# HYBRID MODELING USING i* AND AGENTSPEAK(L) AGENTS IN AGENT ORIENTED SOFTWARE ENGINEERING

Aniruddha Dasgupta, Farzad Salim, Aneesh Krishna, Aditya K.Ghose
*Decision Systems Lab, School of IT and Computer Science,University of Wollongong*
*Wollongong, NSW 2522, Australia*

Keywords: Requirements Engineering, agents, i*, AgentSpeak(L).

Abstract: In this paper we use i* which is a semi-formal modelling framework to model agent based applications. We then describe how we execute these models into AgentSpeak(L) agents to form the essential components of a multi-agent system. We show that by making changes to the i* model we can generate different executable multi-agent systems. We also describe reverse mapping rules to see how changes to agents in the multi-agent system gets reflected in i* model. This co-evolution of two models offers a novel approach for configuring and prototyping agent based systems.

## 1 INTRODUCTION

Agent-oriented approaches are becoming popular in software engineering, both as architectural frameworks, and as modeling frameworks for requirements engineering and design. Many modeling techniques tend to address *late-phase* requirements while the vast majority of critical modeling decisions (such as determining the main goals of the system, how the stakeholders depend from each other, and what alternatives exist (Yu, 1995)) are taken in early-phase requirements engineering. The *i** modeling framework (Yu, 1995) is a semiformal notation built on agent-oriented conceptual modeling that is well-suited for answering these questions. AgentSpeak(L) (Rao, 1996) is an agent programming language with logic-based formalism for specifying processes that involves multiple agents. These two formalisms complement each other well, and in this work, we develop a methodology for their combined use in requirements engineering.
We enhance and apply the techniques developed in (Salim et al., 2005) to design a meeting scheduler using *i** modeling (Yu, 1995) framework to produce executable AgentSpeak(L) agents. The *i** framework is used to model different alternatives for the desired system, analyze and decompose the functions of the different actors, and model the dependency relationships between the actors and the rationale behind process design decisions. The AgentSpeak(L) framework is then used to specify the system behavior described informally in the *i** model. Complete AgentSpeak(L) models are executable which can be used to validate the specifications by simulation. We then describe a set of reverse mapping rules by which we can make modifications to the AgentSpeak(L) executable model to get a new set of of *i** model.
The remainder of this article is organized as follows. Section 2 gives an overview of agent based prototyping using *i** and describes how the meeting scheduler is modeled using *i**. Section 3 gives an overview of AgentSpeak(L). Section 4 discusses how *i** and AgentSpeak(L) can be combined by a set of mapping rules to trace a wide range of properties of agent based architecture. Finally, concluding remarks are presented in the last section.

## 2 *i** MODELING FRAMEWORK

The *i** (Yu, 1995) for agent-oriented conceptual modeling was designed primarily for early-phase requirements engineering. An *i** consists of two main modeling components: the Strategic Dependency (SD) Model and the Strategic Rationale (*SR*) Model.
Intentional actors (*SR*) that are the central concept in *i**, represent the intentional properties of an actor such as goals, beliefs, abilities and commitments.
Both *SD* and *SR* diagrams are graphical representations that describe the world in a manner closer to the


Dasgupta A., Salim F., Krishna A. and K.Ghose A. (2006).
HYBRID MODELING USING i* AND AGENTSPEAK(L) AGENTS IN AGENT ORIENTED SOFTWARE ENGINEERING.
In *Proceedings of the Eighth International Conference on Enterprise Information Systems - ISAS*, pages 420-425
DOI: 10.5220/0002453004200425

users perceptions. The *SD* diagram 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 centers is called the *dependum*. The *SD* diagram represents the goals, task, resource, and soft goal 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 do it. In a task-dependency, the *depender* depends on the *dependee* to carry out an activity. 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 (Chung et al., 2005) and is commonly used to represent optimization objectives, preferences or specifications of desirable (but not necessarily essential) states of affairs.

A *SR* diagram represents the internal intentional characteristics of each actor/agent via task decomposition links and means-end links. The task decomposition links provide details on the tasks and the (hierarchically decomposed) sub-tasks to be performed by each actor/agent while the means-end links relate goals to the resources or tasks required to achieve them. The *SR* diagram also provides constructs to model alternate ways to accomplish goals by asking why, how and how else questions.

We shall use the example of a meeting scheduler as described in (Yu, 1997) throughout the rest of this paper to illustrate how the *i** models can be executed. Interested readers may refer to (Yu, 1997) for a detailed overview. The meeting scheduler should try to determine a meeting date and location so that most of the intended participants will participate effectively. The system would find dates and locations that are as convenient as possible. The meeting initiator would ask all potential participants for information about their availability to meet during a date range, based on their personal agendas. This includes an exclusion set dates on which a participant cannot attend the meeting, and a preference set dates preferred by the participant for the meeting. The meeting scheduler comes up with a proposed date. The date must not be one of the exclusion dates, and should ideally belong to as many preference sets as possible. Participants would agree to a meeting date once an acceptable date has been found. The modeling process includes steps as follows

1. Identify actors.
2. Identify goals.
3. Identify Dependency relationships.
4. Conduct means-end and task-decomposition analysis.

Using steps 1 to 3 above, we get the SD diagram as shown in Figure 1. 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 analyst to understand and analyze new or existing organizational and system configurations even if the internal goals and beliefs of individual agents are not known.

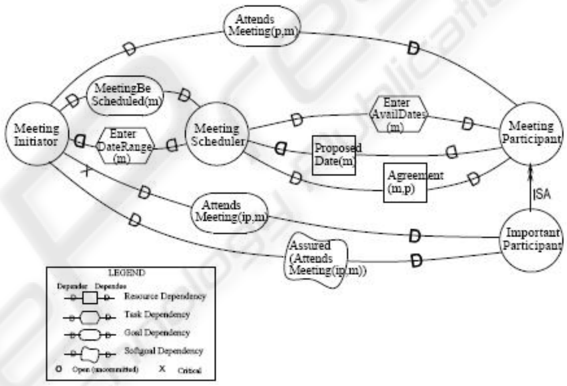


Figure 1: Strategic Dependency Diagram.

To explore the reasons behind such dependency relationships in SD models we need the intentions of the agents that initiate the process. We also define *soft goals* for each agent which are similar to goals except that they do not have clear-cut criteria of satisfaction (Simon, 2005). They commonly express qualitative goals. 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. During this step, goals and tasks are further decomposed into subgoals and subtasks. The output of this step is a SR diagram for each actor. The SR diagram for each of the actors namely *meeting initiator*, *meeting scheduler* and *meeting participant* is shown in Figure 2 using step 4 mentioned earlier.

## 3 AgentSpeak(L)

AgentSpeak(L) is an agent framework/language with explicit representations of beliefs and intentions for agents. This agent programming language

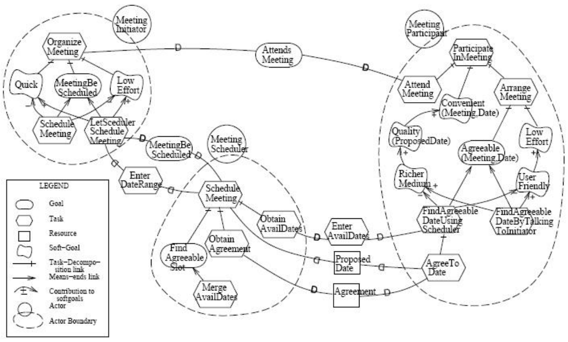


Figure 2: Strategic Rationale Diagram.

was initially introduced in (Rao, 1996). Rao describes an AgentSpeak(L) agent as a set of $\langle E, B, P, I, A, S_E, S_O, S_I \rangle$ where:

- $E$ is a set of events.
- $B$ is a set of base beliefs.
- $P$ is a set of plans.
- $I$ is a set of intentions.
- $A$ is a set of atomic actions.
- $S_E$ selects an event from the set $E$.
- $S_O$ selects a plan from the set $P$.
- $S_I$ selects an intention from the set $I$.

There are two types of goals in AgentSpeak(L). An "achievement goal" (a predicate prefixed with "!"), states that the agent wishes to achieve a state of the world in which the associated predicate is true. A "test goal" (a predicate prefixed with "?"), states that the agent wishes to test if the associated predicate is a true. Events in AgentSpeak(L) might be external or internal. External events represent the changes in the state of the world that should be handled by the agent. On the other hand, internal events are triggered from within the agent as a result of executing a plan. An agent must have pre-designed plans in its plan library to handle the incoming internal or external events. Plans are the central concept to the abilities of an agent. They are means that enable an agent to respond to the changes in its' environment. A plan of an agent is composed of two main parts, *head* and *body*. The *head* is a $pair$ consisting of a triggering event and context. A plan in AgentSpeak(L) is of the form:

$$e : b_1; \ldots; b_n \leftarrow h_1; \ldots; h_n.$$

*e* is a triggering event (*trigger*), $b_1; \ldots; b_n$ are belief literals ($context$), and $h_1; \ldots; h_n$ are goals or actions (*body*). Triggering event is used to identify if the plan is a relevant plan for an given event selected from $E$. Context of a plan consists of beliefs that should hold for that plan to be applicable. Body of a plan is a sequence of sub-goals or actions that should be executed for a plan to be successfully completed. Events, regardless of their types (internal/external), but based on their affects on the agent's belief are divided into two categories:
(1) Events that add a belief/goal is prefixed with "+".
(2) Events that delete a belief/goal from the agent's beliefs is prefixed with "-".
Intentions are formed when an agent commits to a particular set of plans to achieve its goal(s).

## 4 MAPPING *i** MODEL TO AgentSpeak(L) AGENTS

A first step in defining a co-evolution methodology for ***i**** and AgentSpeak(L) is to define a mapping from ***i**** to AgentSpeak(L). We provide the results from the earlier work (Salim et al., 2005) where this mapping was initially defined and full versions of the schemas have been described. The intersected reader may go through (Salim et al., 2005) for a complete overview.
A multi-agent system (MAS) is defined in (Salim et al., 2005) as follows.
*MAS is a pair* $\langle Agents,\ \mathcal{ESA} \rangle$ *where* $Agents = a_1, \cdots, a_n$, *each* $a_i$ *is an AgentSpeak(L) agent and* $\mathcal{ESA}$ *is a specially designated Environment Simulator Agent implemented in AgentSpeak(L).*
$\mathcal{ESA}$ holds the knowledge about the actions that might be performed by actors in SD model and the possible environment transformation after the executions of those actions. The environment agent can verify fulfillment properties (clearly defined in Formal Tropos (Fuxman et al., 2003)), which include conditions such as creation conditions, invariant conditions, and fulfillment conditions of those actions associated with each agent. Every action of each agent has those fulfillment properties. $\mathcal{ESA}$ is used to check whether those actions of all agents in this system satisfy corresponding conditions. While $\mathcal{ESA}$ is an AgentSpeak(L) agent, it must be provided with necessary beliefs as well as the plans. The context of the plans determine the constraints that must hold. Likewise, actions in the body are how to react to the situation.
From the mapping rules, the agents in the MAS are *Meeting Scheduler*, *Meeting Participant* and *Meeting Initiator*. We map the edges and nodes for each agent from the SR diagrams for each actor which defines the *goal*, *task* and *resource* dependencies into AgentSpeak(L) plans. The result of applying these rules are shown in Figures 3, 4 and 5 which depict the AgentSpeak(L) agents. Note that some of the plans that does not have any body does not exist in the actual programs. However, we show them in these figures to avoid the confusion and improve

the clarity of the paper. It is to be noted here that beside the three agents, the $\mathcal{ESA}$ is also supplied by the modeler of the system (not shown here). The $\mathcal{ESA}$ monitors all of the actions/tasks performed by each agent, all of the messages exchanged and all of the beliefs communicated by individual agents for consistency and for constraint violations. When any of these is detected, the $\mathcal{ESA}$ generates a user alert. The softgoals of the actors are translated into the option selection function of AgentSpeak(L) as described in (Salim et al., 2005). By executing the AgentSpeak(L) agents, one can test out the various scenarios whereby given a set of beliefs, whether a given range of dates will be available. Thus the executable specification forms a basis whereby the user can determine the behavior of the system.
Given two goal predicate symbols, $goal, task$, a belief predicate symbol $resource$ and a term $t$:
$-$ $!goal(t)$ is a valid goal iff $t \in N_G$.
$-$ $!task(t)$ is also a valid goal iff $t \in N_T$.
$-$ $resource(t)$ is a valid belief atom iff $t \in N_R$.

Given four action predicate symbols, $RequestAchieve$, $RequestPerform$, $RequestResourse$, $Supply$ and a term $t$:
$-$ $RequestAchieve(t)$ is a valid action iff $t \in N_G$.
$-$ $RequestPerform(t)$ is a valid action iff $t \in N_T$.
$-$ $RequestResource(t)$ is a valid action iff $t \in N_R$.
$-$ $Supply(t)$ is also a valid action iff $t \in N_R$.
$N_G$, $N_T$ and $N_R$ are goal, task and resource node respectively in SR and SD diagrams.

**Actions**
RequestAchieve(AttendMeeting).
RequestAchieve(MeetingBeScheduled).
Perform(EnterDateRange).

**Plans**
+task(OrganizeMeeting): True $< -$
!goal(MeetingBeScheduled),
RequestAchieve(AttendMeeting).

+goal(MeetingBeScheduled): True $< -$
!task(ScheduleMeeting).

+goal(MeetingBeScheduled): True $< -$
!task(LetSchedulerScheduleMeeting).

+task(ScheduleMeeting): True $< -$ .

+task(LetSchedulerScheduleMeeting): True $< -$
RequestAchieve(MeetingBeScheduled),
Perform(EnterDateRange).

Figure 3: AgentSpeak(L) plans for Meeting Initiator Agent.

## 5 CO-EVOLUTION OF *i** AND AgentSpeak(L)

We now propose a hybrid modeling approach from the mapping rules mentioned earlier. This hybrid modeling is composed of *i** model and AgentSpeak(L) agents, that is, when we have an *i** model constructed for a given system, then we can also get the AgentSpeak(L) agents of this system using mapping rules. Our problem representation, as shown in Figures 3, 4 and 5, is an executable specification because it is an operational AgentSpeak(L) programming which can be run in an multi-agent environment like *Jason*(Bordini et al., 2005) which could therefore check the initial *i** model by executing AgentSpeak(L) agents. In this hybrid model, these two basic models, *i** and AgentSpeak(L) agents, might co-evolve. At each stage, the *i** model and AgentSpeak(L) agents are consistent. Using translation steps, they can be translated into each other. This co-evolution process will involve two aspects:

- *reflect the changes of* i* *model on AgentSpeak(L) agents*
- *reflect the changes of AgentSpeak(L) agents on* i* *model*

There are sixteen categories of possible changes that may occur to *i** model. These are the addition and deletion of the following eight elements: Dependencies, Tasks, Goals, Resources, Softgoals, Means-end links, task-decomposition links and Actors. As for our work to reflect the changes of *i** model to AgentSpeak(L) program, we only put emphasis on nodes, goals, tasks, softgoals, dependencies. The changes of those nodes will also bring the changes to the links. We shall consider each of these cases in turn.

- *Addition/deletion of a task to an existing SR model: Addition*: 1) If the new task is a top-level task, add this it into the set of actions, and write corresponding plans if there are subnodes connected to it by task-decomposition links. 2) If the new task is connected to a parent task by task-decomposition link, then add this task to the relevant plan whose head is the parent task. 3) If the new task is connected by means-end link to a goal node which has no other task or goal that connected to it, then add the corresponding plan to the set of plans. 4) If the new task is connected by means-end link to a goal node which has other tasks or goals connected to it and this new task is also jointed with softgoals used as the criteria for means selection, then add the belief of the relationship of task and softgoals and modify the plan for that goal. *Deletion*: Delete all the elements that are relevant to that task. This may

**Actions**
Supply(ProposedDate).
RequestPerform(EnterDateRange).
RequestPerform(EnterAvailDates).
RequestResource(Agreement).

**Plans**
+task(ScheduleMeeting): True $<-$
!goal(FindAgreeableSlot),
!task(ObtainAgreement),
!task(ObtainAvailDates),
Supply(ProposedDate),
RequestPerform(EnterDateRange).

+task(ObtainAvailDates): True $<-$
RequestPerform(EnterAvailDates).

+task(ObtainAgreement):True $<-$
RequestResource(Agreement).

+goal(FindAgreeableSlot): True $<-$
!task(MergeAvailDates).

+task(MergeAvailDates):True $<-$.

Figure 4: AgentSpeak(L) plans for Meeting Scheduler Agent.

**Actions**
Supply(Agreements).
RequestPerform(EnterAvailDates).

**Plans**
+task(ParticipateInMeeting):True $<-$
!task(AttendMeeting), !task(ArrangeMeeting).

+task(AttendMeeting): True $<-$ .

+task(ArrangeMeeting): True $<-$
!goal(AgreeableDate).

+goal(AgreeableDate): True $<-$
!task(FindAgreeableDateByTalkingToInitiator).

+task(FindAgreeableDateUsingScheduler): True
$<-$ RequestPerform(EnterAvailDates).

+task(FindAgreeableDateByTalkingToInitiator):
True $<-$ .

+task(AgreeToDate): True $<-$
Supply(Agreement).

Figure 5: AgentSpeak(L) plans for Meeting Participant Agent.

include deletion of the task and softgoal relationship formula from belief base, deletion of the plans whose head is this task, deletion of the plans whose body has this task only, deletion of this task from a plan which has more than one element in the body part.

- *Addition/deletion of a goal to an existing SR model: Addition*: 1) If the new goal is a top-level goal and there are tasks or goals connected to it by means-ends links then adds a plan to set of plans. 2) If the new goal is connected to a parent task node by task-decomposition link, then add this goal into the body part of the plan whose head is the parent task node. *Deletion*: 1) If this goal is a top level goal and there are some subnodes connected to it - delete the plan whose head is this goal. 2) If this goal is connected to a parent task by task decomposition link, then delete this goal from the body part of that plan whose head is the parent task, and if this goal is the only decomposition element of that task, delete the whole plan.
- *Addition/deletion of a softgoal to an existing SR and SD model: Addition:* Modify the option selection function $S_O$ of the plan by adding this new softgoal as another criterion. *Deletion*: Delete those belief formulas that is relevant to this softgoal and modify the plan by taking out this softgoal criteria.
- *Addition/deletion of a dependency to an existing SR model:* There are three kinds of dependencies in *i** model: *task dependency*, *goal dependency* and *resource dependency*. Changes of a dependency may bring changes to two involved agents. For addition, we need to find out the dependee and depender and which element of them needs this dependency or could provide this dependency. Then for the dependee and depender, just add tasks of the form *RequestResource()/Supply(), RequestPerform()* or *RequestAchieve()* depending on whether it is a resource, action or a goal. *Deletion* of a dependency is just a reverse action to the addition.
- *Addition of an actor to an existing i* diagram*: This will lead to a new agent program for the actor. In the instance of each internal (SR) element for the actor, the steps outlined above are followed. The same applies for any dependencies that this actor might participate in.

We shall now discuss the second area where we are able to localize the impact of changes of AgentSpeak(L) agents to *i** model. Before doing this, we need to specify the translation rules for mapping a AgentSpeak(L) program to an *i** model. This is an opposite process to those translation rules that we have described in the previous section. To reflect the refinement of a AgentSpeak(L) program to *i** model, we give another five informal mapping rules as follows:

- *Addition/deletion of an AgentSpeak(L) agent: Addition*: Add an actor in SD and SR models. *Deletion*: Delete the actor in SD and SR models and also delete all the dependency links connected to it from other actors.
- *Addition/deletion of a goal or task clause in AgentSpeak(L) plan*: *Addition*: Add a goal node or task node with the same name in the actor boundary. A goal or task cannot be added without connecting or being connected with other nodes. All the links associated with the added goal or task node will use mapping rules defined below to be added into *i** model. *Deletion*: Delete corresponding goal or task node from that actor boundary and all the nodes that are subnodes of it. Delete links between them as well.
- *Addition/deletion of a plan: Addition*: If the head of the plan is a goal clause, then add a set of means-end links; If head of the rule is a task clause, then add a set of task-decomposition links. The child nodes are those clauses in the body part of the rule. *Deletion*: Delete a set of means-end links or task-decomposition links from that actor which have the same parent node and that parent node is the head of the deleted rule. After deleting those links, if there is no link connected to the parent node, then delete the parent node from that actor boundary.
- *Addition/deletion of a dependency rule*: *Addition:* If goal, task or resource dependency rules are added into AgentSpeak(L) plans, then corresponding actors $T_o$(Depender) and $T_d$(Dependee) in SD model and SR model needs to be modified to show the reflection of these additions. If $T_d$ has a *RequestAchieve(), RequestPerform()* or a *RequestResource()/Supply()* then these have to be depicted in $T_o$ also showing the dependencies on goal, task and resources. *Deletion*: The reflection to *i** model is the deletion of a goal-dependency or a task-dependency or a resource-dependency from SD model and SR model.
- *Addition/deletion of a softgoal*: *Addition:* If a softgoal is added into the option selection function then corresponding SD model and SR models need to be modified to show the reflection of this addition. *Deletion*: The reflection to *i** model is the deletion of a softgoal from SD model and SR model.

Applying the above set of reverse mapping rules we can see how changes in AgentSpeak(L) programs can be reflected into the *i** model thereby test a wide range of properties of the application.

# 6 CONCLUSION

In this paper we have discussed how the co-evolution of agent technology with *i** model can be used to explore the implication configuring agent based applications. We can analyze the system behavior using real-life example which is otherwise not possible by only looking at the *i** model and AgentSpeak(L) agents separately. The *i** specification of a software system is easily understandable and by mapping it directly into AgentSpeak(L) agents we can get a MAS which is directly executable. We have also defined the reverse mapping rules from AgentSpeak(L) to *i** which also serves as a guide for generating prototypes of complex systems.Using this technique one can specify requirements, define architecture, model behavior as well as do simulation.

This approach makes use of the advantages of *i** for the early-phase of requirement engineering and validates the model by mapping it into an executable specification to see the design result in an emulation program. We are currently working towards enhancing and automating the OME tool as mentioned in (Salim et al., 2005).

# REFERENCES

Bordini, R. H., Hbner, J. F., and et. al. (2005). *Jason: A Java-based AgentSpeak interpreter used with SACI for multi-agent distribution over the net*. http://jason.sourceforge.net/, 0.6 version edition.

Chung, L., Nixon, B. A., Yu, E., and Mylopoulos, J. (2005). Towards executable specifications: Combining i* and agentspeak(l). In *Software Engineering and Knowledge Engineering Conference, Taiwan*.

Fuxman, A., Kazhamiakin, R., Pistore, M., and Roveri, M. (2003). Formal tropos: language and semantics.

Rao, A. (1996). Agentspeak(l): Bdi agents speak out in a logical computable language. In *Agents Breaking Away: Proceedings of the 7th European WS on Modelling Autonomous Agents in a Multi-Agent World*. Springer- Verlag: Heidelberg,Germany.

Salim, F., Chang, C. F., Krishna, A., and Ghose, A. K. (2005). Towards executable specifications: Combining i* and agentspeak(l). In *Software Engineering and Knowledge Engineering Conference, Taiwan*.

Simon, H. (2005). *The Science of the Artificial*. The MIT Press, Cambridge, MA, 2nd edition.

Yu, E. (1995). *Modelling strategic relationships for process reengineering, Phd. Thesis*. University of Toronto, Canada.

Yu, E. (1997). Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE Int. Symp. on Requirements Engineering*.