Agent-Oriented Requirements Engineering Using ConGolog and *i**

Xiyun Wang and Yves Lespérance*

Dept. of Computer Science, York University
Toronto, On, M3J 1P3, Canada
xiyun@cs.yorku.ca, lesperan@cs.yorku.ca

Abstract: Agent-oriented approaches are becoming popular in software engineering, both as architectural frameworks, and as modeling frameworks for requirements engineering and design. *i** is an informal diagram-based language for early-phase requirements engineering that supports the modeling of social dependencies between agents and how process design choices affect the agents' goals both functional and non-functional. ConGolog is an expressive logic-based formalism for specifying processes that involves multiple agents. Tools are being developed to support the validation of ConGolog process models though simulation and verification. The two formalisms complement each other well, and in this work, we develop a methodology for their combined use in requirements engineering. The *i** SR-diagram language is extended with process specification annotations, which allow the SR model of a system to be refined and then mapped into a ConGolog model. The mapping must satisfy a set of mapping rules, which ensure that it specifies which elements in the two models are related and that the models are consistent. The methodology is illustrated on a meeting scheduling application example.

1 Introduction

With the advent of the World-Wide Web and electronic commerce, trends in software are towards open systems, more integration across applications, and systems that can adapt to change. In response to this, many developers are starting to adopt agent-oriented architectures, where a system is composed of agents, autonomous entities that can interact in flexible ways, for instance through negotiation, while working towards their goals and reacting to changes in the environment [JW98]. To support the development of agent-based systems, suitable software engineering methods and tools are required. So far, most efforts in this area have been directed at the design phase of software development. In this paper, we focus mainly on the requirements engineering phase

Requirements engineering started with the study of what the system should do, i.e. late-phase requirements analysis, which focuses on the specification of requirements, their completeness, consistency, automated verification, etc. However, it has become known that the early phase of requirements analysis is as important as the later phase. Early-phase requirements analysis focuses on why the system must be developed, how the desired system will meet its goals, what alternatives can be proposed, what the relationships between various actors or stakeholders are, and how the interests of actors can be achieved.

There are many formal requirements modeling languages and frameworks for late-phase requirements analysis, for instance KAOS [DVF93], ALBERT-II [DuDois95], etc. Early phase requirements analysis on the other hand, has generally been done informally and without much tool/formalism support. The *i**

* Corresponding author.

framework [Yu95] was developed for early phase requirements analysis. It provides an informal diagram-based notation that supports the modeling of social dependencies between agents and how process design choices affect agents' goals. But *i** is not a formal language and it has limited support for describing complex processes. *ConGolog* [DLL00, LKMY99] is a formal language for process specification and agent programming. It supports the formal specification of complex multiagent systems, but lacks features for modeling the rationale behind design choices. The two frameworks complement each other well and it would be good to have a methodology for using them in combination.

In this paper, we present an approach to the combined use of the *i** and ConGolog frameworks for requirements analysis. The *i** framework will be used to model different alternatives for the desired system, analyze and decompose the functions of the different actors, and model the dependency relationships between the actors and the rationale behind process design decisions. The ConGolog framework will be used to formally specify the system behavior described informally in the *i** model. The ConGolog model will provide more detailed information about the actors, tasks, processes, and goals in the system, and the relationships between them. Complete ConGolog models are executable and this will be used to validate the specifications by simulation. To bridge the gap between *i** and ConGolog models, an intermediate notation involving the use of process specification annotations in *i** SR diagrams will be introduced. We will also propose a set of mapping rules that constrain the modeler to map the elements of the annotated SR diagram to appropriate ConGolog entities and ensures that the two models are consistent. Finally, we propose a methodology for the combined use of the *i** and ConGolog frameworks for early to late phase requirements engineering. We then illustrate the use of the methodology on an example application: meeting scheduling.

2 The *i** Modeling Framework

The *i** framework [Yu95, Yu97] was developed for modeling and analyzing organizations to help support business process reengineering and requirements engineering. The framework focuses on modeling intentional and strategic relationships between actors. It consists of two main components — the Strategic Dependency (SD) model and the Strategic Rationale (SR) model. The SD model is used to capture the network of relationships that hold among the actors. The SR model describes in more detail the alternative methods that the actors have for accomplishing their goals and tasks. It helps the modeler in understanding the existing processes, and in generating alternatives in order to reach a new process design that better addresses the organization's objectives.

A SD model is a graph consisting of nodes and links among the nodes. Each node represents an actor, and each link between the actors represents how one actor depends on another for something in order to accomplish a goal or task. The depending actor is called the depender, the actor who is depended upon by another is called the dependee, and the object on which the dependency relationship centers is called the dependum. An actor can be an agent, a role, or a position. An agent is a concrete, physical actor or subsystem. A role is a function in an organization. A position is a group of roles that is institutionalized in an organization. For example, in a meeting scheduling process, the meeting initiator (a role) may depend on a meeting scheduler agent to organize a meeting. The meeting initiator is a depender, the meeting scheduler is a dependee, and the goal "MeetingBeScheduled" is the dependum in a dependency relationship that the meeting initiator has on the meeting scheduler.

Based on the nature of the dependum involved, there are four types of dependencies: goal-, task-, resource-, and softgoal-dependencies. In a goal-dependency, the depender depends on the dependee to bring about some desired state. The dependee is free to decide how to achieve the goal on his own. A task-dependency specifies that the depender depends on the dependee to complete a certain task through some activities, i.e. how the task is to be performed, but not why. A resource-dependency specifies that the depender depends on

the dependee for the availability of an information or physical resource. A softgoal-dependency expresses that the depender depends on the dependee to achieve a softgoal, i.e. a goal that can be achieved to varying degrees, and needs to be optimized. In a dependency, the depender may be vulnerable in case that dependee fails to bring about the dependum. There are three degrees of strength of dependencies in the SD model: open (uncommitted), committed, and critical.

The SD model focuses on the intentional dependencies among actors beyond the usual understanding based on entity flows and activities. It helps the modeler obtain a deeper understanding of a process and identify what is the stake, for whom, and what impacts are likely if a dependency fails. But the SD model only describes why a process is structured in a certain way. It does not really support the process of suggesting, exploring, and evaluating alternative solutions. The Strategic Rationale (SR) model of *i** model addresses this.

The SR model is a graph consisting of four main types of nodes, goal, task, resource, and softgoal nodes, and two main types of links, means-ends links and task-decomposition links. Task-decomposition links describe how a task can be decomposed into subtasks or subgoals. Means-ends links specify how a goal may be achieved. They provide information about why an actor would perform a task, pursue a goal, need a resource, or want a softgoal. From the softgoals, the modeler can tell why one alternative may be chosen over others. In section 5, we give an example of how *i** is used to model a meeting scheduling application.

3 The ConGolog Modeling Framework

ConGolog [DLL00, LKMY99] is a framework for process modeling and agent programming. It is based on the situation calculus [MH69], a language of predicate logic for representing dynamically changing worlds. The ConGolog framework can be used to model *complex processes* involving loops, nondeterminism, concurrency, multiple agents, etc. Because it is logic-based, the framework can accommodate incompletely specified models, either in the sense that the initial state of the system is not completely specified, or that the processes involved are nondeterministic and may evolve in any number of ways.

A ConGolog model of a domain includes two components. One is the specification of the domain dynamics, i.e. how to model the state, what is the initial state of the domain, what actions can be performed, when the actions can be performed and what their effects are. Another is the specification of the process of interest, i.e. the behavior of the agents involved in the domain.

In ConGolog and the situation calculus, a dynamic domain is modeled in terms of the following entities:

- *Agents*: the agents involved in the system to be modeled; these are represented by constants, e.g. `ms1`, the meeting scheduler agent in our example application.
- *Primitive actions*: all changes to the world are taken to be the result of named primitive actions that are performed by some agent in the system; primitive actions are represented by terms, e.g. `acceptAgreementReq(Participant,MS,ReqID,Date)`, i.e. the `Participant` agent accepts the request `ReqID` from the `MS` agent to attend a meeting on `Date`.
- *Situations*: these correspond to possible world histories viewed as sequences of actions. The initial situation (where no actions have been executed) is represented by the constant `s0`. There is a distinguished binary function symbol `do` and a term `do(A,S)` denotes the situation which results from action `A` being performed in situation `S`. The sequence of actions in a history, and the order in which they occur, are obtained from a situation term by reading off its action instances from right to left. For example, `do(A3,do(A2,do(A1,s0)))` represents the history where `A1`, and then `A2`, and then `A3` are performed starting in the initial situation `s0`.

- *Fluents*: these are properties, relations, or functions whose value may change from situation to situation and which are of interest to the modeler; they are represented by predicate or function symbols that take a situation term as their last argument, e.g. `agreementReqRcvd(Participant,MS,ReqID, Date,S)`, i.e. `Participant` has received a request `ReqID` from `MS` to agree to hold a meeting on `Date` in situation `S`.

The dynamics of a domain are specified using three kinds of axioms:

- *Action precondition axioms*: these axioms state the conditions under which an action can be performed; these use the predicate `poss(A,S)`, which means that action `A` is possible in situation `S`. For example, in our meeting scheduling domain, we have the following precondition axiom:

  ```
  poss(acceptAgreementReq(Participant,MS,ReqID,Date),S) ≡
  agreementReqRcvd(Participant,MS,ReqID,Date,S) ∧
  dateFree(Participant,Date,S)
  ```

  This says that in situation `S`, `Participant` may perform the action of accepting a request `ReqID` from `MS` to hold a meeting on `Date` if and only if he has received a request to that effect and the date is free for him.

- *Successor state axioms*: these axioms specify how the fluents are affected by the actions in the domain. For example, in our meeting scheduling domain, we have the following successor axiom:

  ```
  agreementReqRcvd(Participant,MS,ReqID,Date,do(A,S)) ≡
  A = requestAgreement(MS,Participant,Date) ∧ reqCtr(S) = ReqID
  ∨ agreementReqRcvd(Participant,MS,ReqID,Date,S)
  ```

  This says that `Participant` has received a request `ReqID` from `MS` to agree to hold a meeting on `Date` in situation `do(A,S)` if and only if the action `A` is such a request and the value of the request counter is `ReqID` or if he had already received such a request in situation `S`.

  Successor state axioms can be generated automatically from a specification of the effects of primitive actions if we assume that they specify all of the ways that the value of the fluent may change. Successor state axioms were introduced in [Reiter91] and provide a solution to the frame problem. [LKMY99] describes a convenient high-level notation for specifying the effects (and preconditions) of actions and a tool that compiles such specifications into successor state axioms.

- *Initial situation axioms*: these axioms specify the initial state of the modeled system. For example, in our meeting scheduling domain, we might have the following initial situation axiom:

  ```
  participantTimeSchedule(yves,s0) = [10, 12]
  ```

  This says that agent `yves` is busy on the 10^{th} and 12^{th} in the initial situation.

The process of a system is specified procedurally in the ConGolog model. The `main` procedure will specify the whole system's behavior. Every agent also has an associated ConGolog procedure to represent its behavior in the system. The ConGolog framework includes constructs for conditionals, loops, nondeterminism, concurrency, etc. Communication between agents can be represented by actions that the sender agent performs, which affect certain fluents that the recipient agent has access to.

A process simulation and validation tool has been implemented. It uses an interpreter for ConGolog that has been implemented in Prolog. This implementation requires that the precondition axioms, successor state axioms, and axioms about the initial situation be expressed as Prolog clauses. Thus, simulation can only be performed for completely specified initial states.

A verification tool is also being developed. [DLL00] discusses applications of ConGolog in different areas, such as in robot programming, personal assistants, etc. [LKMY99] focuses on applications to process modeling and requirements engineering.

4 An Approach to the Combined Use of *i** and ConGolog

The *i** SR diagram notation allows many aspects of processes to be represented, but it is somewhat imprecise and the models produced are often incomplete. For instance, it is not specified whether the subtask in a task decomposition link has to be performed once or several times. In a ConGolog model, the process must be completely and precisely specified (although non-deterministic processes are allowed). We need to bridge this gap. To do this, we will introduce a set of *annotations* to SR diagrams that allow the missing information to be specified. We also want to have a tight mapping between the annotated SR (ASR) diagram and the ConGolog model, one that specifies which parts of each model are related. This allows us identify which parts of the ConGolog model need to be changed when the SR model is modified and vice versa. So we will require the modeler to define such a mapping. We want to ensure that the mapping respects the semantics of both frameworks, so we define a set of *mapping rules* that define what mappings are allowed. We will also propose a methodology for the combined use of the *i** and ConGolog frameworks.

4.1 SR Diagram Annotations

Two types of annotations are defined: composition annotations and link annotations. Composition annotations are applied to groups of decomposition links in the SR model. These annotations clarify how the linked subtasks/subgoals are to be composed in order to perform the super task/goal.

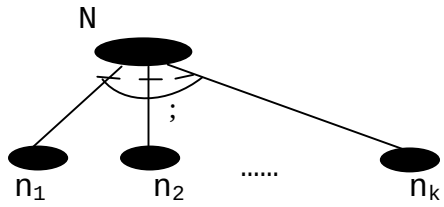


Figure 1: Sequence annotation applied to a group of decomposition links

There are four types of composition annotations: sequence ";", alternative "|", concurrency "||", and prioritized concurrency ">>". For example, if the sequence annotation ";" is put on the links as in Figure 1, the subtasks/subgoals are to be sequentially performed left to right to complete the decomposed task. We take sequence to be the default and generally leave it out.

Link annotations are applied to individual decomposition links connecting a task/goal with the linked subtask/subgoal. Link annotations describe under what condition the subtask/subgoal is to be performed and whether it should be done once or repeatedly. There are five types of link annotations: the while-loop annotation "*while(condition)", the for-loop annotation "*for(variable,listOfValues)", the interrupt annotation "*whenever(variableList,condition)", the if annotation "if(condition)", and the pick annotation "pick(variableList,condition)". The first three are for iterating the subtask/subgoal, and the last is for

nondeterministicaly picking values for the variables in the subtask that satisfy the condition. All of the annotations correspond to ConGolog operators and are taken to have the same meaning.

4.2 Mapping Rules

The modeler must define a mapping m from the elements of the annotated SR (ASR) diagram to the entities in the ConGolog model. We define mapping rules to ensure consistency between the ASR model and the ConGolog model. The mapping must respect the rules, which arise from the semantics of the two formalisms. There are two sets of mapping rules:

Mapping rules for SR diagram nodes

We define mapping rules for each of the five types of nodes in the ASR model, i.e. agent nodes, goal nodes, task nodes, role nodes, and position nodes. These ensure that the nodes are mapped into appropriate ConGolog entities.

For agent nodes, we have the following rule:

If n is an agent node, m(n)=<a, a_behavior>, where a is a ConGolog agent and a_behavior is a ConGolog procedure representing the behavior of the agent.

We use m_agent(n) to refer to the agent a and m_behavior(n) to refer to the agent's behavior a_behavior.

For role/position nodes, the rule is:

If n is a role/positiont node, m(n)=n_behavior, where n_behavior is a ConGolog procedure representing the behavior associated with the role/position.

For task nodes, we have the rule:

If n is a task node, m(n)=t, where t is a ConGolog procedure or primitive action representing the task.

For goal nodes, we have the rule:

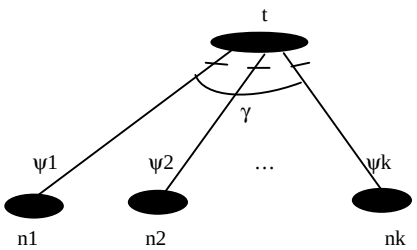
If n is a goal node, m(n)=<g, achieve_g>, where g is a ConGolog fluent (primitive or defined) and achieve_g is a ConGolog procedure specifying a selected set of means for achieving the goal which the agent will perform when he adopts it.

We use m_fluent(n) to refer to the fluent g and m_achieve(n) to refer to the procedure achieve_g.

Mapping rules for SR diagram links

There are mapping rules for each of the two types of links in the ASR model, task decomposition links and means-ends links. The mapping rule for task decomposition links is as follows:

If t is a task node that is decomposed into nodes n_1,..,n_k by task decomposition links, where a composition annotation γ is applied to the group of decomposition links and link annotations ψ_i are applied to the individual decomposition link between t and n_i, i.e.,



then m(t) must be a ConGolog procedure of the form:

```
proc(t(parameterList),
     ψ1'(m_proc(n1))
     γ'
     ψ2'(m_proc(n2))
     γ'
      …
     γ'
     ψk'(m_proc(nk))
). /* end of procedure*/
```

where γ ' (ψi ') is the ConGolog operator corresponding to the annotation γ (ψi) and m_proc(ni) is the ConGolog procedure associated with node ni, i.e. m(ni) if ni is a task node and m_achieve(ni) if ni is a goal node.

Thus, the body of the ConGolog procedure associated to a task node t must be exactly as specified by the decomposition defined for t in the ASR diagram. It should be straightforward to automate the translation.

The mapping rule for means-ends links (goal decomposition links) is very similar to the one above. Essentially, it requires that the m_achieve(g) procedure associated with a goal node g be as specified by the decomposition defined for g in the ASR diagram. There is one additional requirement: that the fluent associated with the goal m_fluent(g) be true when the procedure completes execution.

Dealing with Dependencies: Operationalization

The dependencies between agents, roles and positions in the SR model indicate that the depender depends on the dependee to accomplish one of his tasks or goals or to supply some resource. The *i** SR model generally abstracts over all details of the associated interactions between the agents (requests, communication/interaction protocols). The ConGolog model on the other hand, focuses on the operational aspects of processes rather than their strategic/social aspects. Thus in our approach, we will not represent dependency relationships themselves in the ConGolog model. Instead, we require that the modeler operationalize the dependencies in the ASR model, i.e. specify to a sufficient extent the tasks that must be performed by the depender and dependee in their interaction to ensure that the dependum is supplied. After this, the tasks and goals that result are mapped into ConGolog in the normal way. The details of how a dependency is operationalized depends on the particulars of the case and must be filled in by the modeler. An example is given in section 5.

Discussion

Various issues have not been completely resolved in the definition of our mapping rules, and should be studied further:

- How complete must the mapping be? Must all SR diagram nodes and links be mapped? Must all ConGolog procedures, actions, and fluents be mapped into? From a practical point of view, a possible answer is "complete enough to allow analysis through simulation or verification".
- How should parameters in procedures and goals be handled? In *i** diagrams, they are often absent, while in ConGolog, they are always listed explicitly. Perhaps we can think of them as present in *i** diagrams, but kept hidden unless explicitly made visible (a tool could easily support that).
- Goals have both a declarative and procedural interpretation. The associated procedure specifies a selected set of means for achieving the goal (usually not complete), and the procedure must achieve the goal to terminate. Is this treatment satisfactory?
- The diagram notation does not support well the distinction between a generic system and a system instance (e.g. one used in a particular simulation experiment). How do we extend it to capture this?
- We haven't distinguished between design goals and execution-time goals. But the distinction can be fuzzy. Should we distinguish, and if so, how?

The mapping rules can be viewed as giving a formal semantics to annotated SR diagrams by mapping this notation into ConGolog, a language which already has one. We believe that this semantics is largely consistent with the somewhat abstract axiomatic semantics for *i** developed in [Yu95]. As such, it could perhaps be viewed as a formal semantics for SR diagrams more generally. But one point where the two semantics diverge is with respect to completeness: in *i** , task/goal decompositions are generally not assumed to be complete, but in ConGolog and in our mapping rules they are assumed to be. Should we try to accommodate incompleteness? Should we distinguish between a set of task/goal decompositions and its completion? More study of these issues is required.

4.3 A Methodology for the Combined Use of the *i** and ConGolog Frameworks

Let us outline the methodology that we propose:

Step 1: Building the Strategic Dependency (SD) model for the application.
The SD model specifies the agents, roles, positions, and the intentional dependency relationships between them for the application.

Step 2: Building the Strategic Rationale (SR) model for the application.
The SR model identifies the tasks/goals/softgoals inside the agents, roles, and positions, the decompositions of the tasks/goals, and the contributions of the various alternatives to the softgoals. Dependency relationships are specified between nodes inside the agents/roles/positions.

Step 3: Building the Annotated Strategic Rationale (ASR) model for the application.
This involves the following substeps:

- Suppressing unnecessary information from the SR model. Softgoals and their associated links and dependencies are suppressed. Tasks/goals/dependencies associated with process alternatives other than the one(s) selected are suppressed.
- Relativizing goals. Goals that cannot always be achieved are weakened/conditionalized.
- Operationalizing the dependencies: The interactions (requests/protocols, etc.) through which the task/goal/resource dependencies are realized by the actors are specified.

- Filling out process details using annotations. How tasks and goals are to be composed is specified using annotations. Every task/goal will be decomposed into atomic subtasks/subgoals if applicable. This decomposition ensures that process of the system is specified precisely at a sufficient level of detail.

Step 4: Developing the initial ConGolog model.
The modeler maps elements in the ASR model into entities in the ConGolog model while conforming to the mapping rules; he completes the definition of the initial ConGolog model by specifying the fluents, primitive actions, precondition axioms, and successor state axioms for the system.

Step 5: Validating the ConGolog model by simulation and verification.
Given a specification of an initial state for the system, simulation experiments are performed on the developed ConGolog model and the results are used to validate and evaluate the model.

Iterate steps 1 to 5: Refining the *i and ConGolog models until objectives are met.**
Whenever the *i** model or ConGolog model has to be modified based on the results of the validation step, the modeler refines the corresponding part of the other model. This continues until the client's objectives are achieved.

Step 6: Producing the requirements specification document.

Note that steps 2, 3, and 4 could be performed in sequence, but need not be. Instead, the analyst might successively work on different tasks/goals, decomposing/refining them, and as he does this, introducing annotations and ConGolog domain specifications.

5 Case study: A Meeting Scheduling Process

This case study deals with a meeting scheduling application drawn from [Yu97]. The initial requirements for the process are expressed as "for each meeting request, to determine a meeting date and location so that most of the intended participants will be able to effectively participate". The alternative that will be selected for the process involves a computerized meeting scheduler agent (MS). Participants are assumed to maintain their own time schedule. The process operates roughly as follows. After receiving a meeting scheduling request from the initiator, the MS requests all the potential participants for information about their availability to meet during a date range provided by the initiator. Participants reply with a set of dates when they are available. If there is no date that suits all of the participants, the MS notifies the initiator and participants that it has failed to find a date to schedule the meeting. Otherwise, the MS selects such a date and asks all participants to agree to meet on this date. Participants will agree to the proposed meeting date if it is still available on their schedule at the time of the proposal. If all participants agree, they and the initiator are notified of the confirmed meeting date. If one of the participants rejects the date because it now occupied by other activities, then the MS informs all participants who have accepted the proposed date that it cancels the request, and then goes on to propose another date if there is one available. Otherwise it notifies the initiator that meeting scheduling has failed.

5.1 Building the Strategic Dependency Model

A Strategic Dependency (SD) model for this meeting scheduling process based one from [Yu97] appears in Figure 2 (we specialize the actors into agents, roles, and positions). The SD model of Figure 2 specifies the dependencies that actors have on each other, thus providing the modeler with a better understanding of the "whys" behind the process. Then alternatives can be developed to meet the needs of the organization.

In the model of Figure 2, there are four actor nodes: the meeting scheduler, which is an agent node, and the initiator, important participants, and participants, which are role nodes. Each link between these agents/roles represents how one agent/role depends on another for something. For example, when a meeting m is to be scheduled, the initiator depends on participants' attendance at the meeting. In the SD model, this is represented by a dependency link between the initiator and the participant. The initiator is the depender, the participant is the dependee, and "AttendsMeeting(p,m)" is the dependum.

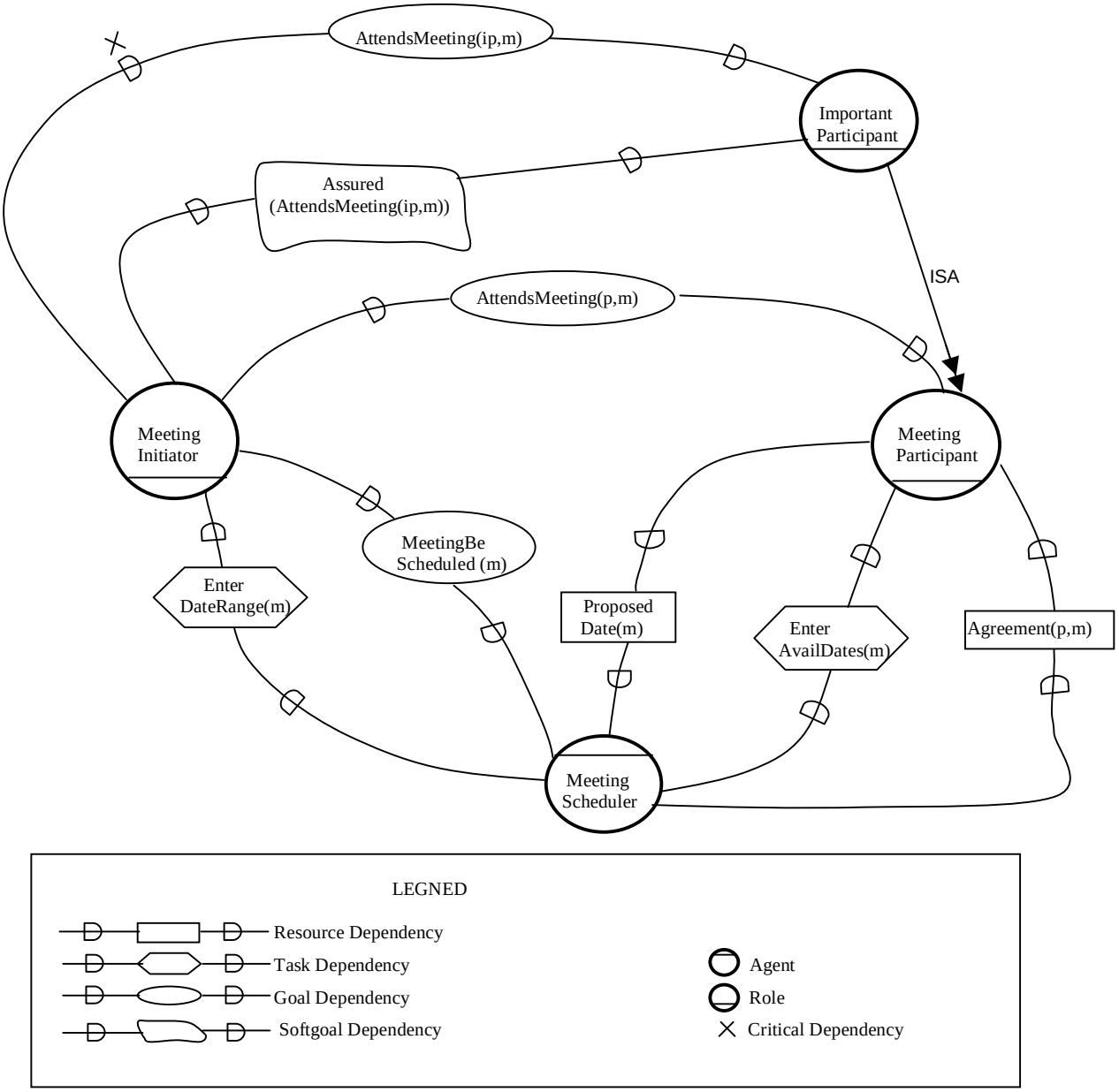


Figure 2. A Strategic Dependency model for the meeting scheduling process.

Different dependency types express different kinds of relationships between the depender and the dependee, involving different types of freedom and constraints. For example, the meeting initiator's dependency on the meeting scheduler to schedule a meeting can be modeled as a *goal dependency* "MeetingBeScheduled(m)". It is the meeting scheduler's responsibility to decide how to achieve the goal. In Figure 2, the *resource*

dependency “Agreement(p,m)”, the *softgoal dependency* “Assured(AttendsMeeting(ip,m))”, and the *task dependency* “EnterAvailDates(m)” describe other kinds of dependencies between actors.

The SD model is used to analyze the meeting scheduling process in terms of intentional relationships among actors in the desired system. This will help the modeler understand the opportunities and vulnerabilities for the actors. For example, the ability of a computerized meeting scheduler to achieve the goal “MeetingBeScheduled(m)” represents an opportunity for the meeting initiator not to have to achieve this goal himself. On the other hand, the initiator becomes vulnerable if the scheduler fails to achieve it.

5.2 Building the Strategic Rationale (SR) Model

In the SR model, “a more detailed level of modeling is performed by looking "inside" the actors to model internal relationships” [Yu97]. Intentional elements such as goals, tasks, resources, and softgoals are modeled not only as external dependencies, but also as internal elements linked by means-ends and task-decomposition relationships. The SR model in Figure 3 taken from [Yu97], elaborates on the relationships between the actors represented in the SD model of Figure 2.

For the meeting initiator, the goal of “MeetingBeScheduled” is an internal goal. The internal tasks “ScheduleMeeting” (by himself) and “LetSchedulerScheduleMeeting” are alternative means to achieve the goal “MeetingBeScheduled”. How the alternatives contribute to softgoals is also represented.

For the MS, the task “ScheduleMeeting” can be decomposed as follows: a subtask of obtaining the available dates from the participants “ObtainAvailDates”, a subgoal of finding an available date slot “FindAvailDateSlot”, a resource dependency of supplying a “ProposedDate”, and a subtask of obtaining agreement from the participants “ObtainAgreement”. The sub-elements of the task are represented as subgoals, subtasks, or resources depending on the way in which these elements can be accomplished. The subgoal “FindAvailDateSlot” can be achieved in various ways. “ObtainAvailDates” and “ObtainAgreement” on the other hand, are both subtasks that refer to specific ways of accomplishing these tasks.

For the participants, “FindAgreeableDateUsingScheduler” is an internal task, which can be decomposed as follows: entering available dates to the meeting scheduler, a resource dependency, and indicating agreement about attending the meeting on a given date, a subtask “AgreeToDate”.

5.3 Developing the Annotated SR Model

Next, we develop an annotated SR diagram that is a more detailed description of the process and that can be mapped into a ConGolog model. This diagram is quite large. In Figure 4, we only show the part concerned with the participant. The diagram is developed in stages:

Suppressing Unnecessary Information from the Initial SR Model
First, we simplify the SR diagram by suppressing softgoal nodes and any link between these and other nodes from the initial SR model; these are not really relevant to mapping the SR model into a ConGolog model. Since we want to focus on the scheduling of meetings and not on how participants attend them, we also suppress the “ParticipateInMeeting” and “AttendMeeting” nodes and the dependency between the latter and the node “OrganizeMeeting” in the initiator (see Figure 3). We also suppress the task nodes “ScheduleMeeting” and “FindAgreeableDateByTalkingToInitiator” because the selected alternative is to schedule a meeting using the computer-based meeting scheduler. The goal node “Agreeable(Meeting, Date)” is also suppressed because we take the participants to just passively answer requests from the meeting scheduler and rely on the meeting scheduler to find an agreeable date.

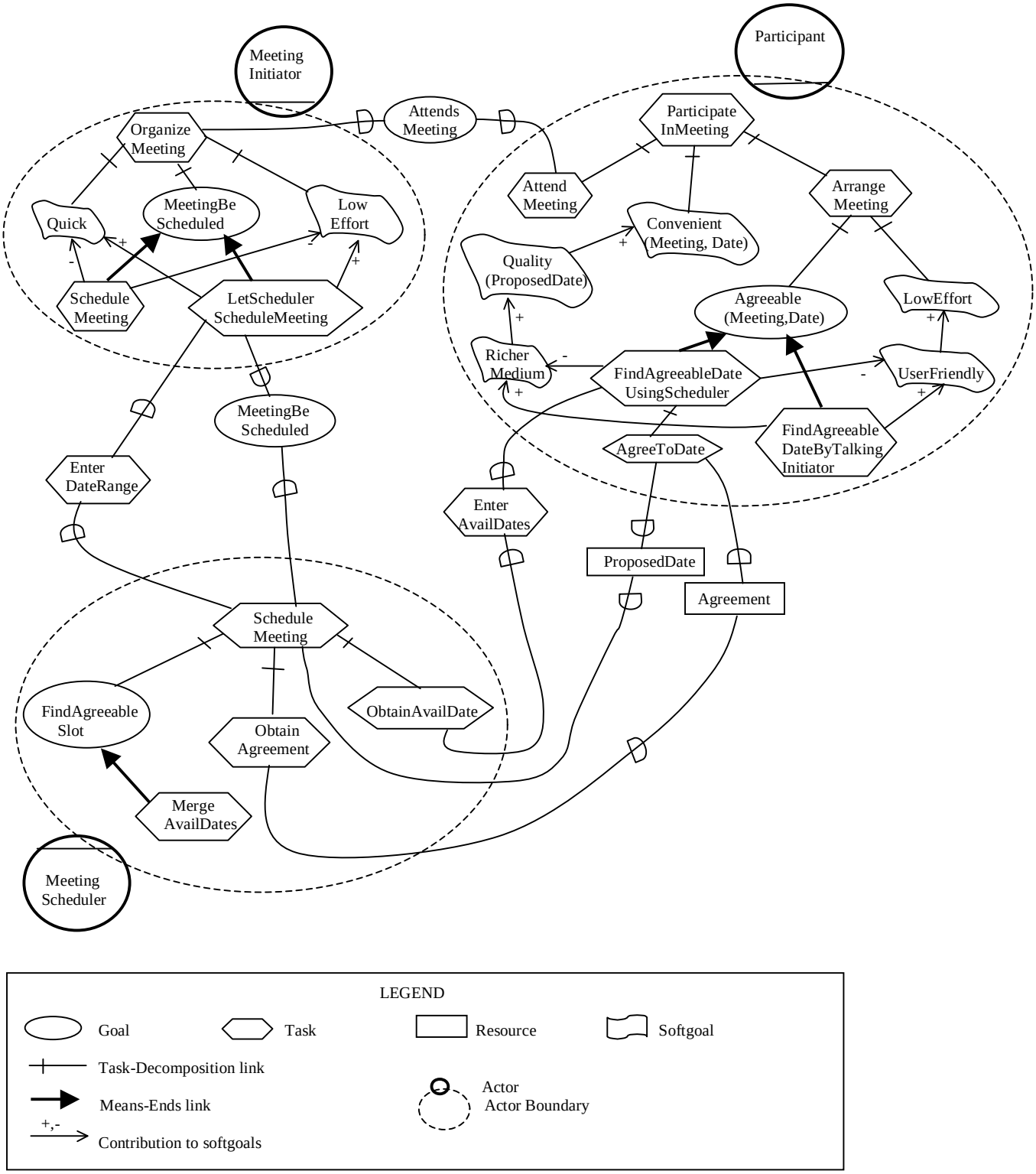


Figure 3. An initial Strategic Rationale model of the meeting scheduling process.

Relativizing Goals that Cannot Always Be Achieved

In Figure 3, the goal "MeetingBeScheduled" is only achievable when there is a date on which all participants can attend the meeting. To allow for failure, we weaken the goal to "MeetingBeenScheduledIfPossible", which is achieved once an attempt to schedule the meeting has been made. The means of achieving this goal are refined into "TryScheduleAMeeting" and "TryObtainAgreement" in the MS and "TryArrangeMeetings" in the participant.

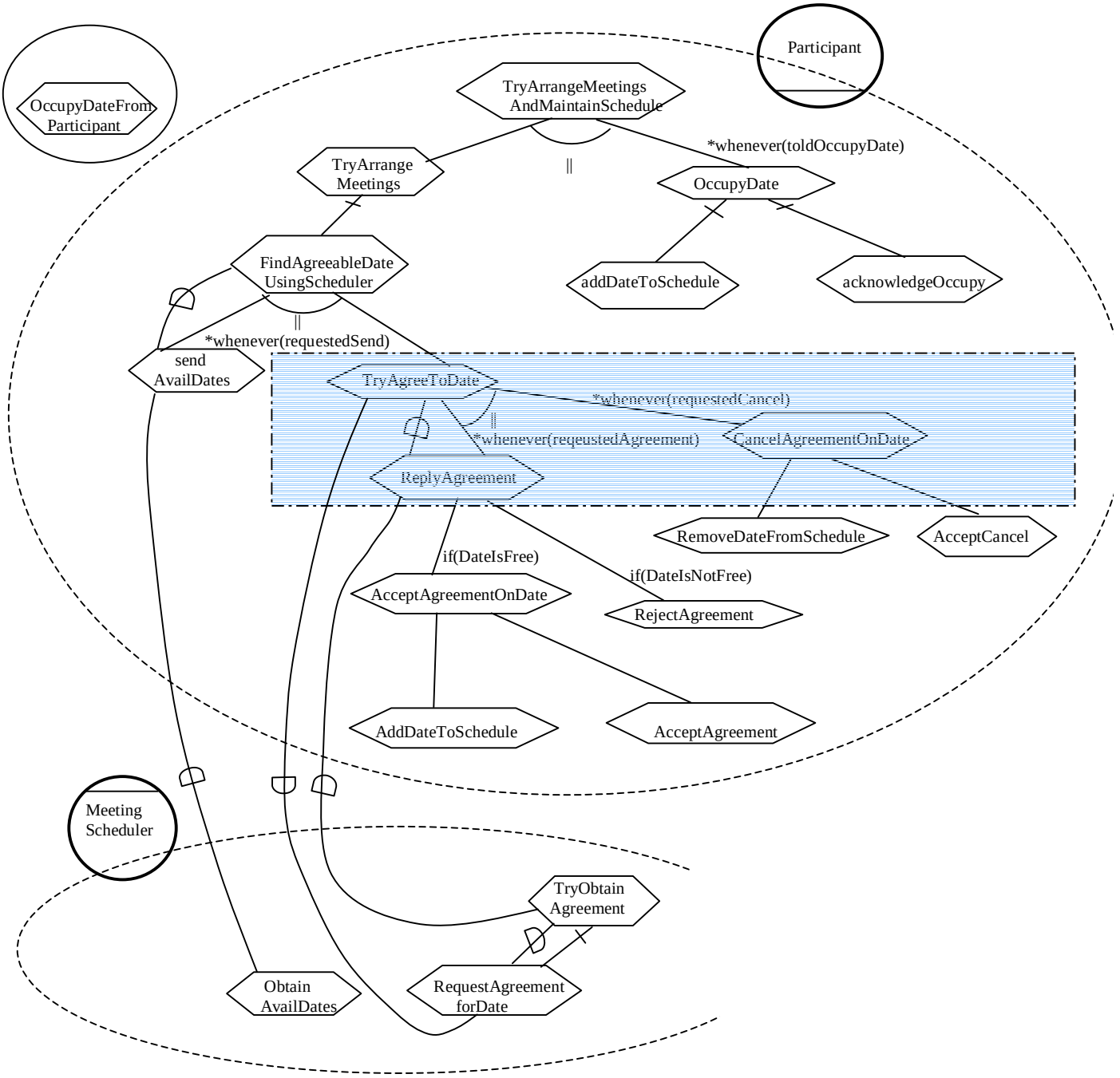


Figure 4. The annotated SR model for the participant.

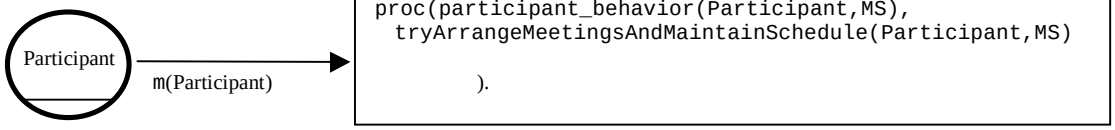
Operationalizing Dependencies
As discussed earlier, this involves making explicit the interaction tasks involved in the depender providing the dependum to the dependee. For example, the task dependency "EnterAvailDates" in Figure 3 has been operationalized by introducing a task "SendAvailDates" in the participant (the dependee) which is performed whenever it receives a request from the meeting scheduler. A task that makes such a request has also been introduced in the MS (a subtask of "ObtainAvailDates" not shown in Figure 4).

Filling Out Process Details Using Annotations
We add annotations to specify how subtasks are composed and whether they are iterated or conditional. For instance, in the shadowed rectangle area of Figure 4, the task node "TryAgreeToDate" is decomposed into two subtask nodes "ReplyAgreement" and "CancelAgreementOnDate", and the group of decomposition links is labeled with the concurrency annotation "||", meaning that the two subtasks will be performed concurrently. The link connecting the node "ReplyAgreement" with "TryAgreeToDate" is accompanied by the annotation "*whenever(requestedAgreement)", which means that whenever there is a request for agreement to a date from the MS, the task "ReplyAgreement" will be performed.

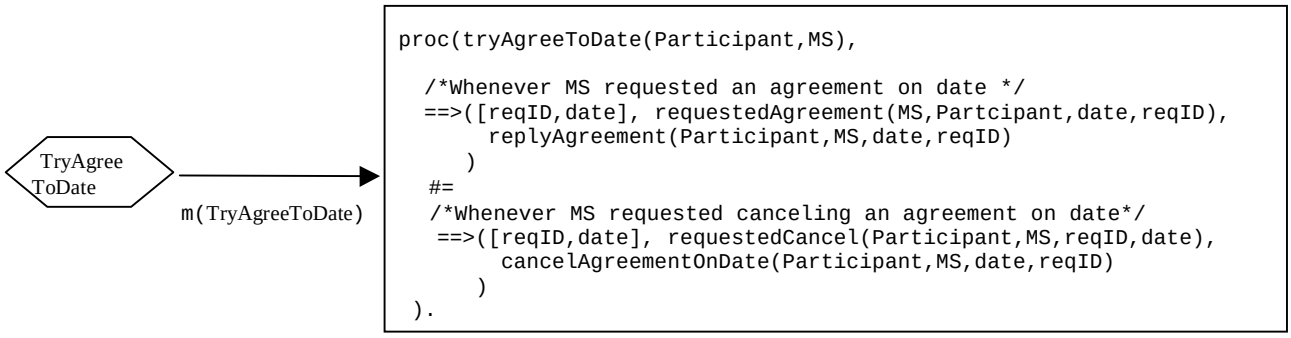
5.4 Developing the ConGolog Model

Here, the modeler must map entities in the ASR diagram into corresponding elements of a ConGolog model and complete the development of this model. We will give some examples describing how the mapping rules are applied to the entities in the annotated SR model to obtain the initial ConGolog model. First, all nodes must be mapped.

For example, the role node "Participant" is mapped into a ConGolog procedure specifying its behavior as follows:



For a second example, consider the task node "TryAgreeToDate" in the shadowed rectangle area of Figure 4. It is mapped into a ConGolog procedure specifying how the task is performed:



All links must also be mapped. As discussed earlier the task node "TryAgreeToDate" is decomposed into subtasks "ReplyAgreement" and "CancelAgreementOnDate" with annotations specifying that the subtasks

are to be performed concurrently and whenever an associated condition holds. The body of the *ConGolog* procedure "`TryAgreeToDate`" is composed exactly as specified by the annotated decomposition links in the ASR diagram.

As we are mapping elements of the ASR diagram into the ConGolog model's process specification, we must also begin to specify the domain dynamics. Primitive actions and fluents will be introduced to model relevant features of the domain. Successor state axioms will be specified to model how the primitive actions affect the fluents. Let us illustrate this by giving a few examples. The ConGolog domain specification includes:

- A primitive action `acceptAgreementReq(Participant,MS,ReqID,Date)`, that models the action of the `Participant` to accept the request `ReqID` from the `MS` to attend a meeting on the `Date`.

- An exogenous action `occupyDateFromParticipant(Participant,Date)`, that models an action by an agent outside the system which causes the `Participant` to occupy `Date` on his schedule.

- A predicate fluent `occupyAcknowledged(Participant,Date)`, that models the fact that the occupation of the `Date` on the `Participant`'s time schedule has been acknowledged (in response to the exogenous action described above).

- A functional fluent `participantTimeSchedule(Participant)`, whose value is the time schedule of the `Participant`.

- A defined fluent `waitingForAgreemeentAnswer(MS,Participant,Date,Meeting)`, that models the fact that the `MS` is waiting for the `Participant` to agree to have the `Meeting` on `Date`.

For each primitive fluent, we specify a successor state axiom that captures how it is affected by the primitive actions in the domain. For example, the successor axiom for the fluent `occupyAcknowledged(Participant,Date)` is as follows:

```
holds(occupyAcknowledged(Participant,Date),do(A,S)):-
   A = acknowledgeOccupy(Participant,Date);
   holds(occupyAcknowledged(Participant,Date), S).
```

This says that a request to occupy a `Date` on `Participant`'s schedule has been acknowledged in the situation that is the result of doing action `A` in situation `S` if and only if action `A` is to acknowledge this occupation, or if the occupation has already been acknowledged in situation `S`. Note that here, we are using the notation of the Prolog implementation of ConGolog.

We also specify an action precondition axiom for each primitive action. For example, the precondition axiom for the action `acceptAgreementReq(Participant,MS,ReqID,Date)` is as follows:

```
poss(acceptAgreementReq(Participant,MS,ReqID,Date),S):-
   holds(agreementReqRcvd(Participant,MS,ReqID,Date),S),
   holds(dateFree(Participant,Date),S).
```

This says that the action can be performed in situation `S` if `Participant` has received a request `ReqID` from `MS` to hold a meeting on `Date` and `Date` is free for him.

5.5 Validating and Evaluating the ConGolog Model by Simulation

As mentioned, complete ConGolog models can be executed to run process simulation experiments. To do this, the modeler must first specify an instance of the overall system. We do this by defining a `main` procedure, for example:

```
proc(main,[
     initiator_behavior(init1,ms1)#=
     meetingScheduler_behavior(ms1,init1)#=
     participant_behavior(yves,ms1)#=
     participant_behavior(paige,ms1)#=
]).
```

Here, there is an initiator agent `init1`, a meeting scheduler `ms1`, and two participants `yves` and `paige`. `#=` is the concurrent execution operator in the ConGolog implementation.

The modeler must also specify the initial state of the system. Here, the possible meeting dates are represented as integers in order to simplify the explanation. Initially the time schedule for participant `paige` is `[11, 12, 14]`, i.e. `paige` is busy on the 11th, 12th, and 14th, and the time schedule for participant `yves` is `[10, 12]`, i.e. `yves` is busy on the 10th and 12th. Initiator `init1` wants to schedule a meeting with `paige` and `yves` on the 12th or 14th. Then the modeler can execute the `main` procedure to obtain a simulation trace. The simulation trace that will be obtained from this instance of the system is as follows:

```
startInterrupts  /* start interrupts in initial situation */
requestScheduleMeeing(ini1,ms1,[paige,yves]) /* ini1 requests ms1 to schedule a
                                                meeting with paige and yves */
requestEnterDateRange(ms1,ini1,1) /* ms1 requests ini1 to enter the possible
                                     date range for meeting no. 1 */
enterDateRange(ini1,ms1,1,[12,14]) /* ini1 enters Feb. 12, 14 as possible
                                      meeting dates */
obtainAvailDates(ms1,yves,1) /* ms1 requests available dates from all
                                participants */
obtainAvailDates(ms1,paige,1)
sendAvailDates(yves,ms1,2,[…])  /* yves sends his available dates */
sendAvailDates(paige,ms1,1,[…]) /* paige sends his available dates */
setAllMergedlist(ms1,1,[]) /* ms1 finds merged available dates to be empty */
notifyFail(ms1,ini1,1,[paige,yves]) /* ms1 notifies ini1 , yves, and paige that
                                       it has failed to schedule meeting no. 1 */
notifyFail(ms1,paige,1,[paige,yves])
notifyFail(ms1,yves,1,[paige,yves])
```

Generally, this step of the methodology involves finding gaps or errors in the specification by simulating the processes. Alternative specifications can be also compared.

5.6 Refining the *i** and ConGolog Models and Producing the Requirements Document

The above five steps will be repeated if errors or inadequacies are found in the process specified or in the way it is modeled. If after specifying the ConGolog model and performing simulation experiments, we determine that the *i** model lacks some element of the desired requirements, it will be modified appropriately. Similarly if we find that the ConGolog model needs to specify additional details or aspects of

the *i** model, modifications to it will be made. Once a satisfactory model of the required system has been developed, a requirement specification document is produced.

6 Conclusion

In this paper, we have proposed an approach to requirements engineering that involves the combined use of the *i** and ConGolog frameworks. This allows the requirements engineer to exploit the complementary features of the two frameworks. *i** can be used to model social dependencies between agents and perform an analysis of opportunities and vulnerabilities; business goals and the rationale behind process designs can also be modeled and tradeoffs can be analyzed. ConGolog can be used to model complex processes formally; models can be validated and alternative processes can be compared through simulation and verification. Both graphical/informal and textual/formal notations are used, which supports a progressive specification process and helps in communicating with the client.

The approach we have proposed for integrating the two frameworks involves the following elements:

- *i** SR diagrams have been extended with a rich set of *process specification annotations*. Using annotated SR diagrams, the modeler can provide a precise specification of the processes of interest, and the resulting diagrams can then be mapped directly into a ConGolog model.
- A set of *mapping rules* has been defined to constrain the modeler to map elements of the annotated SR diagram into appropriate entities in the ConGolog model and ensure that the models are consistent. This allows us to trace corresponding elements in the two models when changes are made.
- A *methodology* has been developed to guide the analyst in specifying requirements using the combined *i** and ConGolog frameworks.

Our approach and the case study are described in more detail in [Wang01]. She also tests the methodology in a second case study involving a mail-order business application taken from [Bissener97].

There are various agent-oriented or goal-oriented requirements engineering frameworks in existence that are related to ours. One is ALBERT-II (Agent-Oriented Language for Building and Eliciting Real-Time Requirements) [DuBois95], a formal framework designed for specifying distributed real-time systems. ALBERT-II is based on temporal logic. Agents' states and behavior are specified through constraints expressed in the logic-based notation. Which aspects of agents' states or actions are known/visible to other agents is also specified formally through "cooperation constraints". Typical patterns of constraints are identified to support the analyst in requirements elaboration. In [YDDM97] and [Bissener97], the combined use of *i** and ALBERT-II for requirements engineering is investigated. The approach proposed in these papers is very different from ours; there is no attempt to develop an intermediate notation to enable a direct mapping from *i** to the ALBERT-II formal framework.

In work done independently from ours, Gans et al. [GJKL01] have also extended *i** SR diagrams and combined them with ConGolog to support the modeling and analysis of trust in agent networks. Their extended SR notation is somewhat different from ours. For instance, they represent task preconditions explicitly and interpret them as trigger conditions, rather that use *whenever/interrupt annotations. We plan to do a comparative study of the two extended SR notations. There has also been work on combining *i** SD diagrams with a formal notation based on temporal logic and analyzing them using model checking techniques [FPMT01].

Also related is the KAOS framework [DVF93], which focuses on the formal modeling of functional and non-functional requirements. The framework addresses issues in requirements acquisition, i.e. goal-directed, scenario-directed, and viewpoint-directed strategies, and the reuse of requirements specifications. There has

also been much work on frameworks targeted at agent-oriented system design, for instance, [WJK99] and [BDJT95].

Further work is necessary to fully realize the benefits of the proposed approach. We would like to develop a computerized tool to help the analyst in completing the steps of the methodology. Once a user-friendly tool has been developed, broader use could be achieved. We are also working on verification methods and tools for the ConGolog framework [LS99] and on integrating them in our methodology. The methodology should also be tested on realistic projects. Moreover, we would also like to extend the methodology to the design phase of system development. Finally, we would like to refine the methodology to better model agent's mental states — what agents know and want. For this, we will use an extended version of the ConGolog framework [SL01, SSL98, LS99, LLR99] that explicitly represents agents' knowledge and goals (using modal operators) and their dynamics, i.e., how they are affected by communication actions (e.g. inform, request, cancel-request, etc.) and perception actions.

References

[Bissener97] Bissener, M., *A Proposal for a Requirements Engineering Method Dealing with Organizational, Non-Functional, and Functional Requirements*, M.Sc. thesis, Dept. of Computer Science, University of Namur, Namur, Belgium, 1977.

[BDJT95] Brazier, F., Dunin-Keplicz B., Jennings, N.R., and Treur, J., Formal Specifications of Multi-Agents Systems: A Real-World Case Study, 25-32, *Proc. of the 1st International Conference on Multi-Agent Systems (ICMAS'95)*, San Francisco, CA, June, 1995.

[DLL00] De Giacomo, G., Lespérance, Y., and Levesque. H.J., ConGolog, a concurrent programming language based on the situation calculus, *Artificial Intelligence*, **121**, 109-169, 2000.

[DuBois95] Du Bois, P., *The Albert II Language - On the Design and the Use of a Formal Specification Language for Requirements Analysis*, Ph.D. thesis, Dept. of Computer Science, University of Namur, Namur, Belgium, 1995.

[DVF93] Dardenne, A., Van Lamsweerde, A., and Fickas, S., Goal-Directed requirements Acquisition, *Science of Computer Programming*, **20**, 3-50, 1993.

[FPMT01] Fuxman, A., Pistore, A., Mylopoulos, J., Traverso, P., Model Checking Early Requirements Specifications in Tropos, submitted to the Fifth IEEE International Symposium on Requirements Engineering (RE01), August, 2001, Toronto, Canada, available at http://www.cs.toronto.edu/km/tropos/.

[GJKL01] Gans, G., Jarke, M., Kethers, S., and Lakemeyer, G., Modeling the Impact of Trust and Distrust in Agent Networks, to appear in *Proc. of the 3rd International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2001)*, Interlaken, Switzerland, June, 2001.

[JW98] Jennings, N.R. and Wooldridge, M. (Eds.), *Agent Technology: Foundations, Applications, and Markets*, Springer-Verlag, Berlin, 1998.

[LKMY99] Lespérance, Y., Kelley, T.G., Mylopoulos, J., and Yu. E.S.K., Modeling Dynamic Domains with ConGolog, in *Advanced Information Systems Engineering, 11th International Conference, CAiSE-99, Proceedings*, 365-380, Heidelberg, Germany, June 1999, LNCS vol. 1626, Springer-Verlag, Berlin.

[LLR99] Lespérance, Y., Levesque, H.J., and Reiter, R., A Situation Calculus Approach to Modeling and Programming Agents, in M. Wooldridge and A. Rao, eds., *Foundations of Rational Agency*, 275-299, Kluwer, 1999.

[LS99] Lespérance, Y. and Shapiro, S., On Agent-Oriented Requirements Engineering, position paper, *International Workshop on Agent-Oriented Information Systems (AOIS'99)*, Heidelberg, Germany, June 1999.

[MH69] McCarthy, J. and Hayes, P., Some Philosophical Problems from the Standpoint of Artificial Intelligence, in Meltzer, B. and Michie, D., eds., *Machine Intelligence*, **4**, 463-502, Edinburgh University Press, Edinburgh, UK, 1979.

[Reiter91] Reiter, R. The Frame Problem in the Situation Calculus: A Simple Solution (Sometimes) and a Completeness Result for Goal Regression, in *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, V. Lifschitz, ed., 359-380, Academic Press, San Diego, CA, 1991.

[SL01] Shapiro, S. and Lespérance, Y., Modeling Multiagent Systems with the Cognitive Agents Specification Language - A Feature Interaction Resolution Application, to appear in Castelfranchi, C. and Lespérance, Y., eds., *Intelligent Agents Volume VII - Proceedings of the 2000 Workshop on Agent Theories, Architectures, and Languages (ATAL-2000)*, LNAI, Springer-Verlag, Berlin, 2001.

[SLL98] Shapiro, S., Lespérance, Y., and Levesque, H.J., Specifying Communicative Multi-Agent Systems with ConGolog, *in Agents and Multi-Agent Systems - Formalisms, Methodologies, and Applications*, W. Wobcke, M. Pagnucco, and C. Zhang, eds., 1-14, LNAI, Springer-Verlag, Berlin, 1998.

[Wang01] Wang, X., *Agent-Oriented Requirements Engineering Using the ConGolog and i* Frameworks*, M.Sc. thesis, Dept. of Computer Science, York University, Toronto, ON, Canada, 2001, to appear.

[WJK99] Wooldridge, M.J., Jennings, N.R., and Kinny, D., A Methodology for Agent-Oriented Analysis and Design, in O. Etzioni, J.P. Muller, and J. Bradshaw, eds., *Agents '99: Proceedings of the 3rd International Conference on Autonomous Agents*, Seattle, WA, May, 1999.

[Yu97] Yu, E.S.K., 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.

[Yu95] Yu, E.S.K., *Modelling Strategic Relationships for Process Reengineering*, Ph.D. thesis, Dept. of Computer Science, University of Toronto, Toronto, ON, 1995.

[YDDM97] Yu, E.S.K., Du Bois, P, Dubois E., and Mylopoulos, J., From Organization Models to System Requirements – A "Cooperating Agents" Approach, *in Cooperative Information Systems: Trends and Directions*, M. P. Papazoglou and G. Schlageter, eds., 293-312, Academic Press, 1997.