

2006

Co-evolution of i^* Models and 3APL Agents

A. Krishna

University of Wollongong, krishna@uow.edu.au

Y. Guan

University of Wollongong, yguan@uow.edu.au

Aditya K. Ghose

University of Wollongong, aditya@uow.edu.au

Publication Details

This article was originally published as: Krishna, A, Guan, Y & Ghose, A, Co-evolution of i^* Models and 3APL Agents, Sixth International Conference on Quality Software (QSIC 2006), Beijing, China, October 2006, 117-124. Copyright IEEE 2006.

Co-evolution of i* Models and 3APL Agents

Abstract

Agent-Oriented Conceptual Modelling (AOCM), as exemplified by the i* notation [9], represents an interesting approach to modelling early phase requirements that is particularly effective in capturing organisational contexts, stakeholder intentions and rationale. Our objective in this paper is to define means for executing i* models by translating these into set of interacting agents implemented in the 3APL language. We also propose a hybrid modelling, or co-evolution, approach in which i* models and 3APL agent programs are concurrently maintained and updated, while maintaining some modicum of loose coupling via consistency constraints. This paper explores how these two otherwise disparate approaches might be used in a synergistic fashion for requirement engineering.

Disciplines

Physical Sciences and Mathematics

Publication Details

This article was originally published as: Krishna, A, Guan, Y & Ghose, A, Co-evolution of i* Models and 3APL Agents, Sixth International Conference on Quality Software (QSIC 2006), Beijing, China, October 2006, 117-124. Copyright IEEE 2006.

Co-evolution of i^* Models and 3APL Agents

Aneesh Krishna, Ying Guan, Aditya K. Ghose
Decision Systems Laboratory, School of IT and Computer Science
University of Wollongong, NSW 2522, Australia
{aneesh, yg32, aditya}@uow.edu.au

Abstract

Agent-Oriented Conceptual Modelling (AOCM), as exemplified by the i^ notation [9], represents an interesting approach to modelling early phase requirements that is particularly effective in capturing organisational contexts, stakeholder intentions and rationale. Our objective in this paper is to define means for executing i^* models by translating these into set of interacting agents implemented in the 3APL language. We also propose a hybrid modelling, or co-evolution, approach in which i^* models and 3APL agent programs are concurrently maintained and updated, while maintaining some modicum of loose coupling via consistency constraints. This paper explores how these two otherwise disparate approaches might be used in a synergistic fashion for requirement engineering.*

1. Introduction

Agent-Oriented Conceptual Modelling in notations such as i^* [9, 10] have become very popular in the recent past. Such notations are commonly used to model organisational context and offer high-level social/anthropomorphic abstractions (such as goals, tasks softgoals and dependencies) as modelling constructs. It has been argued that such notations help answer questions such as what goals exist, how key actors depend on each other and what alternatives must be considered [6]. Our contribution in this paper is to define means for *executing* i^* models. This exercise has been motivated by the following observations. First, we seek to utilise the benefits of executable specifications. Second, we wish to view agent-oriented conceptual models and high-level agent programs as jointly constituting a hybrid modelling notation that leverages the complementary representational capabilities of the two approaches. We are interested in leveraging the (well-known) benefits of executable specifications. This approach permits us to analyse early-phase system models by performing rule-/consistency-checking at higher-levels of abstraction. A version of this is check-

ing for FormalTROPOS [2] style conditions on dependencies. Third, we wish to define methodologies to support the co-evolution of models in the two frameworks, such that distinct groups of stakeholders can concurrently model and specify behavior, while maintaining some modicum of loosely-coupled consistency between the models. Finally, we are interested in *compositional, extensible* and easily *maintainable* modelling frameworks. We claim that the combination of high-level modelling in i^* coupled with high-level specifications of functionality using 3APL agent programs offers such a framework.

Understanding the organisational environment as well as the reasoning and rationale underlying requirements, design and process formulation decisions are crucial to model and build effective computing systems [10]. The i^* modelling framework is a semi-formal notation built on agent-oriented conceptual modelling. The central concept in i^* is the intentional actor agent [9]. 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 (optimisation 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: Strategic Dependency (SD) Models and Strategic Rationale (SR) Models (refer to Figure 2). 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. An SR model 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 model also provides constructs to model alternate ways to accomplish goals by asking why, how and how else questions. We shall use the example of online shopping service from [7]

throughout the rest of this paper to illustrate the i^* framework and consequently how these models can be executed. Readers are encouraged to read [7] for the details of this example.

3APL (An Abstract Agent Programming Language) [1, 4, 5] is a programming language for implementing cognitive agents. 3APL is based on a rich notion of agents, that is, agents have a mental state including beliefs and goals. Each agent has a number of basic capabilities. The basic capabilities of an agent are the basic actions an agent can perform. An agent can have a number of practical reasoning rules for planning and revising its current goals. In this paper, we adopt 3APL platform [1] to support our work. Our work is mainly based on 3APL definitions from [1, 4].

Definition 1 A 3APL agent is defined as a tuple $\langle n, B, G, P, A \rangle$, where n is the name of the agent, B is a set of beliefs (Beliefbase), G is a set of goals (Goalbase), P is a set of practical reasoning rules (Rulebase) and A is a set of basic actions (Capabilities).

As described above, each agent is supposed to have beliefs about its mental state. Beliefs of 3APL are represented using first order logic representation language. For example, a belief of a location of an agent can be written as $agent(x_1, y_1)$. The programming constructs for beliefs are defined in [5] as below:

Definition 2 (Programming constructs for beliefs) Given a set of domain variables and functions, the set of domain terms is defined as usual. Let t_1, \dots, t_n be terms referring to domain elements and $Pred$ be a set of domain predicates, then the set of programming constructs for belief formula, BF , is defined as follows:

- $p(t_1, \dots, t_n) \in BF$
- if $\varphi, \psi \in BF$, then $\neg\varphi, \psi \wedge \varphi \in BF$.

For example, an agent is at a certain position, written as $agent(x_1, y_1)$, it has a task to lift a box at certain position, written as $box(x_0, y_0)$. Then we can define the following beliefs.

$agent(x_1, y_1), box(x_0, y_0), NOT\ carrybox(self)$

The basic actions of an 3APL agent has the following construct: $BASIC\ ACTIONS = \{C|C\text{ is a basic action}\}$. Actually, basic actions compose a simple form of goals. In 3APL, goal has two forms, basic and composite. The following definition is programming constructs for both basic goals and composite goals [5].

Definition 3 (Programming constructs for goals) Let BA be a set of basic actions, BF be a set of belief sentences, $\pi, \pi_1, \dots, \pi_n \in GOAL$ and $\varphi \in BF$. Then, the set of programming constructs for 3APL goals (GOAL) can be defined as follows:

- BactionGoal: $BA \in GOAL$.
- PredGoal: $BF \in GOAL$.

- TestGoal: if $\varphi \in BF$, then $\varphi? \in GOAL$.
- SkipGoal: $skip \in GOAL$.
- SequenceGoal: if $\pi_1, \dots, \pi_n \in GOAL$, then $\pi_1; \dots; \pi_n \in GOAL$.
- IfGoal: If φ THEN π_1 ELSE $\pi_2 \in GOAL$.
- WhileGoal: WHILE φ DO $\pi \in GOAL$.

These programming constructs of goals can be used in the body part of a practical reasoning rule and make 3APL more flexible.

In a 3APL agent, P is a set of rules in the form:

$$\pi_h \leftarrow \neg\varphi \mid \pi_b$$

In this formula, π_h and π_b belong to a goal variable set [4], and φ is a belief. When the agent has goal π_h and believes φ then π_h is replaced by π_b .

A set of beliefs, a set of goals and a set of rules of an agent compose the beliefbase, goalbase and rulebase of this agent. For a 3APL agent, Beliefbase is dynamic. It is updated with executing basic actions from capabilities set. Basic Actions are mental actions that an agent can perform, whose basic form is represented as:

$$\{\varphi_1\} Action(X) \{\varphi_2\}$$

where φ_1 is precondition and φ_2 is postconditions, both of them are belief formula, empty is allowed here. $Action(X)$ is action formula. The execution of the mental action will result in the update of beliefbase through replacing preconditions by postconditions. Note that, Capabilities set is not compulsory to an agent, sometimes, an agent does not have a mental action. In addition, beliefs can be generated from the communications between two agents (sent and received). 3APL has a mechanism to support the communications between agents. A message mechanism is defined in [1] to fulfill the communication between agents. The messages themselves have a specific structure, *Receiver/ Sender, Performative* are three compulsory elements in a message. Usually, there are three type of message: $send(Receiver, Performative, Content)$, $sent(Receiver, Performative, Content)$, and $received(Sender, Performative, Content)$. This agent communication mechanism is described in details in [1]. In this paper, we will not elaborate more on the syntax of 3APL, readers who may want more details are directed to [1, 4, 5].

The remainder of this paper is organised in the following manner. Section 2 provides mapping rules that translate i^* Model to 3APL Agents. In Section 3 co-evolution approach involving i^* models and 3APL agent programs is given and section 4 presents related work and some concluding remarks.

2 Executable Specification of i^* framework

We now present a framework of executable specifications for the i^* notation based on our earlier work [3]. We view an i^* model as a pair $\langle SD, SR \rangle$ where SD is a graph denoted by $\langle Actors, Dependencies \rangle$ where *Actors* is a set of nodes (one for each actor) and *Dependencies* is a set of labeled edges. These edges can be of 4 kinds: *goal dependencies* (denoted by $D_G(SD)$), *task dependencies* ($D_T(SD)$), *resource dependencies* ($D_R(SD)$) and *softgoal dependencies* ($D_S(SD)$). Each edge is defined as a triple $\langle T_o, T_d, ID \rangle$, where T_o denotes the *depender*, T_d denotes the *dependee* and ID is the label on the edge that serves as a unique name and includes information to indicate which of the four kinds of dependencies that edge represents. SR is a set of graphs, each of which describes an actor.

We adopt the concept of an environment simulator agent (ESA) defined in [8]. We define MAS is a pair $\langle Agents, ESA \rangle$ where $Agents = \{a_1, \dots, a_n\}$, each a_i is a 3APL agent and ESA is a specially designated Environment Simulator Agent implemented in 3APL which 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)[2], 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. ESA is used to check whether those actions of all agents in this system satisfy corresponding conditions.

Each graph in an SR model is a triple $\langle SR-nodes, SR-edges, ActorID \rangle$. The *SR-nodes* consist of a set of goal nodes (denoted by N_G), a set of task nodes (N_T), a set of resource nodes (N_R) and a set of softgoal nodes (N_S). *SR-edges* can be of 3 kinds: *means-ends* links (denoted by the set *MELinks*), *task-decomposition* link (denoted by the set *TDLinks*) and *softgoal contribution* link (set *SCLinks*). Each *MELink* and *TDLink* is represented as a pair, where the first element is the parent node and the second element is the child node. A *SCLink* is represented as a triple $\langle s, m, c \rangle$, where the first element is the parent node, the second element is the child node and the third element is the *softgoal contribution* which can be *positive* or *negative*.

Any MAS Agents, ESA obtained from an i^* model $m = \langle SD, SR \rangle$, where $SD = \langle Actors, Dependencies \rangle$ and SR is a set of triples of the form $\langle SR-nodes, SR-edges, ActorID \rangle$ (we assume that a such a triple exists for each actor in Actors) with $SR-nodes = N_G \cup N_T \cup N_R \cup N_S$ and $SR-edges = MELinks \cup TDLinks \cup SCLinks$ must satisfy the following conditions:

1. For all $a \in Actors$, there exists an agent in *Agents* with

the same name.

2. For all $a \in Actors$ and for each node $n \in N_G \cup N_T$ in the SR model for that actor, the agent $\langle a, B, G, P, A \rangle \in Agents$ corresponding to this actor must satisfy the property that $goal(n) \in G$.
3. For all $a \in Actors$ and for each $p \in N_G$ (parent node) for which a link $\langle p, c \rangle \in MELink$ exists in the SR model for that actor, with $c \in N_T$ (children node), the corresponding agent $\langle a, B, G, P, A \rangle \in Agents$ must satisfy the property that $goal(p) < -\varphi \mid SeqComp(T) \in P$. Here $T = \{c_1, \dots, c_n\}$, given that $\langle p, c_1 \rangle, \dots, \langle p, c_n \rangle$ are all the task decomposition links that share the same parent p . *SeqComp(T)* is an operation that generates the body of the procedural reasoning rule referred to above by sequentially composing the goal or task children identified in each of the means-ends links with the same parent p . The i^* model in itself does not provide any information on what this sequence should be. This needs to be provided by the analyst or, by default, obtained from a left-to-right reading of the means-ends links for the same parent in an SR diagram.
4. For all $a \in Actors$ and for each $p \in N_T$ for which a link $\langle p, c \rangle \in TDLink$ exists in the SR model for that actor (where $c \in (N_T \cup N_G)$), the corresponding agent $\langle a, B, G, P, A \rangle \in Agents$ must satisfy the property that $goal(p) < -\varphi \mid SeqComp(T) \in P$. Here $T = \{c_1, \dots, c_n\}$, given that $\langle p, c_1 \rangle, \dots, \langle p, c_n \rangle$ are all the task decomposition links that share the same parent p . *SeqComp(T)* is as defined in rule 3.

Note that, in the rules defined above, the execution orders of sub-tasks within the *Task-decomposition* links are from left to right as default. Belief formulas of each practical reasoning rule cannot be generated completely automatically; instead, those beliefs are specified by designers analyst.

5. For all $a \in Actors$ and for each triple $\langle s, m, c \rangle \in SCLinks$ in the SR model for that actor, the corresponding agent $\langle a, B, G, P, A \rangle \in Agents$ must satisfy the property that $belief(m, s, c) \in B$. We do not describe how beliefs about *softgoal contributions* are used in agent programs for brevity – we will flag however that they can plan a critical role in selecting amongst practical reasoning rules.
6. For all dependencies $\langle T_o, T_d, ID \rangle \in SD$, there exist agents $\langle T_o, B_o, G_o, P_o, A_o \rangle, \langle T_d, B_d, G_d, P_d, A_d \rangle \in Agents$, such that if $\langle T_o, T_d, ID \rangle \in D_G(SD)$, then $goal(ID) \in G_o, goal(ID) < -\varphi \mid BEGIN send(T_d, request, requestAchieve(ID)); send(ESA, inform, believe(\varphi))$

$END \in P_o$,
 $received(T_o, request, requestAchieve(ID))$ |
BEGIN
 $Achieve(ID)$;
 $send(ESA, inform, believe(Achieved(ID)))$
 $END \in P_d$.

Similarly, if $\langle T_o, T_d, ID \rangle \in D_T(SD)$, then $task(ID) \in G_o$,
 $Task(ID) < -\varphi$ |
BEGIN
 $send(T_d, request, requestPerform(ID))$;
 $send(ESA, inform, believe(\varphi))$
 $END \in P_o$
 $received(T_o, request, requestPerform(ID))$ |
BEGIN
 $Perform(ID)$;
 $send(ESA, inform, believe(Performed(ID)))$
 $END \in P_d$.

Similarly, if $\langle T_o, T_d, ID \rangle \in D_R(SD)$ then
 $Request(ID) < -\varphi$ |
BEGIN
 $send(T_d, request, requestProvide(ID))$;
 $send(ESA, inform, believe(\varphi))$
 $END \in P_o$.
 $received(T_o, request, requestProvide(ID))$ |
BEGIN
 $Offer(ID)$;
 $send(ESA, inform, believe(Offered(ID)))$
 $END \in P_d$.

Notice that these rules requires that the creation conditions be communicated by the *dependor* agent to the *ESA* agent. The *ESA* monitors all of the actions/tasks performed by each agent, all of the messages exchanged and all of the beliefs (usually creation conditions for dependencies) communicated by individual agents for consistency and for constraint violations (e.g. the *FormalTROPOS-style* conditions associated with dependencies). When any of these is detected, the *ESA* generates a user *alert*.

3 Co-evolution of *i** and 3APL Agents

The main contribution of this paper is presented in this section. This hybrid modelling approach makes use of *i** model and 3APL agents. 3APL agents can be derived from the *i** models by using the mapping rules already provided. This approach could be employed to check the initial *i** model by executing 3APL programs. In the hybrid model, the *i** models and 3APL agents co-evolve. Figure 1 shows a co-evolution process of *i** Models and 3APL Agents. At each stage, the *i** model and 3APL agents are consistent, that is, by using translation steps, they can be translated into

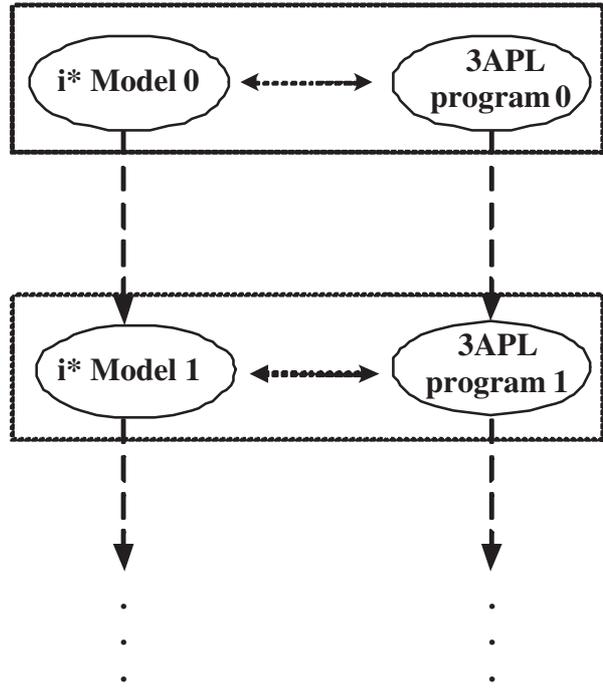


Figure 1. Co-evolution of *i Models and 3APL Agents**

each other. This co-evolution process will involve two aspects, one is to reflect the changes of *i** model on 3APL agents, the other is to reflect the changes of 3APL agents on *i** model. A similar work has been done in [6].

This co-evolution process will involve two aspects, one is to reflect the changes of *i** model on 3APL agents, and the other is to reflect the changes in 3APL agents on *i** model. Let us consider the first aspect. In [6], the authors have listed sixteen categories of possible changes that may occur to the *i** model. These are the additions and deletions, respectively, of the following eight elements: Dependencies, Tasks, Goals, Resources, Softgoals, Means-end links, task-decomposition links and Actors. As far our work is concerned, we shall pay more emphasis on the nodes, goals, tasks, softgoals and dependencies. The changes to those nodes will also bring the changes to the links. We shall consider each of these cases in detail.

- Addition/deletion of a task to an existing SR model:
Addition: 1) If the new task is a top-level task, add this into Goalbase, and write corresponding PR-Rule in the Rulebase provided there are subnode connected to it by a task-decomposition link. 2) If the new task is connected to a parent task by task-decomposition link, then add this task to the relevant PR-Rule whose head

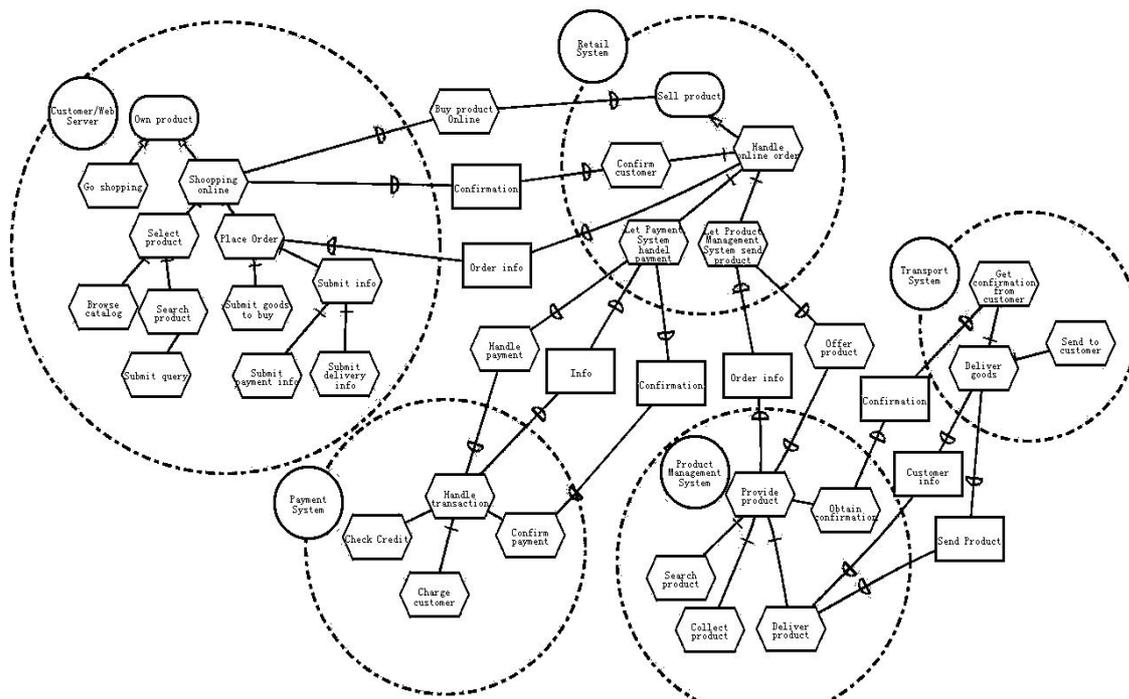


Figure 2. Strategic Rationale Model of online shopping service

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 is connected to it, then add the corresponding PR-Rule to the Rulebase. 4) If the new task is connected by means end link to a goal node which has other task or goals connected to it, and this new task is also connected with the softgoals used as the criteria for means selection, then add the belief of the relationship of the task and softgoals and modify the PR-Rule of that goal.

Deletion: To delete a task from an existing SR model is relatively simpler issue, just to delete all the elements that are relevant to that task, this may include: deletion of the task and softgoal relationship formula from belief base, deletion of the PR-Rules whose head is this task, deletion of the PR-Rules whose only body is this task, deletion of this task part from a PR-Rule which have more than one elements in the body part and this need the modification of that PR-Rule.

- Addition/deletion of a goal to an existing SR model:
Addition: This goal needs to be added into the Goalbase, and then: 1) if the new goal is a top-level goal and there are tasks or goals connected to it by means-ends links then add a PR-Rule to the Rulebase. 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 PR-Rule whose head is the parent task node.

Deletion: First, delete this goal from the Goalbase, and then: 1) if this goal is a top level goal and there are some subnodes connected to it; delete the PR-Rule whose head is this goal. 2) If this goal is connected to a parent task by a task decomposition link, then delete this goal from the body part of that PR-Rule whose head is the parent task, and if this goal is the only decomposition element of that task, delete the whole PR-Rule.

- Addition/deletion of a softgoal to an existing SR model:
Addition: Add belief formulas to represent the relationship between this softgoal and those tasks that are connected to it.
Deletion is a reverse operation to addition. Only delete those belief formulas that are relevant to this softgoal .
- 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. Addition or deletion of a dependency may affect the two involved agents.

Addition: Firstly, we need to find out the dependee and depender and which element of them needs this dependency or could provide this dependency. Then use *rule 6* to add corresponding rules in the Rulebase of the agent of dependee and depender respectively.

Deletion of a dependency is just a reverse action to the addition step.

- Addition of an actor to an existing i^* diagram will lead to the following four steps: A new agent program for the actor is created. In the instance of each internal (SR) element for the actor, the steps outlined above are followed. The same applies for any dependencies between the selected actor and other actors.

We shall now discuss the second area where we are able to localise the impact of changes of 3APL agents to i^* model. Before doing this, we need to specify the translation rules for mapping a 3APL program to an i^* model. This is an reverse process to those translation rules that we have already defined in the previous section. To reflect the refinement of a 3APL program to i^* model, we shall provide further six informal mapping rules as described below:

- Addition/deletion of an agent to an existing SR model:
Addition: Add an actor in SD and SR models. Rulebase and Goalbase of this actor can use the following reflection steps to be added into i^* model.

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 Goalbase:
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 following reflection steps to be added into the i^* model.

Deletion: Delete a goal or task clause from Goalbase, then delete corresponding goal or task node from that actor boundary. Also delete all the nodes that are subnodes of it. Delete links between them as well.

- Addition/deletion of a rule:
Addition: If this rule is in the form of those defined in *rule 3* and *rule 4* in the previous section, and head of the rule 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 clauses in the body part of the rule.

Deletion: If the deleted rule is in the form of those defined in *rule 3* and *rule 4* in the previous

section, then delete a set of means-end links or task-decomposition links from that actor which has the same parent node and that parent node is the head of the deleted rule. After deleting the links, if there is no link connected to the parent node, then delete the parent node from that actor boundary.

- Addition/deletion of a belief of softgoal:
Addition: If a belief clause is added in to the Beliefbase of an agent with the form: Belief(m, s, c), and if m and s already exist, then add a softgoal-contribution link between these two nodes. If any of these two node does not exist, then add node(s) and softgoal-contribution links. If c is *positive*, the type of added softgoal-contribution link is positive contribution, otherwise, it is a negative contribution.

Deletion: If a belief belief(m, s, c) is deleted from Beliefbase, then delete the softgoal-contribution link from the SR model of that actor. After deletion, if there is no link connected to m or c , then delete m node or c node as well.

- Addition/deletion of a dependency rule:
Addition: If the following *rule_a* is added into the Rulebase of an agent, then *rule_b* must be added into the agent named T_d at the same time. The reflection of these two rules result in the addition of a goal-dependency between agent T_o (Depender) and T_d (Dependee) in the SD model and SR model.

rule_a : $goal(ID) < -\varphi$ |
BEGIN
send($T_d, request, requestAchieve(ID)$);
send($ESA, inform, believe(\varphi)$)
END $\in P_o$,
rule_b : $< -received(T_o, request, requestAchieve(ID))$ |
BEGIN
Achieve(ID);
send($ESA, inform, believe(Achieved(ID))$)
END $\in P_d$.

Similarly, If the following *rule_c* is added into the Rulebase of an agent, then *rule_d* must be added into the agent named T_d at the same time. The reflection of these two rules leads to the addition of a task-dependency between agent T_o (Depender) and T_d (Dependee) in the SD and SR models.:

rule_c : $Task(ID) < -\varphi$ |
BEGIN
send($T_d, request, requestPerform(ID)$);
send($ESA, inform, believe(\varphi)$)
END $\in P_o$,
rule_d : $< -received(T_o, request, requestPerform(ID))$ |

```

BEGIN
Perform(ID);
send(ESA, inform, believe(Performed(ID)))
END ∈ Pd.

```

Similarly, If the following $rule_e$ is added into the Rulebase of an agent, then $rule_f$ must be added into the agent named T_d at the same time. The reflection of these two rules leads to the addition of a resource-dependency between agent T_o (Depender) and T_d (Dependee) in the SD and SR models.

```

rulee : Request(ID) < -φ |
BEGIN
send(Td, request, requestProvide(ID));
send(ESA, inform, believe(φ))
END ∈ Po,

```

```

rulef : < -received(To, request,
requestProvide(ID)) |
BEGIN
Offer(ID);
send(ESA, inform, believe(Offered(ID)))
END ∈ Pd.

```

Deletion: If a rule in the form of $rule_a$ or $rule_c$ or $rule_e$ is deleted from the Rulebase of an agent. Correspondingly, $rule_b$ or $rule_d$ or $rule_f$ must be deleted from the Rulebase of the agent named T_d at the same time. The reflection of this deletion to i^* model is the deletion of a goal-dependency or a task-dependency or a resource-dependency from the SD and SR models.

One good application of executable specification for i^* framework is an agent-based prototyping for service-oriented architectures. We shall illustrate how the i^* model of *online shopping system* [7] can be mapped into 3APL agents. For the sake of brevity, we shall only provide one example for each mapping rule here.

1. *rule 1:*

In the *Online Shopping System*, *Retail system* is an actor in SR Model, therefore, there is an agent named “Retail System” in this 3APL agents system.

2. *rule 2:*

Goal *Sell product* and task *Handle Online Order* are in the boundary of actor *Retail System*, according to step 2, *SellProduct()* and *HandelOnlineOrder()* are in the goalbase of agent *RetailSystem*.

3. *rule 3:*

In the SR diagram of actor *RetailSystem* of Figure 2, task *Handle Online Order* and goal *Sell Product* are connected by a *means-end link*, therefore, rule *SellProduct()* < - φ | *HandelOnlineOrder()* can be added into the Rulebase of agent *RetailSystem*. Belief

fomula φ and parameters of goal and task can be specified according to the real case.

4. *rule 4:*

Task *Handle Online Order* is a parent task node. This task is further decomposed into three sub-tasks: *Confirm Customer*, *Let Payment System Handle Payment* and *Let Product Management System Send Product*. Using the above rule, will lead to:

```

HandleOnlineOrder() < - φ |
BEGIN
letpaymentsystemhandelpayment();
confirmcustomer();
letproductmanagementsystemsendsendproduct()
END.

```

5. *rule 5:*

There are two ways to achieve goal *Own Product* for an actor, one is *Go Shopping*, the other is *Shopping Online*. On the assumption that task *GoShopping* has positive contribution to softgoals *low effort*, *convenient* and *time saving* while task *ShoppingOnline* has positive effects on those three softgoals.

The following beliefs are in the beliefbase of agent *Customer*.

```

belief(OwnProduct, GoShopping, timesavingnegative).
belief(OwnProduct, GoShopping, loweffortnegative).
belief(OwnProduct, GoShopping, convenientnegative).
belief(OwnProduct, ShoppingOnline, timesavingpositive).
belief(OwnProduct, ShoppingOnline, loweffortpositive).
belief(OwnProduct, ShoppingOnline, convenientpositive).

```

6. *rule 6:*

We shall select one task-dependency and one resource-dependency related to agent *RetailSystem* in order to illustrate *rule 6*. Actor *Customer* depends on actor *RetailSystem* to perform task *Buy Product Online* and to provide *Confirmation* of buying. According to *rule 6*, for agent *Customer*, *BuyProductOnline()* is in the Goalbase. Rules shown below are in its Rulebase:

```

Request(confirmation) < -
product(P) AND needconfirmation(P)|
BEGIN
Offer(confirmation);
send(ESA, inform, believe(needconfirmation))
END

```

```

Task(BuyProductOnline) <-
needtobuyproductonline |
BEGIN
send(retailsystem, request,
requestPerform(BuyProductOnline));
send(ESA, inform, believe( $\varphi$ ))
END are in Rulebase.

```

For agent *RetailSystem*, two rules are generated for these two dependencies relationships.

```

received(customer, request,
requestProvide(confirmation)) |
BEGIN
send(customer, request, offer(confirmation));
send(ESA, inform, believe(Offered(confirmation))
END

received(customer, request, requestPerform (BuyPro-
ductOnline)) |
BEGIN
Perform(BuyProductOnline);
send(ESA, inform,
believe(Performed(BuyProductOnline))
END

```

3.1. Related Work and Conclusions

Some related work is reported in the literature to achieve similar objective. In [8], an executable specification approach was proposed by combining *i** and AgentSpeak (L). The advantages of using 3APL over AgentSpeak(L) are stated in [8]. 3APL uses the notion of goal rather than the notions of event and intention and it has a wider range of rules which enable agents to modify, revise, skip or drop goals when there are failures or other instances. In [8], mapping rules are suggested to run through the output from the *i** Organization Modelling Environment (OME), which specifies the whole executable specification automatically from *i** to agent programming language.

In this paper we have suggested an approach to executing *i** models by translating these into a set of interacting agents (services) implemented in the 3APL language. 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. In addition, we have proposed a hybrid modelling, or co-evolution, approach in which *i** models and 3APL agent programs are concurrently maintained and updated, while retaining some modicum of loose consistency between the two.

Currently, we only have the mapping rules that can be used to translate *i** diagrams into 3APL agents manually. We use OME (Organization Modelling Environment) to develop *i** models and use 3APL platform to run 3APL agent

programs. There is a gap between OME and 3APL platform. In future work, we plan to develop a program which can convert .tel (OME graphic files) files into .3apl (3APL program) files. Furthermore, 3apl platform provide a message log which we can extend to add rule checking, dependency checking and fulfillment properties checking by validating these messages that are passed between agents.

References

- [1] Dastani, M. 3APL Platform User Guide, Utrecht University, 2004
- [2] Fuxman, A., Liu, L., Pistore, M., Roveri, M., Mylopoulos, J. Specifying and Analyzing Early Requirements in Tropos, Requirements Engineering Journal, 2004, 9(2), pp. 132-150
- [3] Guan, Y., Ghose, A. K. Executable specifications for agent-oriented conceptual modelling. Proceedings of the IEEE/WIC 2005 International Conference on Intelligent Agent Technology, France, 2005, pp. 475-478
- [4] Hindriks, K. V., De Boer, F. S., Van der, H. W., Meyer, J. Agent programming in 3APL. Autonomous Agents and Multi-Agent Systems, 1999, 2(4), pp. 357- 401
- [5] Hoeve, E. 3APL Platform, Master's thesis Computer Science, Utrecht University, 2003
- [6] Krishna, A., Ghose, A. K., Vilkomir, S. Co-evolution of Complementary Formal and Informal Requirements. Proceedings of 7th International Workshop on Principles of Software Evolution, Kyoto, Japan, September, 2004, IEEE Computer Society Press, pp. 159-164
- [7] Lou, D., Mylopoulos, J. Designing Web Services with Tropos, Proceedings of International Conference on Web services, 2004
- [8] Salim, F., Chang, C., Krishna, A., Ghose, A. K. Towards executable specifications: Combining *i** and AgentSpeak (L). Proceedings of 17th International Conference on Software Engineering and Knowledge Engineering, Taipei, Taiwan, July, 2005, pp. 739-742
- [9] Yu, E. Modelling Strategic Relationships for Process Reengineering. PhD Thesis, Graduate Department of Computer Science, University of Toronto, Toronto, Canada, 1995, pp. 124
- [10] Yu, E., Mylopoulos, J. Understanding "Why" in Software Process Modelling, Analysis, and Design. Proceedings of 16th International Conference on Software Engineering, Sorrento, Italy, May 16-21, 1994, pp. 159-168