

2006

Co-evolution of Agent-Oriented Conceptual Models and CASO Agent Programs

A. Dasgupta

University of Wollongong

A. Krishna

University of Wollongong, aneesh@uow.edu.au

Aditya K. Ghose

University of Wollongong, aditya@uow.edu.au

Publication Details

This paper was originally published as: Dasgupta, A, Krishna, A & Ghose, AK, Co-evolution of Agent-Oriented Conceptual Models and CASO Agent Programs, IEEE/WIC/ACM International Conference on Intelligent Agent Technology, (IAT '06), Hong Kong, China, December 2006, 686-689. Copyright 2006 IEEE.

Co-evolution of Agent-Oriented Conceptual Models and CASO Agent Programs

Abstract

Agent-Oriented conceptual modelling notations are highly effective in representing requirements from an intentional stance and answering 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 CASO language. In addition, we suggest a hybrid modelling, or co-evolution, approach in which i^* models and CASO 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 paper was originally published as: Dasgupta, A, Krishna, A & Ghose, AK, Co-evolution of Agent-Oriented Conceptual Models and CASO Agent Programs, IEEE/WIC/ACM International Conference on Intelligent Agent Technology, (IAT '06), Hong Kong, China, December 2006, 686-689. Copyright 2006 IEEE.

Co-evolution of Agent-Oriented Conceptual Models and CASO Agent Programs

Aniruddha Dasgupta, Aneesh Krishna, Aditya K.Ghose
Decision Systems Lab, School of IT and Computer Science, University of Wollongong
Wollongong, NSW 2522, Australia
Email: {ad844,aneesh,aditya}@uow.edu.au

Abstract

Agent-Oriented conceptual modelling notations are highly effective in representing requirements from an intentional stance and answering 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 CASO language. In addition, we suggest a hybrid modelling, or co-evolution, approach in which i^* models and CASO 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 approaches are becoming popular in software engineering, both as architectural frameworks, and as modelling frameworks for requirements engineering and design. The i^* modelling framework [8] is a semiformal notation built on Agent-Oriented conceptual modelling that is well-suited for answering these questions. CASO [2], which is based on AgentSpeak(L)[5], 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 [6] and [3] to design a meeting scheduler using i^* modelling [6] framework to produce executable CASO agents. We apply techniques from our earlier work whereby we produced executable AgentSpeak(L) agents. Since CASO is an extended version of AgentSpeak(L) with additional capabilities, the techniques can be applied very easily to the i^* modelling framework. Complete CASO 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 CASO executable model to get a new set 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 CASO. Sections 4 and 5 discusses how i^* and CASO 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^* Modelling Framework

The i^* [8] for Agent-Oriented conceptual modelling was designed primarily for early phase requirements engineering. An i^* consists of two main modelling 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 (shown in Figure 1) and SR diagrams are graphical representations that describe the world in a manner closer to the 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 *dependor*, while the actor depended upon is known as the *dependee*. In a goal-dependency, the *dependor* depends on the *dependee* to bring about a certain state in the world; in a task-dependency, the *dependor* depends on the *dependee* to carry out an activity; in a resource-dependency, the *dependor* depends on the *dependee* for the availability of a resource. In a softgoal-dependency, a *dependor* depends on the *dependee* to perform certain goals or task that would enhance the performance. We shall use the example of a meeting scheduler as described in [9] throughout the rest of this paper to illustrate how the i^* models can be executed. Interested readers may refer to [9] for a detailed overview.

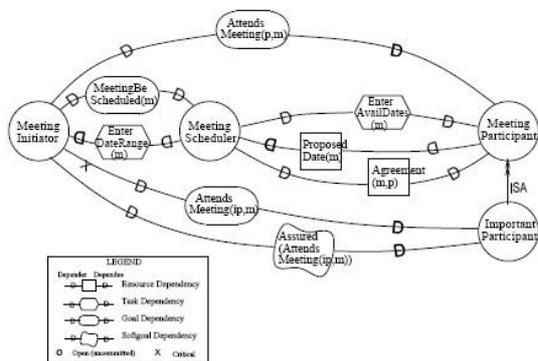


Figure 1. Strategic Dependency Diagram

3 Constraint AgentSpeak with Objectives (CASO)

We have recently worked on the development of CASO [2] - a programming language based on the popular BDI language AgentSpeak [5] which incorporates constraints and objectives into the symbolic approach of BDI model. CASO incorporates Constraint Solving and Optimization (CSOP) techniques where the optimization is based on the objective function (softgoal).

Informally, an agent program in CASO consists of a set of beliefs B , a set of constraints C , an objective function O , a set of events E , a set of intention I , a plan library P , a constraint store CS , an objective store OS and three selection functions S_E, S_P, S_I to select an event, a plan and an intention respectively to process and n_p and n_i are the two parameters which denote the number of steps to look-ahead for plan and intentions selection respectively.

A CASO plan p is of the form $t : b_1 \wedge b_2 \wedge \dots \wedge b_n \wedge c_1 \wedge c_2 \wedge \dots \wedge c_m \leftarrow sg_1, sg_2, \dots, sg_k$ where t is the trigger; each b_i refers to a belief; each c_i is an atomic constraint; each sg is either an atomic action or a subgoal.

Transition of agent program to process events depends on the event triggers. An event trigger, t , can be addition(+) or removal(-) of an achievement goal($\pm!g_i$) or a belief($\pm b_i$). Readers are directed to [2] for more information on the CASO language.

4 Mapping i* Model into CASO Agent Programs

A first step in defining a co-evolution methodology for i^* and CASO is to define a mapping from i^* to CASO. We provide the results from the earlier work [3] where this mapping was initially defined and full versions of the schemas have been described for AgentSpeak(L). The interested readers are directed to [6] and [3] for a complete

overview of the mapping guidelines. A multi-agent system (MAS) is defined in [6] as follows.

MAS is a pair $\{Agents, ESA\}$ where $Agents = a_1 \dots a_n$, each a_i is a CASO agent and ESA is a specially designated Environment Simulator Agent implemented in CASO.

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 [4]), 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. While ESA is a CASO agent, it must be provided with necessary beliefs as well as the plans. The context of the plans determines 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 CASO plans. The result of applying these rules generates the three CASO agents as mentioned above. Due to lack of space we only show the Meeting Initiator CASO agent in Figure 2 in this paper. Note that some of the plans that do not have any body do 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 ESA is also supplied by the modeler of the system (not shown here) which 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 ESA generates a user alert. The softgoals of the actors are translated into the objective function of CASO as described in [2].

Given two goal predicate symbols, goal, task, a belief predicate symbol resource and a term t :

- !goal(t) is a valid goal iff $t \subseteq N_G$.
- !task(t) is also a valid goal iff $t \subseteq N_T$.
- resource(t) is a valid belief atom iff $t \subseteq N_R$.

Given four action predicate symbols, RequestAchieve, RequestPerform, RequestResource, Supply and a term t :

- RequestAchieve(t) is a valid action iff $t \subseteq N_G$.
- RequestPerform(t) is a valid action iff $t \subseteq N_T$.
- RequestResource(t) is a valid action iff $t \subseteq N_R$.
- Supply(t) is also a valid action iff $t \subseteq N_R$.

N_G, N_T and N_R are goal, task and resource node respectively in SR and SD diagrams.

5 Co-evolution of i^* and CASO

We now propose a hybrid modelling approach from the mapping rules mentioned earlier. This hybrid modelling is composed of i^* model and CASO agents, that is, when we have an i^* model constructed for a given system, then we can also get the CASO agents of this system using the proposed mapping rules. Our problem representation, as shown in Figure 2 for the *Meeting Initiator* agent is an executable specification because it is an operational CASO programming which could therefore check the initial i^* model by executing CASO agents. The CASO agents for the *Meeting Scheduler*, *Meeting Participant* are not shown here due to lack of space. In this hybrid model, these two basic models, i^* and CASO agents, might co-evolve. At each stage, the i^* model and CASO agents are consistent. Using translation steps, they can be translated into each other. This co-evolution process will involve two aspects:

(1) *reflect the changes of i^* model on CASO agents* and (2) *reflect the changes of CASO 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 CASO 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 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 SO 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 CASO agents to i^* model. Before doing this, we need to specify the translation rules for mapping a CASO 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 CASO program to i^* model, we give another five informal mapping rules as follows:

-Addition/deletion of a CASO 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 CASO 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 CASO 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 To 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 objective 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 CASO programs can be reflected into the i* model and test a wide range of properties of the application.

<p>Actions RequestAchieve(AttendMeeting). RequestAchieve(MeetingBeScheduled). Perform(EnterDateRange).</p> <p>Plans +task(OrganizeMeeting):True ← !goal(MeetingBeScheduled), RequestAchieve(AttendMeeting). +goal(MeetingBeScheduled):True ← !task(ScheduleMeeting). +goal(MeetingBeScheduled):True ← !task(SchedulerScheduleMeeting). +task(ScheduleMeeting):True ← . +task(SchedulerScheduleMeeting):True ← RequestAchieve(MeetingBeScheduled), Perform(EnterDateRange).</p>

Figure 2. CASO plans for Meeting-Initiator Agent

6 Conclusions

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 CASO agents separately. The i* specification of a software system is easily understandable and by mapping it directly into CASO agents we can get a MAS which is directly executable. We have also defined the reverse mapping rules from CASO 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 which in turn increases the quality of the software being developed. 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 [6].

References

- [1] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. Towards executable specifications: Combining i* and agentspeak(l). In *Software Engineering and Knowledge Engineering Conference, Taiwan*, 2005.
- [2] A. Dasgupta and A. K. Ghose. Caso: A framework for dealing with objectives in a constraint-based extension to agentspeak(l). In *Proc. of the 2006 Australasian Computer Science Conference*, 2006.
- [3] A. Dasgupta, F. Salim, A. Krishna, and A. K. Ghose. Hybrid modeling using i* and agentspeak(l) agents in agent oriented software engineering. In *International Conference on Enterprise Information Systems*, 2006.
- [4] A. Fuxman, R. Kazhamiakin, M. Pistore, and M. Roveri. Formal tropos: language and semantics. 2003.
- [5] A. S. Rao. 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, 1996.
- [6] F. Salim, C. F. Chang, A. Krishna, and A. K. Ghose. Towards executable specifications: Combining i* and agentspeak(l). In *Software Engineering and Knowledge Engineering Conference, Taiwan*, 2005.
- [7] H. Simon. *The Science of the Artificial*. The MIT Press, Cambridge, MA, 2nd edition, 2005.
- [8] E. Yu. *Modelling strategic relationships for process reengineering*, Phd. Thesis. University of Toronto, Canada, 1995.
- [9] E. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE Int. Symp. on Requirements Engineering*, 1997.