Modeling Mental States in Agent-Oriented Requirements Engineering

Alexei Lapouchnian[1] and Yves Lespérance[2]

[1] Department of Computer Science, University of Toronto,
Toronto, ON M5S 3G4, Canada
alexei@cs.toronto.edu
[2] Department of Computer Science and Engineering, York University,
Toronto, ON M3J 1P3, Canada
lesperan@cs.yorku.ca

Abstract. This paper describes an agent-oriented requirements engineering approach that combines informal *i** models with formal specifications in the multiagent system specification formalism CASL. This allows the requirements engineer to exploit the complementary features of the frameworks. *i** can be used to model social dependencies between agents and how process design choices affect the agents' goals. CASL can be used to model complex processes formally. We introduce an intermediate notation to support the mapping between *i** models and CASL specifications. In the combined *i**-CASL framework, agents' goals and knowledge are represented as their mental states, which allows for the formal analysis and verification of, among other things, complex agent interactions and incomplete knowledge. Our models can also serve as high-level specifications for multiagent systems.

1 Introduction

Modern software systems are becoming increasingly complex, with lots of intricate interactions. The recent popularity of electronic commerce, web services, supply chain management and other inter-organizational systems, digital libraries, etc. confirms the need for software engineering methods for constructing applications that are open, distributed, and adaptable to change. This is why many researchers and practitioners are looking at agent technology as a basis for distributed applications.

Agents are active, social, and adaptable software system entities situated in some environment and capable of autonomous execution of actions in order to achieve their set objectives [20]. Furthermore, most problems are too complex to be solved by just one agent — one must create a multiagent system (MAS) with several agents having to work together to achieve their objectives and ultimately deliver the desired application. Therefore, adopting the agent-oriented approach to software engineering means that the problem is decomposed into multiple, autonomous, interacting agents, each with a particular objective (goal). Agents in MAS frequently represent individuals, companies, etc. This means that there is an underlying organizational context in MAS. Like humans, agents need to coordinate their activities, cooperate, request help

from others, etc., often through negotiation. Unlike in object-oriented or component-based systems, interactions in multiagent systems occur through high-level agent communication languages, so these interactions are mostly viewed not at the syntactic level, but at the knowledge level, in terms of goal delegation, etc. [20].

In requirements engineering (RE), *goal-oriented approaches* (e.g, KAOS [3]) have become prominent. In Goal-Oriented Requirements Engineering (GORE), high-level stakeholder objectives are identified as goals and then refined into fine-grained requirements assignable to agents/components in the system-to-be or in its environment. Their reliance on goals makes GORE methods and agent-oriented software engineering a great match. Moreover, agent-oriented analysis is central to requirements engineering since the assignment of responsibilities for goals and constraints to components in the system-to-be and agents in its environment is the main result of the RE process. Therefore, it is natural to use a goal-oriented requirements engineering approach when developing MAS. With GORE, it is easy to make the transition from the requirements to the high-level MAS specifications. For example, strategic relationships among agents will become high-level patterns of inter-agent communication. Thus, it would be desirable to devise an *agent-oriented RE approach with a formal component that supports rigorous formal analysis, including reasoning about agents' goals and knowledge*.

In the above context, while it is possible to informally analyze small systems, formal analysis is needed for any realistically-sized system to determine whether such distributed requirements imposed on each agent in a MAS are correctly decomposed from the stakeholder goals, consistent and, if properly met, achieve the system's overall objectives. Therefore, the aim of this work is to devise an agent-oriented requirements engineering approach with a formal component that supports reasoning about agents' goals (and knowledge), thereby allowing for formal analysis of the requirements expressed as the objectives of the agents in a MAS.

In our approach we integrate the *i** modeling framework [21] with CASL [14], a formal agent-oriented programming language supporting the modeling of agent mental states. This gives the modeler the flexibility and intuitiveness of the *i** notation as well as the powerful formal analysis capability of CASL. To bridge the gap between informal *i** diagrams and formal CASL specifications we propose an intermediate notation that can be easily obtained from *i** models and then mapped into CASL. With our *i**-CASL-based approach, a CASL model can be used both as a requirements analysis tool and as a formal high-level specification for a multiagent system that satisfies the requirements. This model can be formally analyzed using the CASLve [15] tool or other tools and the results can be fed back into the requirements model.

One of the main features of this approach is that goals (and knowledge) are assigned to particular agents thus becoming their subjective attributes as opposed to being objective system properties as in many other approaches (e.g., Tropos [1] and KAOS [3]). This allows for the modeling of conflicting goals, agent negotiation, information exchange, complex agent interaction protocols, etc.

The rest of the paper is organized as follows: Section 2 briefly describes the concepts of *i** and CASL; Section 3 discusses our approach in detail and Section 4 concludes the paper.

2 Background

2.1 The *i** Framework

*i** [21] is an agent-oriented modeling framework that can be used for requirements engineering, business process reengineering, etc. *i** centers on the notion of *intentional actor* and *intentional dependency*. Actors are described in their organizational setting and have attributes such as goals, abilities, beliefs, and commitments. In *i**, an actor can use opportunities to depend on other actors in achieving its objectives, at the same time becoming vulnerable if those actors do not deliver. *i** actors are *strategic* in the sense that they are concerned with the achievement of their goals and strive to find a balance between their opportunities and vulnerabilities. Similarly, dependencies in *i** are *intentional* since they appear as a result of actors pursuing their goals.

In this paper, we use a variant of the meeting scheduling problem, which has become a popular exemplar in Requirements Engineering (e.g., [17]). In the context of the *i** modeling framework the process was first analyzed in [21]. We introduce a number of modifications to the meeting scheduling process to make our models easier to understand. For instance, we take the length of meetings to be the whole day. We also assume that in the environment of the system-to-be there is a legacy software system called the Meeting Room Booking System (MRBS) that handles the booking of meeting rooms. The complete case study is presented in [7].

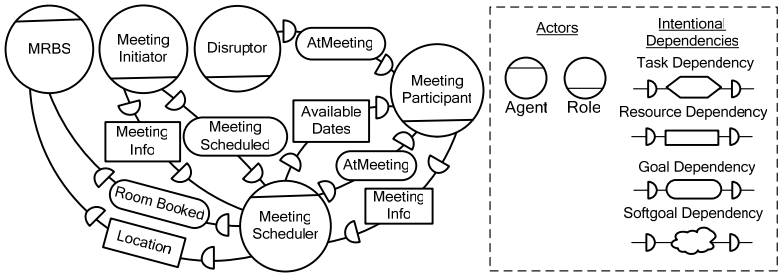

Fig. 1. The Meeting Scheduler in its environment

The *i** framework has two main components: the *Strategic Dependency (SD) model* and the *Strategic Rationale (SR) model*. The former describes the external relationships among actors, while the latter focuses on exploring the rationale behind the processes in organizations from the point of view of participating actors. SD models are networks of actors (which can be agents, positions, and roles) and dependencies. Depending actors are called *dependers* and depended-upon actors are called *dependees*. There can be four types of dependencies based on what is being delegated – a goal, a task, a resource, or a softgoal. Softgoals are related to the notion of non-functional requirements [2] and model quality concerns of agents.

Fig. 1 is an SD diagram showing the computerized Meeting Scheduler (MS) agent in its environment. Here, the role Meeting Initiator (MI) depends on the MS for scheduling meetings and for being informed about the meeting details. The MS, in

turn, depends on the Meeting Participant role for attending meetings and for providing his/her available dates to it. The MS uses the booking system to book rooms for meetings. The Disruptor actor represents outside actors that cause changes in participants' schedules, thus modeling the environment dynamics.

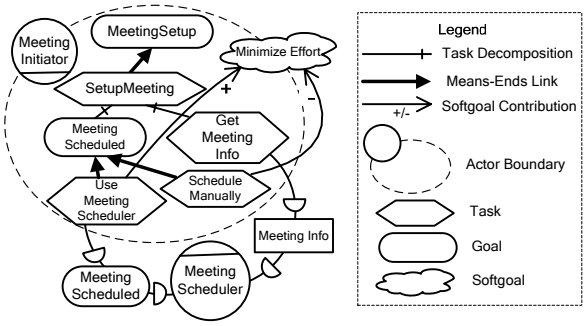

Fig. 2. SR model for the meeting initiator

SR models enable the analyst to assess possible alternatives in the definition of actor processes to better address their concerns. Four types of nodes are used in SR models – goals, tasks, softgoals, and resources – and three types of links – means-ends links, task decompositions links, and softgoal contribution links. Means-ends links specify alternative ways to achieve goals; task decomposition links connect tasks with components needed for their execution. For example, Fig. 2 is a simple SR models showing some details of the MI process. To schedule meetings, the MI can either do it manually, or delegate it to the scheduler. Softgoal contribution links specify how process alternatives affect quality requirements (softgoals), and so softgoals such as `MinimizeEffort` in Fig. 2 are used to evaluate these alternatives.

2.2 The Formal Foundations: CASL

The Cognitive Agents Specification Language (CASL) [13][14] is a formal specification language that combines theories of action [10][11] and mental states [12] expressed in situation calculus [9] with ConGolog [4], a concurrent, non-deterministic agent-oriented programming language with a formal semantics. In CASL, agents' goals and knowledge are modeled formally; communication actions are provided to update these mental states and ConGolog is then employed to specify the behaviour of agents. This combination produces a very expressive language that supports high-level reasoning about the agents' mental states. The logical foundations of CASL allow it to be used to specify and analyze a wide variety of MAS including non-deterministic systems and systems with incompletely specified initial state.

CASL specifications consist of two parts: the model of the domain and its dynamics (the declarative part) and the specification of the agents' behaviour (the procedural part). The domain is modeled in terms of the following entities: 1) *primitive actions* – all changes in the domain are due to primitive actions being executed by agents; 2) *situations*, which are states of the domain that result from the execution of

sequences of actions (there is a set of initial situations, with no predecessor, corresponding to the ways agents think the world might be like initially); 3) *fluents*, which are predicates and functions that may change from situation to situation. The fluent `Room(meetingID,date,room,s)`, where *s* is a situation parameter, models the fact that a room has been booked on some day for some meeting in a situation *s*.

To specify the dynamics of an application domain, we use the following types of axioms: 1) *action precondition axioms* that describe under what conditions actions can be performed; 2) *successor state axioms* (SSA), which were introduced in [10] as a solution to the frame problem and specify how primitive actions affect fluents; 3) *initial state axioms*, which describe the initial state of the domain and the initial mental states of the agents; 4) *other axioms* that include unique name axioms for actions and domain independent foundational axioms.

Agents' behaviour is specified using a rich high-level programming language with recursive procedure declarations, loops, conditionals, non-determinism, concurrency, and interrupts [4]. A special predicate ***Do****(Program,s,s')* holds if there is a successful execution of *Program* that ends in situation *s′* after starting in *s*.

CASL supports formal modeling of agents' goals and knowledge. The formal representation for both is based on a *possible worlds semantics* incorporated into the situation calculus, where situations are viewed as possible worlds [12]. CASL uses accessibility relations *K* and *W* to model what an agent knows and what it wants respectively. *K(agt,s′,s)* holds if the situation *s′* is compatible with what the agent *agt* knows in situation *s*, i.e., in situation *s*, the agent thinks that it might be in the situation *s′*. In this case, the situation *s′* is called *K-accessible*. When an agent does not know the value of some formula φ, it considers possible (formally, *K*-accessible) some situations where φ is true and some where it is false. An agent knows some formula φ if φ is true in all its *K*-accessible situations: **Know***(agt,*φ*,s)*$= \forall s'(K(agt,s,s') \supset \varphi[s'])$. Constraints on the *K* relation ensure that agents have positive and negative introspection (i.e., agents know whether they know/don't know something) and guarantee that what is known is true. Communication actions such as `inform` are used for exchanging information among agents. The precondition for the `inform` action ensures that no false information is transmitted. The changes to agents' knowledge due to communication and other actions are specified by the SSA for the *K* relation. The axiom ensures that agents are aware of the execution of all actions. This formal framework is quite simple and idealized. More complex versions of the SSA can be specified, for example, to handle encrypted messages [14] or to provide belief revision [16].

The accessibility relation *W(agt,s′,s)* holds if in situation *s* an agent considers that everything that it wants to be true actually holds in *s′*, which is called *W*-accessible. We use the formula **Goal***(agt,*ψ*,s)* to indicate that in situation *s* the agent *agt* has the goal that ψ holds. The definition of **Goal** says that ψ must be true in all *W*-accessible situations that have *K*-accessible situation in their past. This ensures that, while agents may want something they know is impossible to obtain, the goals of agents must be consistent with what they currently know. In our approach, we mostly use achievement goals that specify the desired states of the world. We use the formula **Goal***(agt,***Eventually***(*ψ*),s)* to state that *agt* has the goal that ψ is eventually true. The `request` and `cancelRequest` actions are used by agents to request services from

other agents and cancel their requests respectively. Requests are used to establish intentional dependencies among actors and lead to changes in goals of the requestee agent. The dynamics of the *W* relation are specified, as usual, by an SSA. There are constraints on *W* and *K* relations, which ensure that agents' goals are consistent and that that agents introspect their goals.

3 The *i**-CASL Notation and Process

3.1 A Motivating Example

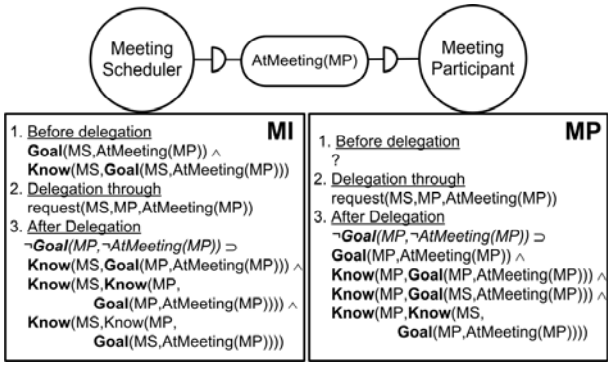

Fig. 3. A motivating example

Suppose that we are employing an approach like Tropos [1], a requirements-driven agent-oriented software development methodology that uses the *i** modeling notation, to model a simple goal delegation involving two agents. Fig. 3 shows a goal dependency where the Meeting Scheduler depends on the Meeting Participant for attending a meeting. We would like to be able to analyze this interaction and predict how it will affect the goals and the knowledge of these agents. Using the approach proposed in this paper, one can develop a formal model based on the SD diagram in Fig. 3 and the corresponding SR-level models (as will be shown later), analyze it, and conclude that, for example, before the goal delegation, the MS has the goal `AtMeeting(MP)` and knows about this fact. After the delegation (and provided that the MP did not have a conflicting goal), the MS knows that the MP has acquired the goal, that the MP knows that it has the goal, that the MP knows that the MS has the same goal, etc. For the Participant agent in Fig. 3, we cannot say what its mental state was before the goal delegation. But, after the request from the MS we know that it has the goal `AtMeeting(MP)` and knows about it, etc. The MP also knows how it has acquired the goal and thus will be able to trace its intention to achieve `AtMeeting(MP)` to the Meeting Scheduler's request.

Note that the change in mental state of the requestee agent is the core of goal delegation. Also, in our approach, goals and knowledge are attributes of particular agents. This allows for better models of agent conflicts, interaction, negotiation, etc.

3.2 Increasing Precision with iASR Models

Our aim in this approach is to tightly associate SR models with formal specifications in CASL. The standard SR diagrams are geared to informal analysis and can be very ambiguous. For instance, they lack the details on whether the subtasks in task decompositions are supposed to be executed sequentially, concurrently, under certain conditions, etc. CASL, on the other hand, is a precise language. To handle this precision mismatch we use Intentional Annotated SR (iASR) models that help in bridging the gap between SR models and CASL specifications. Our goal is to make iASR models precise graphical representation for the procedural component of CASL specifications while helping with the identification of axioms and definitions in the declarative one.

The starting point for developing an iASR diagram for an actor is the regular SR diagram for that actor (e.g., see Fig. 2). It can then be appropriately transformed to become an iASR model every element of which can easily be mapped into CASL. The steps for producing iASR models from SR ones include the addition of model annotations and the details of agent interactions, the removal of softgoals, the *deidealization* of goals [17], etc.

Annotations. The main tool that we use for disambiguating SR models is *annotations*. Annotations allow analysts to model the domain more precisely and to capture data/control dependencies among goals and other details. Annotations, proposed in [19] for use with SR models and ConGolog, are textual constraints on iASR models and can be of three types: composition, link, and applicability conditions. Composition annotations (specified by σ in Fig. 4) are applied to task and means-ends decompositions and specify how the subtasks/subgoals are to be combined to execute the supertask and achieve the goal respectively. Four types of composition are allowed: sequence (";"), which is default for task decompositions, concurrency ("||"), prioritized concurrency ("»"), and alternative ("|"), which is the default for means-ends decompositions. These annotations are applied to subtasks/subgoals from left to right. E.g., in Fig. 4, if the "»" annotation is applied, n_1 has the highest priority, while n_k has the lowest. The choice of composition annotations is based on the ways actions and procedures can be composed together in CASL.

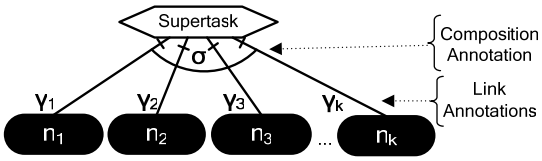

Fig. 4. Composition and link annotations

Link annotations (γ_i in Fig. 4) are applied to subtasks/subgoals (n_i) and specify how/under what conditions they are supposed to be achieved/executed. There are six types of link annotations (corresponding to CASL operators): *while* loop, *for* loop, the *if* condition, the pick, the interrupt, and the guard. The pick annotation (π`(variableList,condition)`) non-deterministically picks values for variables in the subtask that satisfy the condition. The interrupt (`whenever(variableList, condition,cancelCondition)`) fires whenever there is a binding for the variables

that satisfies the condition unless the cancellation condition becomes true. Guards (`guard(condition)`) block the subtask's execution until the condition becomes true. The absence of a link annotation on a particular decomposition link indicates the absence of any conditions on the subgoal/subtask.

The third annotation type is the *applicability condition* (`ac(condition)`). It applies to means-ends links used with goal achievement alternatives and specifies when the corresponding alternatives are applicable (see below for an example).

Softgoals. Softgoals (quality requirements) are imprecise and thus are difficult to handle in a formal specifications language. Therefore, we use softgoals to help in choosing the best process alternatives (e.g., by selecting the ones with the best overall contribution to all the softgoals in the model) and then remove them before iASR models are produced. Alternatively, softgoals can be operationalized or metricized, thus becoming hard goals. Also, applicability conditions in iASR models can be used to capture the fitness of the remaining alternatives w.r.t. softgoals, which is normally encoded by softgoal contributions in SR diagrams. For example, one can specify that phoning participants to notify them of the meeting details is applicable only in cases with few participants (see Fig. 8), while the email option is applicable for any number of participants. This may be due to the softgoal "Minimize Effort".

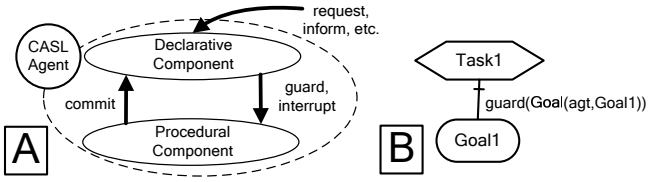

Fig. 5. Synchronizing procedural and declarative components of CASL specifications

Agent Goals in iASR Models. A CASL agent has a procedural and a declarative components. iASR diagrams only model agent processes and therefore can only be used to represent the procedural component of CASL agents. The presence of a goal node in an iASR diagram indicates that the agent knows that the goal is in its mental state and is prepared to deliberate about whether and how to achieve it. For the agent to modify its behaviour in response to the changes to its mental state, it must detect that change and synchronize its procedural and declarative components (see Fig. 5A). Agent mental states are specified declaratively and usually change as a result of communication acts that realize goal delegation and information exchange. Thus, the procedural component of the agent must monitor for these changes. To do this we use interrupts or guards with their conditions being the presence of certain goals or knowledge in the mental state of the agent (Fig. 5B). Procedurally a goal node is interpreted as invoking the means to achieve it.

In CASL, as described in [14], only communication actions have effects on the mental state of the agents. We, on the other hand, would like to let agents change their mental state on their own by executing the action `commit(agent,φ)`, where φ is a formula that the agent/modeler wants to hold. Thus, in iASR diagrams all agent goals must be acquired either from intentional dependencies or by using the `commit` action.

By introducing goals into the models of agent processes, the modeler captures the fact that multiple alternatives exist in these processes. Moreover, the presence of goal nodes suggests that the designer envisions new possibilities for achieving these goals. By making the agent acquire the goals, the modeler makes sure that the agent's mental state reflects the above intention. In this way the agents would be able to reason about various alternatives available to them or come up with new ways to achieve their goals at runtime. Self-acquired goals add flexibility to system models by preserving within the corresponding formal specifications the variability in the way goals can be achieved and by avoiding early operationalization of goals. Self-acquired goals can be used to "load" goal refinements and AND/OR goal decompositions, which are abundant in GORE and AI, into the mental state of the agent if reasoning about these refinements is required. This is unlike the approach in [19] where agent goals had to be operationalized before being formally analyzed.

Another way of increasing the precision of the iASR model is the addition of parameters to iASR models. For example, in Fig. 6B, all of the nodes in the model have the parameter `mid` (short for "meeting ID"), a unique meeting identifier. Quite frequently, we replace the conditions in annotations and other model elements (they tend to be long) with suitably named abbreviations, e.g., `RequestedDateRange(mid)`.

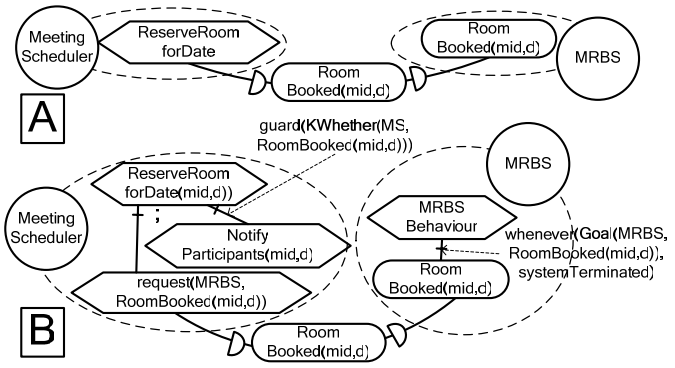

Fig. 6. Adding iASR-level agent interaction details

Providing agent interaction details. *i** usually abstracts from modeling any details of agent interactions. CASL, on the other hand, models high-level aspects of inter-agent communication: requests for services or information, the selection of the course of action upon the receipt of the information, etc. Because of the importance of agent interactions in MAS, in order to formally verify multiagent system specifications in CASL, all high-level aspects of these interactions must be provided in the corresponding iASR models. This includes the tasks that request services or information from agents in the system, the tasks that supply the information or inform about success or failure in providing the service, etc. We assume that the communication links are reliable.

For example, the SR model with the goal dependency `RoomBooked` (see Fig. 1) in Fig. 6A is refined into the iASR model in Fig. 6B showing the details of the requests, the interrupts with their trigger conditions referring to mental states of the agent, etc.

3.3 Mapping iASR Diagrams into CASL

Once all the necessary details have been introduced into an iASR diagram, it can be mapped into the corresponding CASL model, thus making the iASR model amenable to formal analysis.

The modeler defines a mapping **m** that maps every element (except for intentional dependencies) of an iASR model into CASL. This mapping associates iASR model elements with CASL procedures, primitive actions, and formulas so that a CASL program can be generated from an iASR model. Specifically, agents are mapped into constants that serve as their names as well as into CASL procedures that specify their behaviour; roles and positions are mapped into similar procedures with an agent parameter so that they can be instantiated by individual agents; leaf-level task nodes are mapped into CASL procedures or primitive actions; composition and link annotations are mapped into the corresponding CASL operators, while the annotation conditions map into CASL formulas.

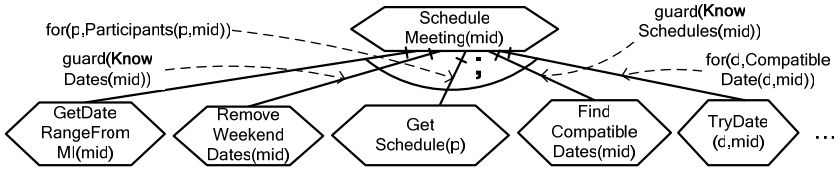

Fig. 7. Example iASR task decomposition

Mapping Task Nodes. A task decomposition is automatically mapped into a CASL procedure that reflects the structure of the decomposition and all the annotations. While the possibility of mapping leaf-level tasks into CASL procedures may reduce model size and increase the level of abstraction, restricting the mapping of these tasks to similarly named primitive actions allows the CASL procedures to be automatically constructed from these actions based on iASR annotations.

Fig. 7 shows how a portion of the Meeting Scheduler's task for scheduling meetings can be decomposed. This task will be mapped into a CASL procedure with the following body (it contains portions still to be (recursively) mapped into CASL; they are the parameters of the mapping **m**):

```
proc ScheduleMeetingProc(mid)
 m(GetDateRangeFromMI(mid));
 guard m(KnowDates(mid)) do m(RemoveWeekendDates(mid))
   endGuard;
 for p: m(Ptcp(mid)) do m(GetSchedule(p)) endFor;
 guard m(KnowSchedules(mid)) do m(FindCompatibleDates(mid))
   endGuard;
 for d: m(CompatibleDate(d,mid)) do m(TryDate(d,mid)) endFor;
…
endProc
```

Note how the body of the procedure associated with the `ScheduleMeeting` task is composed of the results of the mapping of its subtasks with the annotations provid-

ing the composition details. This procedure can be mechanically generated given the mapping for leaf-level tasks and conditions.

Mapping Goal Nodes. In our approach, an iASR goal node is mapped into a CASL formula, which is the formal definition for the goal, and an achievement procedure, which encodes how the goal can be achieved and is based on the means-ends decomposition for the goal in the iASR diagram. For example, a formal definition for `MeetingScheduled(mid,s)` could be: `∃d[AgreeableDate(mid,date,s) ∧ AllAccepted(mid,date,s) ∧ RoomBooked(mid,date,s)]`. This says that there must be a date agreeable for everybody on which a room is booked and all participants have accepted to meet. This seems correct, but initial formal goal definitions are often too ideal for the goal that cannot always be achieved. Such goals must be *deidealized* [17]. In order to weaken the goal appropriately, one needs to know under what circumstances the goal cannot be achieved. Modeling an achievement process for a goal using an iASR diagram allows us to understand how that goal can fail and thus iASR models can be used to come up with a correct formal definition for the goal. For example, it is not always possible to schedule a meeting. Here is one way to deidealize the goal `MeetingScheduled` based on our iASR model analysis:

```
MeetingScheduledIfPossible(mid,s)=
//1. The meeting has been successfully scheduled
SuccessfullyScheduled(mid,s) ∨
//2. No agreeable (suitable for everybody) dates
∀d[IsDate(d) ⊃ ¬AgreeableDate(mid,d,s)] ∨
//3. For every agreeable date at least one participant declined
∀d[AgreeableDate(mid,d,s)⊃ SomeoneDeclined(mid,d,s)] ∨
//4. No rooms available
∀d[SuggestedDate(mid,d,s) ∧ AllAccepted(mid,d,s) ⊃
   ¬RoomBookingFailed(mid,date,s)]
```

CASL's support for reasoning about agent goals presented us with an interesting possibility. In the case study, we decided not to maintain schedules for meeting participants explicitly. Instead, we relied on the presence of goals `AtMeeting(participant,mid,date,s)` in their mental states as indications of the participants' intention to attend certain meetings on certain dates (the absence of meeting commitments indicates an available time slot). Then, we made the participants know that they can only attend one meeting per time slot (a day in our case) with the following initial state axiom (this can be shown to persist in all situations):

```
∀agt[Know(agt,∀p,mid1,mid2,date[AtMeeting(p,mid1,date,now) ∧
    AtMeeting(p,mid2,date,now) ⊃ mid1=mid2],S0)]
```

Thus, the consistency of participants' schedules is automatically maintained since meeting requests conflicting with already adopted `AtMeeting` goals are rejected.

The achievement procedures for goals are automatically constructed based on the modeled means for achieving them and the associated annotations including the applicability conditions (see Fig. 8). By default, the alternative composition annotation is used, which means that some applicable alternative will be non-deterministically selected. Other approaches are also possible, e.g., one can try all appropriate alterna-

tives concurrently or in sequence. Note that the applicability condition (ac) maps into a guard operator to prevent the execution of unwanted alternatives.

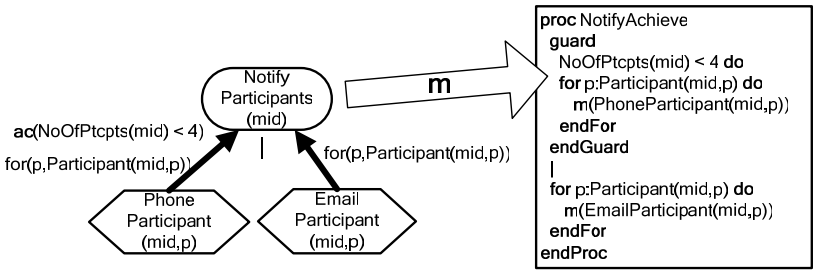

Fig. 8. Generating achievement procedures

Modeling Dependencies. Intentional dependencies are not mapped into CASL per se – they are established by the associated agent interactions. iASR tasks requesting help from agents will generally be mapped into actions of the type `request(FromAgt,ToAgt,`**`Eventually`**`(φ))` for achievement goals φ. We add a special abbreviation **DoAL**`(δ,s,s′)` (Do At Least) to be used when establishing task dependencies. It stands for **Do**`(δ||(πa.a)*,s,s′)`, which means that the program δ must be executed, but that any other action may occur. Thus, to ask an agent to execute a certain known procedure, the depender must request it with: `request(FromAgt, ToAgt,`**DoAL**`(SomeProcedure))`.

In order for an intentional dependency to be established we also need a commitment from a dependee agent to act on the request from the depender. Thus, the dependee must monitor its mental state for newly acquired goals. For example, here is an interrupt that is used by the Meeting Participant to check for a request for the list of its available dates:

```
<mid:Goal(mp,DoAL(InformAvailableDates(mid,MS),now,then) ∧
  Know(mp,¬∃s,s′(s ≤ s′ ≤ now ∧
          DoAL(InformAvailDates(mid,MS),s,s′))) →
  InformAvailDates(mid,MS)
until SystemTerminated>
```

Here, if the MP has the goal to execute the procedure `InformAvailDates` and knows that it has not yet executed it, the agent sends the available dates. The cancellation condition `SystemTerminated` indicates that the MP always monitors for this goal. Requesting agents use similar interrupt/guard mechanism to monitor for requested information or confirmations. When modeling agent interaction protocols in this approach, for every incoming message an agent will have an interrupt monitoring for it with its body specifying the appropriate response to the message. Since the interrupts fire when changes in the mental state are detected, agents can execute the protocols flexibly by, for example, self-acquiring the goal of buying some urgently required product from a vendor and thus skipping the lengthy price negotiation part of the protocol. Also, cancellation conditions in interrupts allow the agents to monitor for certain requests/informs only in particular contexts (e.g., while some interaction

protocol is being enacted). A CASL specification for a simple interaction protocol is described in [7].

3.4 Formal Verification

Once an iASR model is mapped into the procedural component of the CASL specification and after its declarative component (e.g., precondition axioms, SSAs, etc.) has been specified, it is ready to be formally analyzed. One tool that can be used is CASLve [15], a theorem prover-based verification environment for CASL. CASLve provides a library of theories for representing CASL specifications and lemmas that facilitate various types of verification proofs. [13] shows a proof that there is a terminating run for a simplified meeting scheduler system as well as example proofs of a safety property and consistency of specifications. In addition to physical executability of agent programs, one can also check for the *epistemic feasibility* [8] of agent plans, i.e., whether agents have enough knowledge to successfully execute their processes.

Other approaches could be used as well, for instance, simulation or model checking. However, tools based on these techniques work with much less expressive languages than CASL. Therefore, CASL specifications must be simplified before these methods can be used on them. For example, most simulation tools cannot handle mental state specifications; these would then have to be operationalized before simulation is performed. The ConGolog interpreter can be used to directly execute such simplified specifications, as in [19]. Model checking methods (e.g. [5]) are restricted to finite state specifications, and work has only begun on applying these methods to theories involving mental states (e.g., [18]).

If expected properties of the system are not entailed by the CASL model, it means that the model is incorrect and needs to be fixed. The source of an error found during verification can usually be traced to a portion of the CASL code and to a part of the corresponding iASR model since our systematic mapping supports traceability.

3.5 Discussion and Future Work

In the approach presented in this paper and in [7], we produce CASL specifications from *i** models for formal analysis and verification. The approach is related to the Tropos framework in that it is agent-oriented and is rooted in the RE concepts. Our method is not the first attempt to provide formal semantics for *i** models. For example, Formal Tropos (FT) [5], supports formal verification of *i** models through model checking. Also, in the *i**-ConGolog approach [19], on which our method is based, SR models are associated with formal ConGolog programs for simulation and verification. Additionally, the Trust-Confidence-Distrust approach [6] combined *i** and ConGolog to model and analyze trust in social networks. The problem with all these methods is that goals of the agents are abstracted out and made into objective properties of the system in the formal specifications. This is done due to the fact that the formal components of these approaches (the model checker input language for FT and ConGolog for the other) do not support reasoning about agent goals (and knowledge). However, most of the interactions among agents involve knowledge exchange

and goal delegation since multiagent systems are developed as social structures. Thus, complementing informal modeling techniques such as *i** with formal analysis of agent goals and knowledge is very important in the design of multiagent systems.

We use a version of CASL where the precondition for the `inform` action requires that the information being sent by an agent be known to it (we assume that what is known must be true). This prevents agents from transmitting false information. The removal of this restriction allows the modeling of systems where agents are not always truthful. This can be useful when dealing with security and privacy requirements. However, dealing with false information may require belief revision, which complicates the model somewhat (see [16]). Similarly, the precondition for `request` makes sure that the sender does not itself have goals that conflict with the request. Relaxing this constraint also allows for the possibility of modeling malicious agents.

Other extensions to CASL to accommodate various characteristics of application domains are possible. For example, in many domains one needs to specify whom an agent trusts and to whom it is helpful. In [7] we proposed a simple way to handle trust and helpfulness in CASL. Fine-grained modeling of trust and helpfulness among agents in our approach is future work.

We also point out that CASL assumes that all agents are aware of all actions being executed in the system. Often, it is useful to lift this restriction, but dealing with the resulting lack of knowledge about agents' mental states can be challenging. In future work, we plan to address these issues. We would also like to accommodate reasoning about softgoals in our framework as well as to test the method on more realistic case studies. Additionally, we are developing a toolkit to support requirements engineering using our approach.

While the procedural component of a CASL specification accurately reflects the corresponding iASR model, the model only hints on what has to be in the declarative component of the specification (e.g., the axioms for actions, the definitions of annotation conditions, etc.) We expect that our RE toolkit will be able to significantly simplify the specification of the declarative component of CASL models.

4 Conclusion

In this paper, we have proposed a framework for agent-oriented requirements engineering incorporating both graphical and formal notations. The graphical notation allows for comprehensive modeling of system requirements as well as of its organizational setting including stakeholder goals and goal delegation. These models are then gradually made more precise so that they can be mapped into formal agent specifications where goals are not removed, but are modeled formally and can be updated following requests. This allows agents to reason about their objectives. Information exchanges among agents are also formalized as changes in their knowledge state. In our approach, goals and knowledge are not system-wide properties, but belong to concrete agents. This supports the modeling of conflicting goals, agent negotiation, information exchange, complex agent interaction protocols, etc. The generated formal model can be used both as a requirements analysis tool and as a formal high-level specification for the multiagent system.

References

1. Castro J., Kolp M., Mylopoulos, J.: Towards Requirements-Driven Information Systems Engineering: The Tropos Project. *Information Systems*, 27(6) (2002) 365-389
2. Chung, L.K., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer (2000)
3. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-Directed Requirements Acquisitions. *Science of Computer Programming*, 20 (1993) 3-50
4. De Giacomo, G., Lespérance, Y., Levesque, H.: ConGolog, A Concurrent Programming Language Based on the Situation Calculus. *Artificial Intelligence*, 121 (2000) 109-169
5. Fuxman, A., Liu, L., Mylopoulos, J., Pistore, M., Roveri, M., Traverso, P.: Specifying and Analyzing Early Requirements in Tropos. *RE Journal*, 9(2) (2004) 132-150
6. Gans, G., Jarke, M., Kethers, S., Lakemeyer, G., Ellrich, L., Funken, C., Meister, M.: Requirements Modeling for Organization Networks: A (Dis-)Trust-Based Approach. Proc. *RE'01* (2001) 154-163
7. Lapouchnian, A.: Modeling Mental States in Requirements Engineering – An Agent-Oriented Framework Based on *i** and CASL. M.Sc. Thesis. Department of Computer Science, York University, Toronto (2004)
8. Lespérance, Y.: On the Epistemic Feasibility of Plans in Multiagent Systems Specifications. Proc. *ATAL-2001*, Revised papers, LNAI 2333, Springer, Berlin (2002) 69-85
9. McCarthy, J., Hayes, P.: Some Philosophical Problems From the Standpoint of Artificial Intelligence, *Machine Intelligence*, Vol. 4, Edinburgh University Press (1969) 463-502
10. Reiter, R.: The Frame Problem in the Situation Calculus: A Simple Solution (Sometimes) and a Completeness Result for Goal Regression. Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy, V. Lifschitz (ed.), Academic Press (1991) 359-380
11. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press, Cambridge MA (2001)
12. Scherl, R.B., Levesque, H.: Knowledge, Action, and the Frame Problem. Artificial Intelligence, 144(1-2) (2003) 1-39
13. Shapiro, S.: Specifying and Verifying Multiagent Systems Using CASL. Ph.D. Thesis. Department of Computer Science, University of Toronto (2004)
14. Shapiro, S., Lespérance, Y.: Modeling Multiagent Systems with the Cognitive Agents Specification Language - A Feature Interaction Resolution Application. *ATAL-2000*, LNAI 1986, Springer, Berlin (2001) 244-259
15. Shapiro, S., Lespérance, Y., Levesque, H.: The Cognitive Agents Specification Language and Verification Environment for Multiagent Systems. Proc. *AAMAS'02*, Bologna, Italy, ACM Press (2002) 19-26
16. Shapiro, S., Pagnucco, M., Lespérance, Y., Levesque, H.: Iterated Belief Change in the Situation Calculus. Proc. *KR-2000* (2000) 527-538
17. van Lamsweerde, A., Darimont, R., Massonet, P.: Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt. Proc. *RE'95*, York, UK (1995) 194-203
18. van Otterloo, S., van der Hoek, W., Wooldrige, M.: Model Checking a Knowledge Exchange Scenario. *Applied Artificial Intelligence*, 18:9-10 (2004) 937-952
19. Wang, X., Lespérance, Y.: Agent-Oriented Requirements Engineering Using ConGolog and *i**. Proc. *AOIS-01* (2001) 59-78
20. Wooldridge, M.: Agent-Based Software Engineering. *IEE Proceedings on Software Engineering*, 144(1) (1997) 26-37
21. E. Yu. Towards modeling and reasoning support for early requirements engineering. Proc. *RE'97*, Annapolis, USA (1997) 226-235