

2005

Executable specifications for agent oriented conceptual modelling

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: Guan Y & Ghose, AK, Executable specifications for agent oriented conceptual modelling, IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 19-22 September 2005, 475-478. Copyright IEEE 2005.

Executable specifications for agent oriented conceptual modelling

Abstract

Agent-oriented conceptual modelling (AoCM) notations such as i^* have received considerable recent attention as a useful approach to early-phase requirements engineering. AoCM notations are useful in modeling organizational context and in offering high-level anthropomorphic abstractions as modeling constructs. AoCM notations such as i^* help answer questions such as what goals exist, how key actors depend on each other and what alternatives must be considered. In this paper, we suggest an approach to executing i^* models by translating these into set of interacting agents implemented in the 3APL language. In addition, we suggest a hybrid modeling, 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. This allows us to benefit from the complementary representational capabilities of the two frameworks.

Disciplines

Physical Sciences and Mathematics

Publication Details

This article was originally published as: Guan Y & Ghose, AK, Executable specifications for agent oriented conceptual modelling, IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 19-22 September 2005, 475-478. Copyright IEEE 2005.

Executable Specifications for Agent Oriented Conceptual Modelling

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

Abstract

Agent-oriented conceptual modelling (AoCM) notations such as i^ have received considerable recent attention as a useful approach to early-phase requirements engineering. AoCM notations are useful in modeling organizational context and in offering high-level anthropomorphic abstractions as modeling constructs. AoCM notations such as i^* help answer questions such as what goals exist, how key actors depend on each other and what alternatives must be considered. In this paper, we suggest an approach to executing i^* models by translating these into set of interacting agents implemented in the 3APL language. In addition, we suggest a hybrid modeling, 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. This allows us to benefit from the complementary representational capabilities of the two frameworks.*

1. Introduction

Agent-oriented conceptual modeling in notations such as the i^* framework [8] have gained considerable currency in the recent past. Such notations model organizational context and offer high-level social/anthropomorphic abstractions (such as goals, tasks, softgoals and dependencies) as modeling 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. Our objective in this paper is to define means for *executing* i^* models. This exercise has been motivated by the following observations. First, we seek to exploit the benefits of executable specifications. Second, we wish to view agent-oriented conceptual models and high-level agent programs as jointly constituting a hybrid modeling notation that leverages the complementary representational capabilities of the two approaches. 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, ex-*

tensible and easily *maintainable* modeling frameworks. We claim that the combination of high-level modeling in i^* coupled with high-level specifications of functionality using 3APL agent programs offers such a framework.

This research has been conducted concurrently (and within the same group) with a project to develop means for executing i^* models via sets of AgentSpeak agents [7]. While the starting points and motivations for both exercises are similar, the eventual mapping of models to multi-agent systems is defined in very different ways. A detailed comparison of the two approaches (which reveals many interesting differences due to the subtly different capabilities of 3APL and AgentSpeak [6]) is omitted here for brevity.

i^* modeling framework

The i^* framework [8] is an informal diagram-based language designed for early-phase requirements engineering. There are two kinds of graphical models in an i^* framework: Strategic Dependency model (SD) and Strategic Rationale model (SR). Strategic Dependency model captures the social context of the system. It consists of a set of nodes that represent actors (an agent, position, or role) and a set of links that represent social dependencies. The social dependency relationship between two actors indicates an actor may depend on another to achieve a goal, perform a task, provide a resource or achieve a softgoal. The depending actor is known as *dependor*, while the actor depended upon is known as *dependee*. The object around the dependency relationship centers is called *dependum*. The Strategic Rationale (SR) model of i^* framework is a set of graph which represents the internal intentional characteristics of each actors. It consists of four types of nodes, goal, task, resource and softgoal, and two main types of links, means-ends link and task decomposition link. For more details about the functionality of i^* model, please refer to [8].

3APL (An Abstract Agent Programming Language)

3APL (An Abstract Agent Programming Language) [3][1][4] 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. Finally, 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 [3][1].

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).

In [4], a set of programming constructs for goals are defined, namely *BactionGoal*, *PreGoal*, *TestGoal*, *SkipGoal*, *SequenceGoal*, *IfGoal*, *WhileGoal* and *JavaGoal*, which can be used in the body part of a practical reasoning rule and make 3APL more flexible.

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

$$\pi_h \leftarrow \varphi \mid \pi_b.$$

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 .

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:

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

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. 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 types 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 [3][1][4].

2. Executable Specification

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* (denoted by $D_T(SD)$), *resource dependencies* (denoted by $D_R(SD)$) and *softgoal dependencies* (denoted by $D_S(SD)$). Each edge is defined as a triple

$\langle T_o, T_d, ID \rangle$, where T_o denotes the *dependor*, T_d denotes the *dependum* 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 [7].

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 $\langle SR\text{-nodes}, SR\text{-edges}, ActorID \rangle$. The *SR-nodes* consist of a set of goal nodes (denoted by N_G), a set of task nodes (denoted by N_T), a set of resource nodes (denoted by N_R) and a set of softgoal nodes (denoted by 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 (denoted by the 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. An *SCLink* is represented as a triple, 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 $\langle Agents, ESA \rangle$ 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 $\langle SR\text{-nodes}, SR\text{-edges}, ActorID \rangle$ (we assume that a such a triple exists for each actor in *Actors*) with $SR\text{-nodes} = N_G \cup N_T \cup N_R \cup N_S$ and $SR\text{-edges} = MELinks \cup TDLinks \cup SCLinks$ 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 $goal(p) \leftarrow \varphi \mid SeqComp(T) \in P$. 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 $goal(p) \leftarrow \varphi \mid SeqComp(T) \in P$. 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.

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 play a critical role in selecting amongst procedural 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 $goal(ID) \in G_o$, $goal(ID) \leftarrow \varphi \mid BEGIN send(T_d, request, requestAchieve(ID)); send(ESA, inform, believe(\varphi)) END \in P_o$, $received(T_o, request, requestAchieve(ID)) \mid BEGIN Achieve(ID); send(ESA, inform, believe(Achieved(ID))) END \in P_d$. Here φ denotes the creation condition of the dependency ID . Similarly, if $\langle T_o, T_d, ID \rangle \in D_T(SD)$, $task(ID) \in G_o$, $task(ID) \leftarrow \varphi \mid BEGIN send(T_d, request, requestPerform(ID)); send(ESA, inform, believe(\varphi)) END \in P_o$, $received(T_o, request, requestPerform(ID)) \mid BEGIN Perform(ID); send(ESA, inform, believe(Performed(ID))) END \in P_d$. Similarly, if $\langle T_o, T_d, ID \rangle \in D_R(SD)$ then $Request(ID) \leftarrow \varphi \mid BEGIN send(T_d, request, requestProvide(ID)); send(ESA, inform, believe(\varphi)) END \in P_o$, $received(T_o, request, requestProvide(ID)) \mid BEGIN send(T_o, request, offer(ID)); send(ESA, inform, believe(Offered(ID))) END \in P_d$. Notice that these rules require that the creation conditions are communicated by the depender 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. Example

This section briefly illustrates how those mapping rules defined in section 2 can be applied through the example of a meeting scheduler system from [8]. Due to the space limitation, reader please refer to [8] for the details of this example.

In this instance, we have three actors, *meetinginitiator*, *meetingscheduler*, *meetingparticipant* and *ESA*. Meetinginitiator wants a meeting, named *DSLmeeting*, to be scheduled. MeetingScheduler will be chosen to schedule the meeting by using softgoals *loweffort* and *quick* as criteria. We only give one example for each mapping rule here.

As for actor *meetinginitiator*, *meeting(dslmeeting)*, *scheduler(meetingscheduler)*, *participant(meetingparticipant)* and *requirementforschedulingmeeting(dslmeeting)* are initially in the beliefbase. And goal *MeetingBeScheduled(dslmeeting)* is in its Goalbase. The task node, *OrganizeMeeting*, has one sub-goal, *MeetingBeScheduled* and a goal dependency *AttendsMeeting*, which show that *meetinginitiator* need to depend on *meetingparticipant* to achieve this goal. So we can use rule 7 to generate the following rules.

OrganizeMeeting() \leftarrow *meeting(dslmeeting)* AND *requirementforschedulingmeeting(M)* |

BEGIN MeetingBeScheduled(); AttendsMeeting()
END

AttendsMeeting() \leftarrow *meeting(M)* AND *participant(P)* AND *requireparticipanttoattentmeeting(M,P)* |

BEGIN send(P, query, attendsmeeting(M));
send(esa, inform, requireparticipanttoattentmeeting(M,P))
END

Goal *MeetingBeScheduled* has two tasks connected with it by means-end links. To achieve this goal, we need to select from two alternative ways, *ScheduleMeeting* and *LetSchedulerScheduleMeeting*. Those softgoals which are using as selection criteria are *quick* and *loweffort*.
schedulemeeting(quick, negative).
schedulemeeting(loweffort, negative).
letschedulerschedulemeeting(quick, positive).
letschedulerschedulemeeting(loweffort, positive).

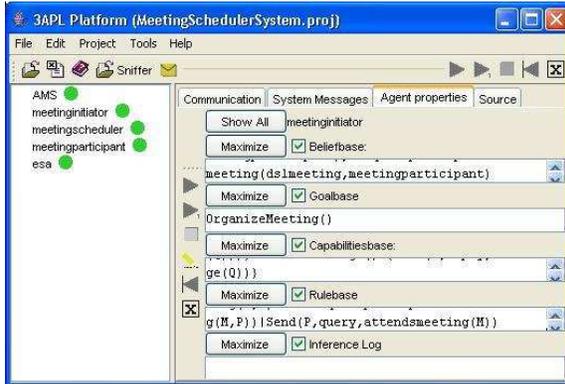


Figure 1. 3APL program for Meeting-scheduler System

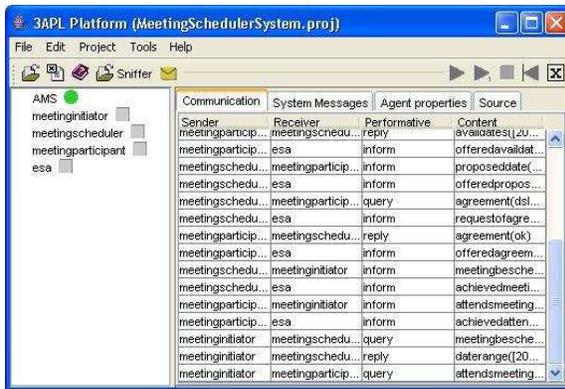


Figure 2. Communication messages of four actors

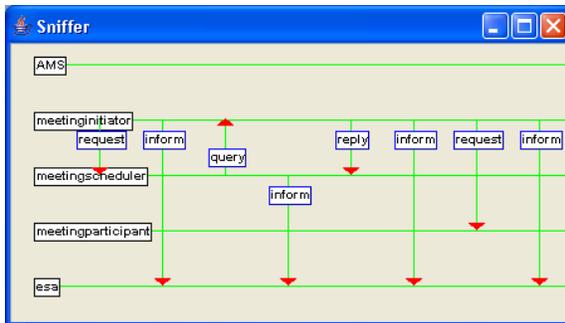


Figure 3. Communication flow of four actors

Similarly, all other part of this SR model can be mapped into 3APL. Figure 1, 2 and 3 show the running result of this meeting scheduler multiagent system, specifically, figure 2 and 3 are the communication among three actors.

4. Related Work and Conclusions

Some related work has been done for similar objective. Authors of [7] propose executable specification by combining i^* and AgentSpeak(L). The advantages of using 3APL over AgentSpeak(L) stated in [7] are 3APL using notion of goal rather than notions of event and intention and it has a larger range of rules which enable agents to modify, revise, skip or drop of goals when there are failure or other instance. In [7], the authors also suggest to apply their mapping rules on ACK through the output from the i^* Organization Modeling Tool (OME) which might make the whole executable specification automatically from i^* to agent programming. This is what our work lacks at this stage and remains as future research.

Our proposal in this paper is that the i^* modeling framework can be executable after mapping into a set of interacting agents implemented in the 3APL language. This approach makes uses 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. Furthermore, we also proposed a hybrid modeling approach in which models are composed of i^* model and 3APL agents. How to co-evolve i^* model and 3APL agents remains for future works.

REFERENCES

- [1] Mehdi Dastani. 3APL Platform User Guide November 16, , Utrecht University, 2004
- [2] . Castro, M. Kolp and . Mylopoulos. Towards Requirements-Driven Information Systems Engineering: The Tropos Project , in *Information Systems (27)*, Elsevier, Amsterdam, The Netherlands, 2002
- [3] K. V. Hindriks, F. S. De oer, Hoek Wiebe van der, and . c Meyer. Agent programming in 3APL. *Autonomous Agents & Multi-Agent Systems*, 2(4):357--401, 1999
- [4] E.C ten Hoeve, 3APL Platform, Master's thesis Computer Science, Utrecht University, 2003
- [5] Aneesh Krishna, Aditya K. Ghose, Sergiy A. Vilkomir, Co-evolution of complementary formal and informal requirements, Proceedings of the 7th International Workshop on Principles of Software Evolution, 2004
- [6] A. Rao. Agentspeak(l): di agents speak out in a logical computable language. In *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Eindhoven, The Netherlands, 1996.
- [7] Salim F, Chang C. F, Krishna A, Ghose A ,Towards Executable Specification: Combining i^* and AgentSpeak(L),SEKE2005
- [8] Eric S. K. Yu. Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In Proc. of the 3rd IEEE Int. Symp. On Requirements Engineering (RE'97), 226-235, Washington, DC, 1997.