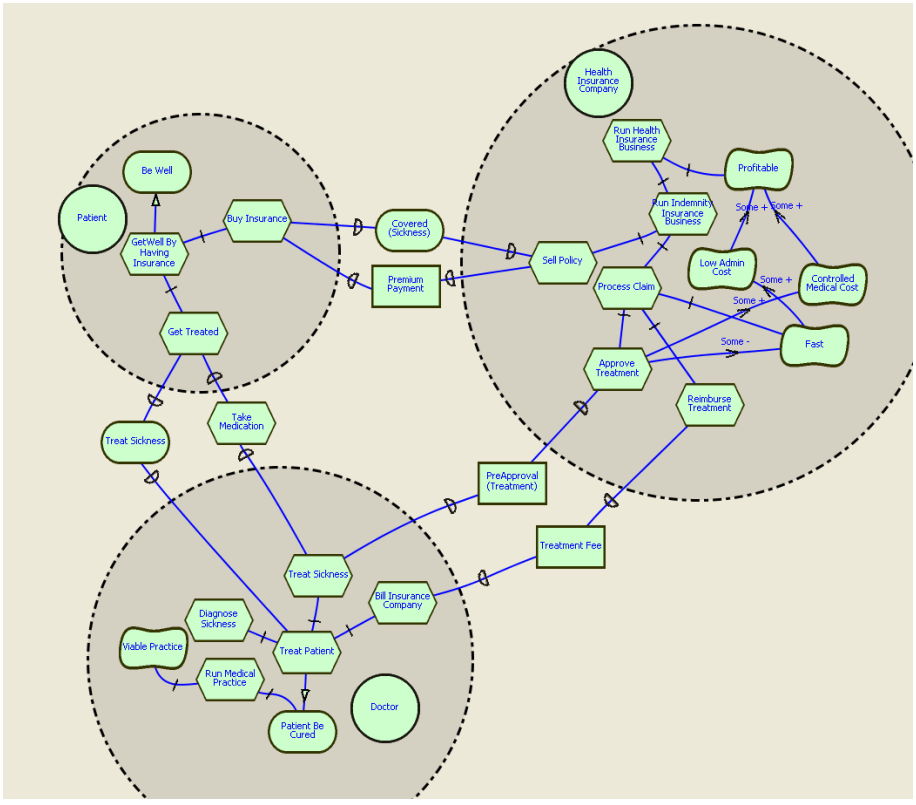

Fig. 7. SR model of *i** Framework.

8. Case Study 2

In this section we would explore another example to demonstrate the co-evolution of *i** and use case diagram. In the following Health System example, there are three actors which are identified from the SD model of the *i** diagram.

These actors are Patient, Doctor and Health Insurance Company. These actors are dependent on each other for certain goals, tasks and resources. The Patient actor is dependent on both the Doctor and Health Insurance Company to treat sickness and to get the treatment covered by the Health Insurance Company.

The Health Insurance Company is dependent on Patient and Doctor to run the Insurance business. The SR model details the internal behaviors of each actor within the diagram which is also shown on the next page.

The Patient actor buys insurance from the Health Insurance Company by paying the Premium Payment. The Doctor issues bills to Health Insurance Company to get the Treatment Fee for the Patient who is covered by the insurance company.

8.1. Deriving Use Case Model

From the above *i** model we can now derive a Use case diagram by following the procedures proposed in [12]. First we will identify the system actors by practicing the following steps.

8.2. Discovering System Actors

The system actors are identified from the SD model of the diagram. From Fig. 6, three actors are identified: Patient, Doctor and Health Insurance Company. These three actors depend on each other and communicate with each other to achieve goals, complete tasks and receive resources. We will discover the use cases for these three actors in the following steps.

8.3. Discovering Use Case Actors

According to Guideline 5 in [12]all the dependencies for each actor in which the actor is a dependee requires observation to discover the use cases.

Table 4. Use Case Discovery

ACTORS	DEPENDENCY	TYPE OF DEPENDENCY
PATIENT	Take Medication	TASK
PATIENT	Premium Payment	RESOURCE
HEALTH INSURANCE COMPANY	Covered (Sickness)	GOAL
HEALTH INSURANCE COMPANY	Pre-Approval (Treatment)	RESOURCE
DOCTOR	Treat Sickness	GOAL
DOCTOR	Treatment Fee	RESOURCE

8.4. Discovering & Describing Use Case Scenario

At this step, we observed the SR model and described the Covered (sickness) use case of Health Insurance Company actor for discovering the use case scenario as this is classified as Business goal which contains all the essential steps to describe the scenario.

Table 5. Use Case Goal Classification

ACTORS	Use Case Goal	Goal Classification
PATIENT	Take Medication	User Goal
PATIENT	Premium Payment	User Goal

HEALTH INSURANCE COMPANY	Covered (Sickness)	Business
HEALTH INSURANCE COMPANY	Pre-Approval (Treatment)	Sub function
DOCTOR	Treat Sickness	User goal
DOCTOR	Treatment Fee	Sub function

Table 6. Use Case Scenario Discovery

Scenario

1. The Patient actor initiates the use case by paying the premium payment to buy an insurance cover from Health Insurance Company actor. So the Premium Payment is included in this step by assigning <<included>> with it.
2. Once the Patient has bought the insurance, visits the Doctor for getting treatment and then Doctor give advice to patient to take medication. For this reason Treat Sickness Use Case is also included as <<include>>
3. The Doctor now sends a bill to the health insurance company, as the Patient has the health insurance cover from the company, for treating the Patient and so the Pre-Approval (Treatment) is included and Treatment Fee by assigning <<included>> to it.
4. Now observing the softgoals in SR model, they are being used as an optimization goal for the actor. Softgoals have a positive or negative involvement for accomplishing a goal, task, resource [12]. We put forward these softgoals as an extension to the use case [12]. The Profitable softgoal has created an <<extend>> relationship to Covered(Sickness) use case of Health Insurance Company and the Fast softgoal for processing tasks has been added to Pre-Approval (Treatment) uses case through extended relationship.
5. In same approach, the Viable Practice Doctor has made an <<extend>> relationship to Treat Sickness use case.

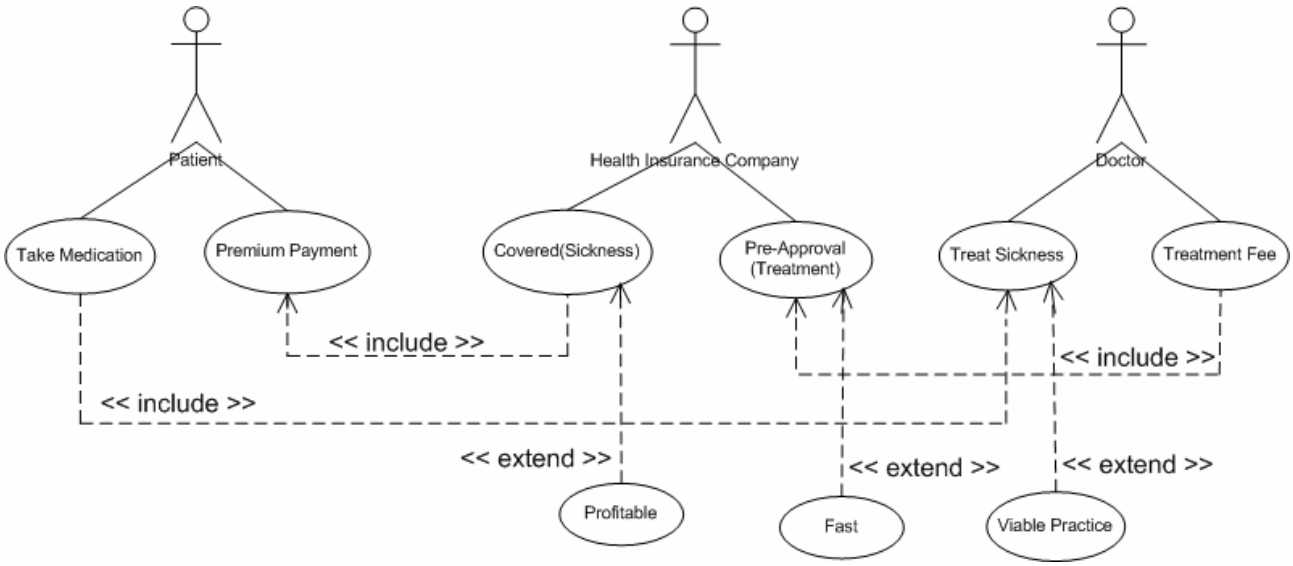

Fig. 8. Use Case Diagram derived from the SD & SR model.

9. Reflecting Changes in *i** Diagram to Corresponding Use Case Diagram

At this stage we made changes to the current *i** diagram which introduces more actors and dependencies.

In Fig. 9, two more actors Investor and Medical Lab have been introduced in the SR diagram. The Investor actor invests capital on Health Insurance Company with the expectation of achieving an optimal profit from investment.

The Medical Lab provides health testing facilities. It charges Patients for lab testing fee and then it sends the medical testing reports of Patients to the Doctor for diagnosing sickness.

According to the guideline-1, we identify that two additional actors were added to the model; these are Investor and Medical Lab. These actors would create new dependencies with other actors.

Four additional dependencies have been introduced in the model and according to guideline-2 these four dependencies will be added to the Use case discovery table as new use cases. The Optimal Return Analysis task in Investor actor creates a dependency between Investor and Health Insurance Company via the Profitable softgoal. According to the guideline-3 this task introduces a new use case for the actor.

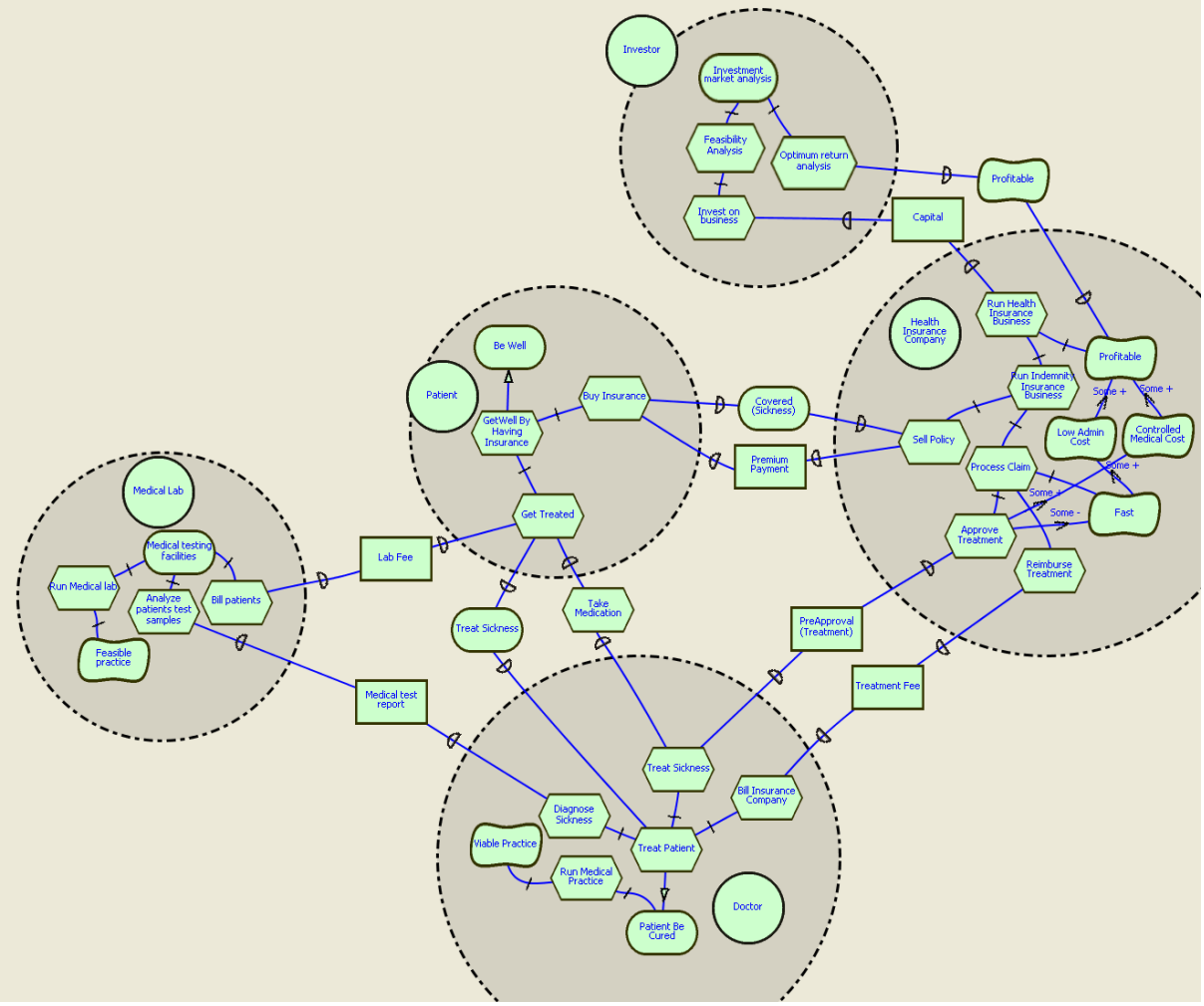

Fig. 9. Modified SR model.

Now from Fig. 9 we may now derive a new use case discovery table which is an extension to Table 4.

Table 7. Use Case Discovery Table from Fig. 9

ACTORS	DEPENDENCY	TYPE OF DEPENDENCY
PATIENT	Take Medication	TASK
PATIENT	Premium Payment	RESOURCE
PATIENT	Lab Fee	RESOURCE
HEALTH INSURANCE COMPANY	Covered (Sickness)	GOAL
HEALTH INSURANCE COMPANY	Pre-Approval (Treatment)	RESOURCE
HEALTH INSURANCE COMPANY	Profitable	SOFTGOAL
DOCTOR	Treat Sickness	GOAL
DOCTOR	Treatment Fee	RESOURCE

MEDICAL LAB	Medical test report	RESOURCE
INVESTOR	Capital	RESOURCE
INVESTOR	Optimal return analysis	TASK

We note that some reverse procedures can be followed to reflect the changes of Use Case diagram on *i** model given 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.

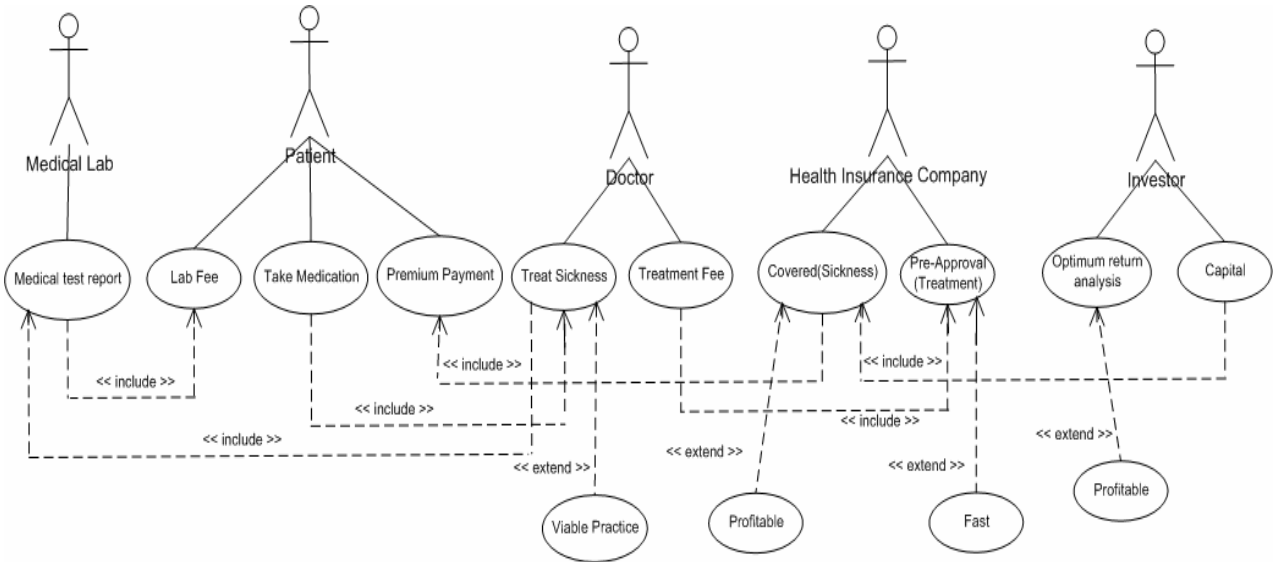

Fig. 10. Use case diagram reflecting the changes in *i** diagram.

10. Conclusion

In this paper we 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. This enables us to analyse the system behavior at a greater detail which otherwise is 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 can be viewed as providing formal semantics to the *i** diagrams while mapping into Use Case notations of UML specifications, a language which already has one. We believe that these semantics are largely consistent with the somewhat implicit semantics for *i**. The proposed set of mapping rules [12] 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.

References

[1] Yu, E. (1997). Towards modelling and reasoning support for early-phase requirements engineering. *IEEE* (pp. 226–235).

[2] Yu, E., Gunter, D. A., Franch, X., & Castro, J. (2013). Practical applications of i* in industry: The state of the art. *IEEE*.

[3] Norta, A., Mahunnah, M. Tenso, T., & Taveter, K. (2014). An agent-oriented method for designing large socio-technical service-ecosystems. *Proceedings of IEEE 10th World Congress on Services*

[4] Castro, J., Kolp, M., & Mylopoulos, J. (2002), Towards requirements-driven information systems engineering: The tropos project. Elsevier, Amsterdam, The Netherlands.

[5] Costal, D., Gross, D., *et al.* (2014), Quantifying the impact of OSS adoption Risks with the help of i* model. GESSI Research Group.

[6] Bhuiyan, M. (2012). *Managing Process Design in a Dynamic Organisational Context*. University of Wollonggong. Australia.

[7] Mayer, N., Dubois, E., & Rifaut, A. (2007). *Enterprise Interoperability II: Requirements Engineering for improving Business/IT Alignment in Security Risk Management Methods*. Springer London.

[8] Shabnam, L., Haque, F., & Bhuiyan, M. (2014). Risk measurement propagation through organization network. *Proceedings of IEEE 38th Annual International Computers, Software and Applications Conference Workshops.*

[9] Bennett, S., McRobb, S. & Farmer, R. (2002). Object-oriented systems analysis and design using UML. London: McGraw-Hill.

[10] Jurkiewicz, J., & Nawrocki, J. (2015). Automated events identification in use cases. *Information and Software Technology, 58*, 110-122.

[11] Tiwari, S. & Gupta, A. (2015). A systematic literature review of use case specifications research. *Information and Software Technology, 67*, 128-158.

[12] Santander, V. F. A., & Castro, J. F. B. (2002). Deriving use cases from organizational modelling. *Proceedings of IEEE Joint Conference on Retirements Engineering*, Essen, Germany.

[13] Santander, V. F. A., & Castro, J. F. B. (2002). *Deriving Use Cases From Organizational Modeling*. Los Alamitos, California, USA: IEEE, *1*, 32–39.

[14] Kenett, R., Franch, X., Susi, A., & Galanis, N. (2014). Adoption of free library open source software (FLOSS): A risks management perspective. *Proceedings of 2014 IEEE 38th Annual Computer Software and Applications Conference (COMPSAC)*. 171, 180, 21-25.

[15] Chung, L., & Nixon, B. (1995). Dealing with non-functional requirements: three experimental studies of a process-oriented approach. *Proceeding of 17th International Conference on Software Engineering, Seattle*, Washington, 24-28.

[16] Chung, L. (1993). 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.

[17] Chung, L., Nixon, B., Yu, E., & Mylopoulos, J. (2000) *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 472.

[18] Wang, Y., Zhao, L., Wang, X., Yang, X. & Supakkul, S. (2013). PLANT: A pattern language for transforming scenarios into requirements models. *International Journal of Human-Computer Studies, 71(11)*, 1026-1043.

[19] Research on building baseline of IT risk control and its application in IT risks Management. *Management Science and Engineering*. CS Canada, *8(3)*, 11-16.

[20] Krishna, A., Ghose, A. & Vilkomir, S. (2004). Consistency preserving co-evolution of formal and informal models. *Proceedings of World Congress on Lateral-Computing (WCLC-2004)*, Bangalore, India.

[21] Krishna, A., Ghose, A & Vilkomir, S. (2004). Co-evolution of complementary formal and informal requirements. *Proceedings of 7th International Workshop on Principles of Software Evolution* (pp. 159-164). Kyoto, Japan.

[22] Vilkomir, S., Ghose, V. & Krishna, A. (2004). Combining agent-oriented conceptual modeling with

formal methods. *Proceedings of 15th Australian Software Engineering Conference* (pp. 147-157). Melbourne, Australia, IEEE Computer Society.

Moshiur Bhuiyan is an experienced IT Management Consultant, who possesses extensive expertise in Management Consulting, Business Analysis, BPM, Change Management and Enterprise Architecture. He has significant passion in research and teaching. His research areas include but are not limited to Business Process Discovery & Modelling, Process Rules and Policy Integration, Process Execution, Process Reengineering and Optimization, Process Lifecycle Management, Change Management, Software Requirement Engineering, Cloud Computing, ICT Governance & Architecture. He has published his works in reputed international conferences and journals. He has served as program committee member and reviewer in several conferences and workshops. Dr. Moshiur is also the founder member of a technology entrepreneurship company named Enterprise Cloud Systems (www.ecloudsys.com) which develops innovative cloud applications.

Farzana Haque is an undergraduate student of Computer Science & Engineering at North South University, Bangladesh. She has a sharp analytical mind, with a powerful leaning towards complex problem solving and always willing to innovate the new things which can improve the existing business and technical processes. Her research areas include Data Mining, Risk Management, and Fuzzy Logic.

Luba Shabnam is a PhD candidate in the School of Chemical and Bimolecular Engineering at the University of Sydney. She is a recipient of Australian Postgraduate Award in 2014. Luba holds Bachelor (with Honors) and Masters of Science in Applied Chemistry and Chemical Technology. Her research areas include conceptual modelling, risk management, electrochemical sensing platform, chemical process simulation etc.