RESEARCH ONLINE

University of Wollongong
Research Online

Faculty of Informatics - Papers (Archive)

Faculty of Engineering and Information Sciences

2007

Integration of Agent-Oriented Conceptual Models and UML Activity Diagrams Using Effect Annotations

M. Bhuiyan
University of Wollongong

M. M. Islam
University of Wollongong

A. Krishna
University of Wollongong, aneesh_krishna@uow.edu.au

Aditya K. Ghose
University of Wollongong, aditya@uow.edu.au

Publication Details

This conference paper was originally published as Bhuiyan, M, Islam, MMZ, Krishna, A, Ghose, A, Integration of Agent-Oriented Conceptual Models and UML Activity Diagrams Using Effect Annotations, 31st IEEE Annual International Computer Software and Applications Conference COMPSAC 2007, 24-27 July, Vol 1, 171-178.

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au

RESEARCH ONLINE

Integration of Agent-Oriented Conceptual Models and UML Activity Diagrams Using Effect Annotations

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, Unified Modeling Language (UML) is suitable for later phases of requirement capture which usually focus on completeness, consistency, and automated verification of functional requirements for the new system. In this paper, we propose a methodology to facilitate and support the combined use of notation for modeling requirement engineering process in a synergistic fashion. For organizational modeling/early phase requirements capturing we use the i* modeling framework that describes the organizational relationships among various actors and their rationales. For late (functional) requirements specification, we rely on UML Activity Diagram.

Disciplines

Physical Sciences and Mathematics

Publication Details

This conference paper was originally published as Bhuiyan, M, Islam, MMZ, Krishna, A, Ghose, A, Integration of Agent-Oriented Conceptual Models and UML Activity Diagrams Using Effect Annotations, 31st IEEE Annual International Computer Software and Applications Conference COMPSAC 2007, 24-27 July, Vol 1, 171-178.

This journal article is available at Research Online: http://ro.uow.edu.au/infopapers/625

Integration of Agent-Oriented Conceptual Models and UML Activity Diagrams Using Effect Annotations

Moshiur 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, Unified Modeling Language (UML) is suitable for later phases of requirement capture which usually focus on completeness, consistency, and automated verification of functional requirements for the new system. In this paper, we propose a methodology to facilitate and support the combined use of notation for modeling requirement engineering process in a synergistic fashion. For organizational modeling/early phase requirements capturing we use the i* modeling framework that describes the organizational relationships among various actors and their rationales. For late (functional) requirements specification, we rely on UML Activity Diagram.*

1. Introduction

Understanding the organizational environment as well as the reasoning and rationale underlying requirements, design and process formulation decisions is crucial to model and build effective computing systems. Conceptual modeling notations employing knowledge representation techniques have been developed to support such an understanding [14]. Many modeling techniques tend to address "late phase" requirements while the vast majority of critical modeling is arguably taken in early phase requirements engineering. Agent-oriented Conceptual Modeling (AOCM) offers an interesting approach in modeling the early phase requirements. The *i** modeling framework [14] is a semi-formal notation built on agent-oriented conceptual modeling.

The central concept in *i** is that of the intentional actor agent. Intentional properties of an agent such as goals, beliefs, abilities and commitments are used in modeling requirements. 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 modeling of rationale by representing key internal intentional characteristics of actors/agents.

A number of proposals have been made for combining *i** modeling with late phase requirements analysis and the downstream stages of the software lifecycle. The TROPOS project [2] uses the *i** notation to represent early and late phase requirements, architectures and detailed designs. However, the *i** notation itself is not expressive enough to represent late phase requirements, architectures and designs. To address this problem, a custom designed formal language called FormalTropos [6] has been proposed. Proposals to integrate *i** with formal agent programming languages and formal methods have also been reported in the literature [8] [12] [13]. 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 mapping of *i** models with more traditional specifications. In the current instance, we examine how this might be done with Unified Modeling Language (UML) [1]. 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 UML Activity Diagram. 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 that lead towards system modeling. In this paper, we propose some guidelines to facilitate and support the combined use of notations for modeling requirement-engineering process in a synergistic fashion. For organizational modeling early phase requirements capturing we use the *i** modeling framework that describes the organizational relationships among various actors and their rationales. For late (functional) requirements specification, we rely on a UML Activity Diagram. The heuristics described in this paper helps the system modeler to develop activity diagram based on *i** models.

In Section 2 & 3, below, we present *i** modeling framework and UML Activity Diagrams with an example. Section 4 presents benefits of mapping *i** models into Activity Diagram. Section 5 discusses a methodology

31st Annual International Computer Software and Applications Conference(COMPSAC 2007)
0-7695-2870-8/07 $25.00 © 2007 IEEE

supporting the mapping of *i** into UML Activity Diagram using effect annotations. Section 6 contains a discussion on reflecting changes in an *i** model to an associated Activity Model. Finally, Section 7 presents some concluding remarks.

2. The *i** Modeling Framework

The *i** framework for agent-oriented conceptual modeling was designed primarily for early phase requirements engineering. The central concept in *i** is that of the intentional actor (agent). Intentional properties of an agent such as goals, beliefs, abilities and commitments are used in modeling requirements. The *i** framework consists of two main modeling components: the Strategic Dependency (SD) Model and the Strategic Rationale (SR) Model [14] [15].

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*.

Strategic Dependency Models: An Example:

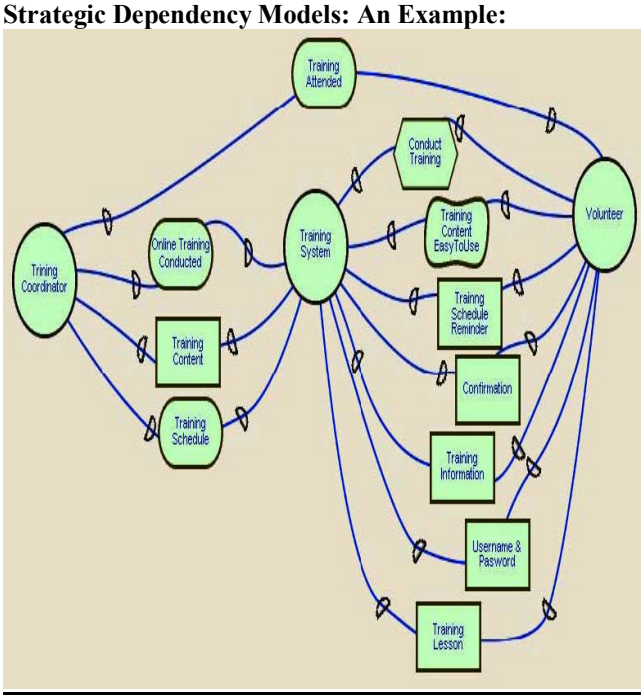

Figure 1: A Strategic Dependency model for computer based training system

The SD model provides an important level of abstraction for describing systems in relation to their environments, in terms of intentional relationships among them. This allows the modeler to understand and analyse new or existing organisational and system configurations even if the internal goals and beliefs of individual agents are not known.

An example concerning a computer based training system (CBT) for volunteers of emergency services will be used to illustrate the SD Model notation (see figure 1 for the model). The *TrainingCoordinator* agent depends on *Volunteer* agents to achieve its *TrainingAttended* goal. The *TrainingCoordinator* has two goal dependencies on the *TrainingSystem*, *TrainingScheduled* and *OnlineTraining-Conducted* (i.e., the *TrainingCoordinator* agent relies on the *TrainingSystem* agent to schedule training sessions and to conduct online training). The *TrainingSystem* has a dependency on the *TrainingCoordinator* to provide *TrainingContent*, modeled as a resource dependency. The *TrainingSystem* has a dependency on *Volunteers* to achieve its *TrainingAttended* goal. The *TrainingSystem* has a dependency on *Volunteers* to provide *Confirmation* of their attendance, modeled as a resource dependency. *Volunteers* depend on the *TrainingSystem* to perform the *Conductraining* task. Observe that we have chosen not to model this as a goal dependency since the *TrainingSystem* cannot autonomously decide how the corresponding goal might be achieved but must work with the *depender* in a tightly coupled fashion to perform the task. *Volunteers* have a further dependency on the *TrainingSystem* to *TrainingScheduleRemider* and *TrainingInformation*, modeled as resource dependencies. *Volunteers* have a preference for the *TrainingSystem* to satisfy the softgoal *TrainingModulesEasyToUse*. The notion of a softgoal derives from the Non-Functional Requirements (NFR) framework [3] [4] and is commonly used to represent optimisation objectives, preferences or specifications of desirable (but not necessarily essential) states of affairs.

The Strategic Rationale Models: An Example

In the *i** framework, the SR model provides a more detailed level of modeling by looking "inside" actors to model internal intentional relationships. Intentional elements (goals, tasks, resources, and softgoals) appear in the SR model not only as external dependencies, but also as internal elements linked by task decomposition and means-ends relationships (figure 2). The SR model in figure 2 thus elaborates on the relationships between the *TrainingCoordinator*, *TrainingSystem* and *Volunteer* as represented in the SD model of figure-1.

For example, the *TrainingCoordinator* has an internal task to *OrganiseTraining*. This task can be performed by sub-tasks *ScheduleTraining* and *GenerateTrainingContent* (these are related to the parent task via task decomposition links). The task *OrganizeTraining* is related to the *LowEffort*, *Quick* softgoals via a task decomposition link. The intention is not to suggest that the softgoal plays the role of a sub-task but to relate the softgoal to the highest-level task for which the softgoal may be viewed as an optimization objective. The softgoal thus serves to con-

strain design decisions on how the task might be decomposed. In this instance, the contribution is positive, i.e., organizing the training material contributes (positively) to achieving the broader goal of making the *TrainingMaterialEasyToUse*.

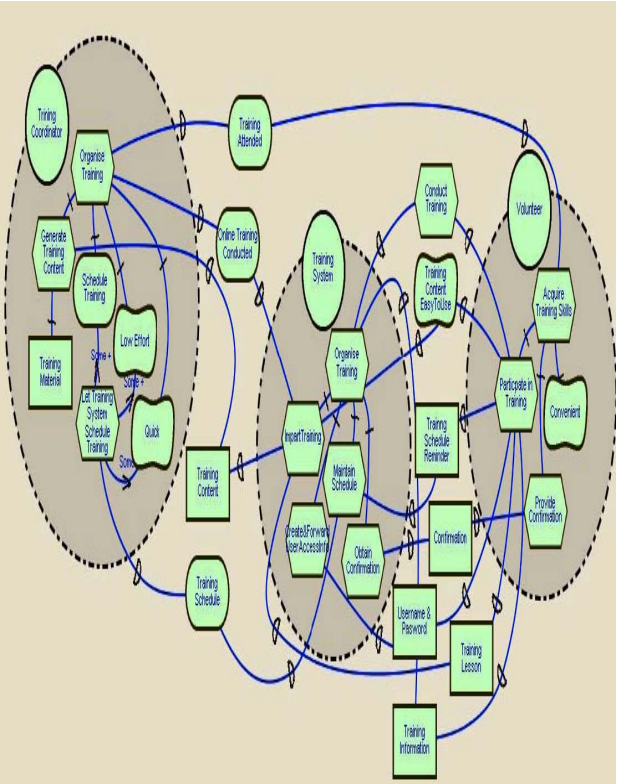

Figure 2: Strategic Rationale model for computer based training system

3. Activity Diagram in UML

An activity diagram is an uncomplicated and perceptive illustration that depicts the actions, parallel activities and any possible alternative ways through the workflow. Activity diagrams defined in the Unified Modeling Language [11] are consequential from various methods to pictorially express sequence of activities or sub-activities and conditions taken within a process. Activity Diagrams explain the operational flow from an initiating point to the terminating point specifying many decision paths that exist in the development of processes contained in the activity. They are also used to explain states where parallel processing may occur in carrying out of some activities.

The design of an activity diagram may demonstrate the organisational stage or the system stage. The most important exercise of using activity diagrams are in designing the operational progression that defines the sequences of operations and the realization of operation. In a system development process, design is important as the lack of good design of processes will lead to non-maintainable, non-reusable system having obscure functionality [9]. Activity diagrams are useful and important for modeling the dynamic aspects of a system for several reasons; it describes the internal actions of an operation graphically, helps to recognize activities whose accountability belongs to another place, illustrates activities that can occur in parallel, allows the detection of common functionality within a system and can construct executable systems through forward and reverse engineering.

Graphically an activity diagram is an anthology of vertices and arcs which generally contains activity states, action states, transitions and objects. Activity states are non-atomic as they can be interrupted and usually they may take some time to be accomplished. But action states are atomic, their work is non-interrupted. Action states can not be decomposed. Transitions depict the path initiated from one action or activity state and passed to next action or activity state as the action or activity of a state is completed. Transitions are represented as a simple directed line in the activity diagram [1].

4. Benefits of Mapping *i** model into Activity Diagram

Constructing a system that adheres to organisational environment and meets end users need (such as determining the main goals of the intended system, relations and dependencies among stakeholders, alternatives in the early-stage requirements analysis etc.), requires developing clearly defined early stage functional requirements. The *i** modeling framework which is a semi-formal notation built on agent-oriented conceptual modeling is well suited for this purpose.

We need to focus on the functional and non-functional requirements of the system as we continue the development process. In this phase we can adopt the UML activity diagram to discover and reason about the functional requirements of the system. An activity diagram is a dynamic illustration, which demonstrates the movement and the event of objects in the particular state. It clearly supports parallel activities and their synchronization. Activity diagrams are functional for analysing actions and the states of a use case, illustrating complex sequential algorithm and designing applications with parallel processes [1] [9] [10]. They represent the operational workflow of a system by capturing actions performed and provide a broad representation of the overall flow. Some benefits of integrating these two notations are given below.

• We feel that the usefulness and effectiveness of *i** can be increased manifold by using it with UML activity diagram. Mapping rules provide a semantics to *i** framework. Our view is that the *i** modeling framework and UML activity diagram can function in a complementary and synergistic way.

• There is a need to map both SD and SR models into late phase requirements specification. Activity diagram can be

used effectively to realize the actions and states in the late phase which cannot be represented in the *i** diagram.

• For translating informal specifications provided in *i** into Activity diagram, there is no need to add more details into the corresponding *i** model. The mapping from *i** models into Activity diagram does not result in any information loss.

• Using Activity diagram, we are in a position to express properties that are not restricted to the current state of the system, but also to its past and future history.

5. Methodology Supporting the Integration of *i** and UML Activity Diagram

We shall provide some guidelines for the mapping of *i** model into UML activity diagram. Mapping will be done in two phases; phase-1 effect annotations, phase-2 mapping rules. These guidelines ensure the consistency of the generated activity diagram with the initial *i** model.

Our proposed methodology uses the notion of cumulative effect annotation to determine whether the *i** models and UML Activity Diagrams are consistent with each other. An effect is the result (outcome) of an activity being executed by some cause or agent. It indicates the achievement of a certain environmental state communicated through an event. In our work, every goal/task/resource dependency must have an effect annotation. A cause relationship exists between an activity and an effect. In other words, activity causes the effects to occur. An activity can cause many effects and an effect can be caused by a number of activities. For each selected dependency we have an object in the UML Activity Diagram with the same effect. This we consider as a weak notion of consistency. It clearly states the result of activity if the conceptual model were to be theoretically executed. We also annotate every task in the SR model that is related to a dependency with a cumulative effect annotation. We then use the Activity Diagram and annotate actions with effects. Our approach ensures that a dependency is achieved through the cumulative effect of the actions on the UML Activity Diagram. This we refer to as strong consistency. Using this notion of cumulative effect annotations an analyst can ensure that a UML Activity Diagram is consistent with respect to the *i** model under this regime.

5.1 Consistency Evaluation

We introduce consistency rules to provide a mechanism for ensuring consistency between *i** model and UML Activity Diagram. The rules are developed with consideration to [7].

Rule 1: Every actor in an *i** model required as a participant in the Activity Diagram must be represented in the model. Required participants are identified via the associated dependencies among the actors.

Rule 2: Every 'primitively workable' task decomposed (or required by decomposition where a dependency exists) from the chosen routine within the *i** model, must be represented as an action or activity under the control of the appropriate actor in the process model.

Rule 3: There must exist a coordinated transition in the Activity Diagram, whereby the operational objective (as encoded in the fulfillment conditions or effect annotations) of the routine is achieved, and the sequence of activities is consistent with the requirements specified in the routine. There must exist a coordination of activities in the activity diagram that satisfy the requirements of the routine further outlined below.

Rule 3.1: The fulfillment conditions of the operational goal at the root of the routine and all its sub-elements must be achieved through the accumulation of effects during forward traversal of the transition.

Rule 3.2: The fulfillment conditions of a task in the chosen routine must not be fulfilled prior to all tasks that decompose it, upon accumulation of effects during forward traversal of the transition.

Rule 3.3: The fulfillment of a task on the *depender* side of a dependency must not be realized before the fulfillment of the dependency, upon the accumulation of effects during forward traversal of the transition.

5.2 Phase 1: Effect Annotations

The concept of effect annotation denotes the potential outcomes of activities and fulfilment conditions that are required to meet dependencies by achieving certain results. An effect generally defines that a result or consequence of an activity has generated because of its being accomplished by an agent or some previous phenomenon. As an example, effects can be annotated to activity/task nodes or even complete sub-processes in graphical notations. In *i**, we annotate effects to tasks assigned to actors which indicate the realization of a certain conditions aimed in the direction of (i.e. and perhaps required for) some higher order goal. The effect annotations is intended to provide a notation free methodology rather than limited to a specific notation. An effect annotation is a testimonial to the outcome of an activity related to a state that alters construction of a given model.

An effect annotation includes: a label that generalizes the effect (e.g. '*CustomerDetailsStored*'); a designation specifying whether the effect is a normal (i.e. desired) outcome for an activity (e.g. '*RegistrationValidated*'), or an abnormal (i.e. undesired) outcome for the activity that may require the application of some mitigation strategy; an optional informal definition describing the effect in relation to the result achieved in its environment (e.g. 'The details relating to the current customer have been stored within the system.'); an optional formal definition may be used to define achieved states in a chosen formalism.

Fulfillment conditions are annotated to intentional actor elements and dependencies in an *i** model (i.e. not including softgoals as these are used during assessment of alternatives and describe non-functional properties to be

IEEE COMPUTER SOCIETY

addressed). A fulfillment condition [7] is a statement specifying the outcomes required to satisfy a given goal or dependency. Fulfillment conditions recognize the required effects on a business process model. For example, a fulfillment condition for a task dependency to '*ConductTraining*', may be the '*TrainingArranged*' effect (subsequently required by the task assigned to a *dependee* actor).

Intuitively, for a dependency to be fulfilled, explicit assignment of responsibility is made to a *dependee* actor who possesses an intentional element that can satisfy the dependency. Therefore, one guiding rule during the annotation of fulfillment conditions to an *i** model is that all fulfillment conditions annotated to a dependency must be annotated to the intentional element the dependency is linked to on the *dependee*.

In this case we are only concerned with the fact that the *dependee* has the knowledge to achieve the dependency, not the ability (e.g. where another dependency may be required with another actor). We have introduced two steps to derive the effect annotations from the CBT *i** diagram. Step 1, Annotate the *i** model with effects and then derive the annotations of dependencies with fulfilment conditions. Step 2, define fulfilment conditions to the tasks that Realizes/Requires the fulfilment conditions.

Step 1: Annotate model with effects and/or fulfilment condition

The tasks assigned to the actors in the CBT model are initially annotated with effects. Table 1, illustrates the annotation in a tabular form.

Actor	Task	Effect Annotation
TC	Let Training System Schedule Training	Training System Schedule Training
TC	Generate Training Content	Training Material Generated
TC	Organize Training	Training Organized
TS	Obtain Confirmation	Confirmation obtained
TS	Create & Forward User Access Info	User Name & Password created
TS	Impart Training	Training Imparted
TS	Maintain Schedule	Training Schedule Maintained
TS	Arrange Training	Training Arranged
Vol	Provide Confirmation	Confirmation Provided
Vol	Participate in Training	Participated in Training
Vol	Acquire Training Skills	Training Skills Acquired

Table-1: Annotation of tasks with effects.

The second segment of the model annotation involves annotating dependencies with fulfilment conditions that relate to required effects in the *i** model. The following table depicts the dependency among the actors and their fulfilment condition to meet the dependencies in the Training System model.

Dependency	Fulfillment Condition
Training Content	TC: Training Content Generated
Training Schedule	TS: Scheduled Training
Confirmation	Vol: Confirmation Provided
Username & Password	TS: Username & Password Created
Training Schedule Reminder	TS: Training Schedule Reminded
Online Training Conducted	TS: Training Conducted
Training Lesson	TS: Training Lesson provided
Training Information	TS: Training Information Provided
Conduct Training	TS: Training Arranged
Training Attended	Vol: Acquired Training Skills

Table -2: Annotation of dependencies with fulfilment conditions

Step 2: Propagate fulfilment conditions in *i** models to task assigned to *dependee* and *depender* actors

The analysis of dependency proliferate effect annotations of dependencies into tasks that realise/require the fulfilment conditions. The task that realizes the dependency obtains the effect annotation as a required post and task requiring the dependency obtains the effect annotations as a required pre-condition. The following table illustrates the dependency with the fulfilment conditions and tasks that realize/require condition.

Dependency	Fulfilment Condition	Task – Realizes Fulfilment Condition	Task – Requires Fulfilment Condition
Training Content	TC: Training Content Generated	TC: Generate Training Content	TS: Impart Training
Training Scheduled	TS: Schedule Training	TS: Maintain Schedule	TC: Let Training System Schedule Training
Confirmation	Vol: Confirmation Provided	Vol: Provide Confirmation	TS: Obtain Confirmation

Username & Password	TS: Username & Password Created	TS: Create & Forward User Access Info	Vol: Participate in Training
Training Schedule Reminder	TS: Training Schedule Reminded	TS: Maintain Schedule	Vol: Participate in Training
Online Training Conducted	TS: Training Conducted	TS: Impart Training	TC: Organize Training
Training Lesson	TS: Training Lesson provided	TS: Impart Training	Vol: Participate in Training
Training Information	TS: Training Information Provided	TS: Arrange Training	Vol: Participate in Training
Conduct Training	TS: Training Conducted	TS: Arrange Training	Vol: Participate in Training
Training Attended	Vol: Acquired Training Skills	TC: Organize Training	Vol: Acquired Training Skills

Table-3: Tasks that Realizes/Requires the fulfilment conditions

Now we have the effect annotations for the intentional elements such as goals, resources and tasks. The dependency analysis will recognize the pre/post conditions of the elements. Below is an illustration of the fulfilment condition propagation of the Training System Model in Table 4.

Task	Effect Annotation	Required Pre
TC: Let Training System Schedule Training	Training System Schedule Training	TC: Organize Training Schedule
TC: Generate Training Content	Training Material Generated	TC: Conduct a computer based training
TC: Organize Training	Training Organized	TS: Impart Training
TS: Obtain Confirmation	Confirmation obtained	Vol: Provide Confirmation
TS: Create & Forward User Access Info	User Name & Password created	TS: Arrange Training
TS: Impart Training	Training Imparted	TC: Generate Training Content
TS: Maintain Schedule	Training Schedule Maintained	TC: Let Training System Schedule Training
TS: Arrange Training	Training Arranged	TC: Organise Training
Vol: Provide Confirmation	Confirmation Provided	None
Vol: Participate in Training	Participated in Training	TS: Create & Forward User Access Info TS: Training Lesson TS: Training Schedule Reminder TS: Training Information TS: Conduct Training
Vol: Acquire Training Skills	Training Skills Acquired	Vol: Attend Training

Table-4: Propagation of Fulfilment conditions to respective tasks

5.3 Phase-2: Mapping Rules

Rule-1: Discover the actors and represent them in activity diagram

We should go through the *i** model to discover the actors. This step can be completed by looking at either SD model or SR models. Once the actors are found they will be placed as the names of the swimlanes of the activity diagram. We prefer using swimlanes pattern of the activity diagram as they are used to organize responsibilities for the actions. They can often correspond to organisational units in a business process model. Each swimlane represents a high level responsibility for part of the overall activity of an activity diagram. Every activity will belong to exactly one swimlane, but transitions may cross lanes.

For example, to discover the actors in the CBT system, we can look at the SD model in figure 1. From the SD model we get three actors, *Training Coordinator*, *Training System* and *Volunteer*. When we map the *i** model into UML activity diagram, these actors are represented in the swimlanes to show the responsibilities of each actor (for each actions) associated with the overall system

Rule -2: Discover task/ actions

In this step we need to identify the tasks involved in the system. SR model of the *i** diagram shows the internal tasks and their rationales. For each actor, the SR model will be analysed to discover the tasks. In our methodology identification of tasks/actions and their effects has been analysed in the effect annotation part. We will take the tasks from table-1 for the mapping and then categorize them according to the actors.

We can discover the task of the CBT system by looking at its SR model. This model represents all internal tasks and their rationales. From table-1 we get the complete list of tasks with their effect annotations. Tasks in *i** model will be regarded as actions in activity diagram. Thus *TrainingCoordinator* has *GenerateTrainingContent*,

LetTrainingSystemScheduleTraining and *OrganiseTraining* actions, *TrainingSystem* has *ObtainConfirmation*, *Create&ForwardUserAcessInfo*, *MaintainSchedule*, *ImpartTraining* and *ArrangeTraining* actions, and *Volunteer* has *ProvideConfirmation*, *ParticipateInTraining* and *AcquireTrainingSkills* actions.

Rule -3: Identify the Initiating Actor

Among the discovered actors we need to find the initiating actor. This actor will be responsible for the initial action in the activity diagram. The initiating actor can be identified through their ability to satisfy the pre-condition with an action that realizes the required effect. The initial actor will be represented in the first swimlane of the activity diagram.

There are three actors in the CBT system. To find the initiating actor, we need to analyse the actions, their effect annotations, required pre-condition and fulfilment conditions. By going through these we can conclude that *TrainingCoordinator* is the initiating actor which has the ability to satisfy the pre-condition of conducting a computer based training by triggering the action *LetTrainingSystemScheduleTraining*. *TrainingCoordinator* actor will be placed in the first swimlane of the activity diagram.

Rule -4: Sequence actions by analysing pre/post conditions derived during annotation

The tasks required for the fulfilment of the trigger condition for the course of action will be chosen initially and placed as action within the initiating actor's swimlane in UML activity diagram. After the fulfilment of the pre-condition, the post-condition must be satisfied through the interaction of multiple actors, and the execution of their assigned tasks. These tasks are mapped to activity diagram as actions and placed in the respective swimlanes that represents the controlling actors. The sequencing for actions is a guided task by identifying the required actions and dependencies in order to achieve the operational goal. For the CBT system we will start from the initiating actor that initiates the first action. The initiating action is *LetTrainingSystemScheduleTraining*, so it is placed in the *TrainingCoordinator's* swimlane. After fulfilment of the pre-condition of this action, the post-conditions will be satisfied through the execution of one or more actions with the interaction of other actors. Thus, we get *GenerateTrainingContent* and *MaintainSchedule* actions and so on.

Rule -5: Discover dependencies and represent them in activity diagram

It is very straightforward to discover dependencies among actors from *i** model. We can get the dependencies from SD or SR models. We shall then represent goal, task and resource dependencies as objects in the activity diagram. The actions will specify which objects perform its operation and their states. The actions within a swimlane can be handled by the same objects or multiple objects.

Softgoal dependency in *i** model is considered as a non-functional requirement of the system, which has a positive or negative contribution for achieving, accomplishing a goal, task, resource. For this reason, softgoal dependency will not represent an object.

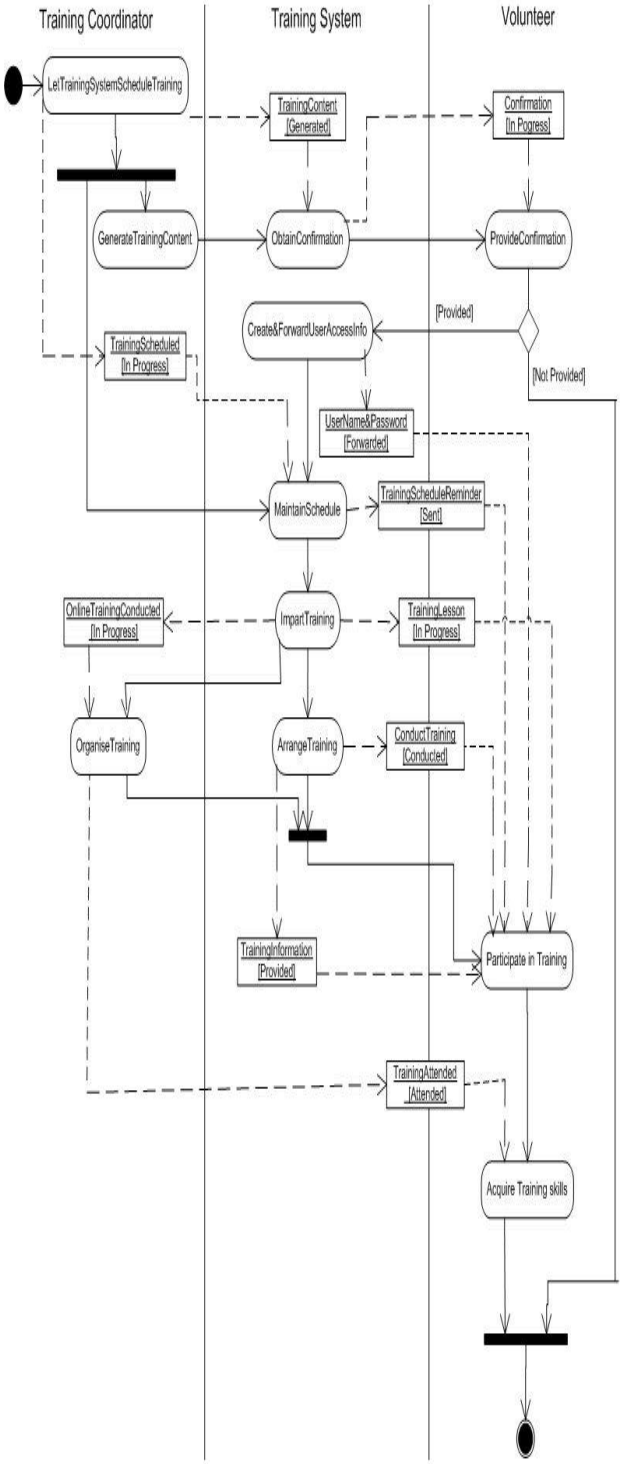

Figure-3: UML Activity Diagram Derived Using the Methodology

We have a total of eleven dependencies in CBT system

including one softgoal dependency *TrainingContentEasyToUse*. All these dependencies except the goal dependency will be represented as objects. For example the resource dependency *TrainingContent* will be the object for *GenerateTrainingContent* and *ObtainConfirmation*. The state of the object *TrainingContent* in this case will be [Generated].

Rule -6: Introduce required actions and object flow links between swimlanes

The final step includes introducing required actions and objects flow links between actions. The actions will be linked according to their sequence and then flow links will be represented among them which will include the objects and their states. In this step we need to consider the decision points of the activities if there is any. Decision points reflect the previous activity state. On each outgoing transition from decision points, we should cover all possibilities.

In this step we represent all the actors and their respective actions with actions and object flow links. The activity *ProvideConfirmation* in *Volunteer* swimlane renders a decision point. It has two guards, [provided] and [not provided], which directs the action links accordingly.

7. Conclusions

In this paper we have presented a consistent methodology to support the mapping of early phase requirement modeling notation *i** into UML activity diagram. The methodology supports the mapping of these two otherwise disparate approaches in a synergistic fashion. We can now analyze the system's behavior and explain the workflow from an initiating point to the terminating point which is otherwise not possible by only looking at the *i** model and activity diagram separately. When proposing the mapping of two otherwise disparate approaches for requirements engineering, we need to maintain consistency between the two approaches. Effect annotations and mapping rules can be viewed as providing semantics to the *i** diagrams while mapping into activity diagram 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 constrains the modeler to map the elements of the *i** model to appropriate activity diagram and ensures that the two models are consistent.

We have not however investigated the possibility of articulating semantic consistency constraints between *i** models and activity diagrams. We have not focused on the reflection of changes in one model into another. There are sixteen categories of changes that may occur to an *i** model [8]. We need to localize these changes to maintain consistency. Further research is needed to relate non-functional requirements (NFRs) with functional requirements of the system [3] [4].

REFERENCE

[1] Booch G., Jacobson I., and Rumbaugh J. (1999) The Unified Modeling Language User Guide, AddisonWesley.

[2] Castro J., Kolp M., and Mylopoulos J. (2002) Towards Requirements Driven Information Systems Engineering: The Tropos Project. Information Systems, Elsevier, Amsterdam, The Netherlands.

[3] Chung, L. (1993) Representing and Using Non - Functional Requirements for Information System Development. A ProcessOriented Approach. PhD Thesis, Graduate Department of Computer Science, Toronto, University of Toronto.

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

[5] Davis, I., Green, P., Rosemann, M., Gallo,S. (2004) Conceptual Modeling – What And Why in Current Practice. In: Lecture notes in Computer Science, Volume 3288, 30 – 42.

[6] Fuxman, A., Pistore, M., Mylopoulos, J., Traverso, P. (2001) Model checking early requirements specifications in Tropos. Proceedings of Fifth IEEE International Symposium on Requirements Engineering, Toronto, Canada, August 27-31, pp. 174 -181.

[7] Fuxman, A. Liu, L. Mylopoulos, J. Pistore, M. Roveri, M. Traverso, P. (2004) "Specifying and analyzing early requirements in Tropos," Requirements Engineering, Springer London, Volume 9, Issue 2, 132 – 150.

[8] Krishna A., Ghose A., and Vilkomir S. (2004) Co-Evolution of Complementary Formal and Informal Requirements, Proceedings of 7th International Workshop on Principles of Software Evolution (IWPSE'04), September 06 - 07, Kyoto, Japan, pp. 159-164.

[9] Larman, Craig (1998). Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design, Prentice Hall. Upper Saddle River, NJ.

[10] Martin Fowler, Kendall Scott: UML Distilled Addison Wesley 2000.

[11] [UML2.0] (2004) – OMG UML Specification v. 2.0, available at http://uml.org/

[12] Vilkomir S., Ghose A. and Krishna A., (2004) Combining agent-oriented conceptual modeling with formal methods, Proceedings of 15th Australian Software Engineering Conference (ASWEC 2004), Melbourne, Australia, IEEE Computer Society, pp. 147-157.

[13] Wang, X., Lesprance, Y. (2001) Agent-Oriented Requirements Engineering Using ConGolog and *i**. Proceedings of 3rd International Bi-Conference Workshop Agent-Oriented Information Systems (AOIS 2001), Berlin, Germany, pp. 59-78.

[14] Yu, E. (1995) Modeling Strategic Relationships for Process Reengineering. PhD Thesis, Graduate Department of Computer Science, University of Toronto, Toronto, Canada, pp. 124.

[15] Yu, E. (1997) Towards Modeling and Reasoning Support for Early Phase Requirements Engineering. Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97), Washington D.C., USA. pp. 226-235.