# Deliberation in a Modeling and Simulation Environment for Inter-organizational Networks

Günter Gans[1], Matthias Jarke[1,2], Gerhard Lakemeyer[1], and Dominik Schmitz[1]

[1] RWTH Aachen, Informatik V, Ahornstr. 55, 52056 Aachen, Germany
[2] Fraunhofer FIT, Schloss Birlinghoven, 53754 Sankt Augustin, Germany
{gans,jarke,lakemeyer,schmitz}@cs.rwth-aachen.de

**Abstract.** Inter-organizational networks of people, information and communication systems are often described by the interplay between individual goals and actions and the strategic dependencies among individuals and subgroups. Our research aims at improving requirements engineering for such networks by not just representing these goals and dependencies statically, but also by studying the dynamic interactions between both. In previous work, we proposed the prototype environment SNet for the representation and dynamic evaluation of agent-based designs for inter-organizational networks. A key feature of SNet was the automatic translation of extended i* models into the action language ConGolog. While this allowed the simulation of agent networks specified in i*, the resulting agents were purely reactive, which limits the usefulness of the system, in particular as a decision-support tool for network members, who need to evaluate the utility of different courses of action. In this paper we propose to remedy the situation by explicitly incorporating deliberation into the agent design of SNet. At the level of i*, deliberation is represented in terms of goals which are satisfiable by different tasks or agents. Utilities are modeled, in part, using the existing concept of softgoals, which are given a quantitative interpretation. At the level of ConGolog, decision-theoretic features are built into the interpreter, which drives the simulations, and the process of delegating tasks to other agents is explicitly represented.

## 1 Introduction

Given the need to understand and evaluate business processes, a number of formal approaches have been proposed over the years. Some of these [Sch94],[OSS94], and [PJ96] support the simulation of such processes thus enabling the assessment of the effects of different business strategies.

It has been argued that these methods do not suffice to adequately model inter-organizational networks, which are comprised of human, organizational, and technological actors. A crucial aspect of these networks are the interdependencies among the various actors, which result, for example, from the need to delegate certain activities, which in turn requires a certain level of trust between the (human) members of the network. The graphical modeling language i* [Yu95],

J. Eder and M. Missikoff (Eds.): CAiSE 2003, LNCS 2681, pp. 242–257, 2003.

which was developed for early requirements engineering, has proven to be particularly suitable as a modeling tool in this context because it explicitly deals with dependency relations, besides other notions like actors, goals, resources, and tasks. To capture the dynamic aspects of agent networks, i* was recently extended to include a linear time logic, which together with a model checker allows to verify whether certain states of a network are reachable [FPMT01]. While this approach is perhaps best suited for debugging purposes, it seems desirable to include mechanisms which allow simulations of different scenarios within a network, which would be useful in analyzing its properties and could serve as a decision-support tool for network members. For this purpose we [GJK+01, GJLV02] and Wang and Lespérance [WL01] independently proposed to amalgamate i* and the action formalism ConGolog [dGLL00]. In our system called SNet, extended i* diagrams are automatically translated into executable ConGolog programs, supported by the metadata manager ConceptBase [JEG+95].

One major drawback of the existing approaches, including our own, is that the agents themselves are limited in the kinds of choices they are able to make during a simulation. In our case, these are essentially determined by the values of certain parameters supplied before and during a simulation. In other words, during a particular simulation run, the agents simply commit reactively to the choices required by the design of the agent network and do not themselves engage in deliberation about what the most appropriate course of action might be.[1] Including such a facility in the agents has a number of advantages. For one, it allows for more flexibility in agent design. For another and perhaps more importantly, the decision-making process itself seems to be a crucial aspect of human actors, which should be reflected in the model so that its effects can be studied formally. Finally, such an approach promises to lead to more intelligent designs of technological agents within inter-organizational networks.

In order to achieve deliberation, we make use of the fact that i* already supports the notion that a goal can be achieved by more than one task. While in the old version of SNet, a goal was treated merely as postcondition of the tasks that achieve them, we now focus on the *intentional* force of a goal in that it triggers the process of finding the currently best way or plan of achieving it.

As we already mentioned, an important aspect of inter-organizational networks is that agents depend on each other in performing certain tasks or subtasks, which gives rise to *delegation*. Moreover, there are often a number of possible alternatives for delegation which are often best resolved by some form of *bidding* or *negotiations* between a delegator and candidate delegatees. Our framework includes an explicit model of bidding and delegation.

In order to decide among different courses of action, it is important to rank them according to some measure of *utility*. To this end we make use of another construct offered by i*, *softgoals*. Roughly, softgoals can be thought of as desirable properties such as the level of comfort of car seats produced by a company. While in Yu's original work, softgoals were treated in a rather vague way, we

[1] Strictly speaking, [WL01] allow choices using ConGolog's notion of nondeterministic actions. However, this results in a commitment to the first choice that works, which may not be the most reasonable one to take.

have chosen to give them a precise quantitative interpretation so that they can be used as criteria from which utility measures can be derived. For example, when two car seat producers assign different values to the comfort level they offer for their seats, this can be used by the car manufacturer in deciding who to work with. Based on earlier work on trust and distrust in agent networks [GJK+01], we also take into account the level of trust between agents when computing utilities, which is particularly important when deciding among different possibilities for delegation.

Roughly, our overall approach then is this: taking our previous work on SNet as the starting point, we provide a new mapping between i* diagrams and the action language ConGolog; when goals can be achieved in more than one way, this will lead to ConGolog programs with nondeterministic choices. During execution, these choices will first be evaluated and compared, resulting in a plan with highest utility ready for execution. During evaluation, bidding may occur, which itself may trigger plan evaluations and bidding by other agents such as subcontractors.

The rest of the paper is organized as follows. In Section 2, we briefly introduce the foundations of SNet, i*, Congolog, and the mapping between the two. In Section 3, we present our approach to deliberation by mapping i*-models into suitable programs of ConGolog, whose evaluation includes a form of decision-theoretic planning. In Section 4, we consider implementation issues and we end the paper with a brief discussion and an outlook on future work.

## 2 The Modeling and Simulation Environment SNet

In this section, we very briefly go over those parts of SNet which are relevant to this paper. The reader is referred to [GJLV02] for a more comprehensive account. Before we begin, let us introduce a simplified fragment of an inter-organizational network taken from the automobile industry, which we use as a running example. We consider a car producer together with subcontractors producing seats, rims, or headlights. We are interested in scenarios where, for example, the car producer wants to build a new car and thus needs seats and has to decide which subcontractor to appoint. The appointment possibly depends on the type of car the car producer wants to build, say a sportive car or a limousine.

### 2.1 An Extended Version of i*

The i* framework is a graphical language and includes the *strategic dependency (SD)* model for describing the network of relationships among actors and the *strategic rationale (SR)* model, which, roughly, describes the internal structure of an agent in terms of tasks, goals, resources, etc. Compared to Yu's original formulation we added a few new features to SR models such as task preconditions. Figure 1 shows part of an extended SR model of the *SeatProducerA* car seat producer with a focus on the tasks (denoted as hexagons) to be carried out.

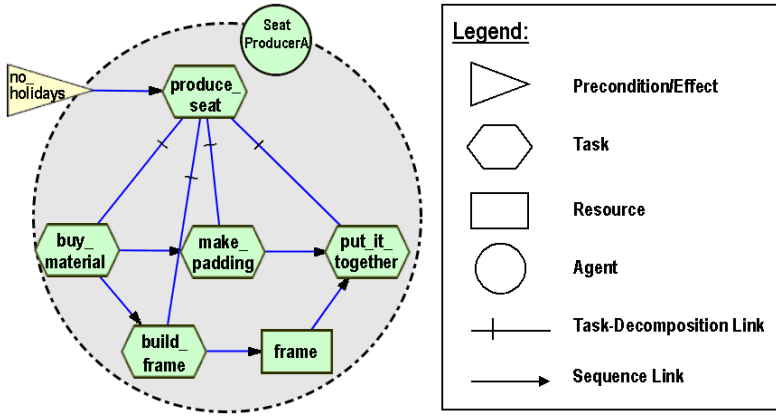


**Fig. 1.** Modeling in i*/SNet

The main task *produce_seat* is decomposed into four subtasks, which are partially ordered using *sequence links*.[2] Resources (denoted as rectangles) are entities which can be created, handed over between tasks and consumed. In the example, the resource *frame* is produced by the task *build_frame* and afterwards consumed by the task *put_it_together*, which depends on the existence of frames. Tasks can have preconditions and effects (both denoted as triangles). In Figure 1, the precondition *no_holidays* restricts the execution of *produce_seat* to regular working days.

### 2.2 Introduction to ConGolog

We can only present a very short overview over the logic-based high-level language ConGolog. ConGolog is based on the situation calculus, an increasingly popular language for representing and reasoning about the preconditions and effects of actions [McC63]. It is a variant of first-order logic,[3] enriched with special function and predicate symbols to describe and reason about dynamic domains. We will not go over the language in detail except to note the following features: all terms in the language are one of three sorts, ordinary objects, actions or situations; there is a special constant $S_0$ used to denote the *initial situation*, namely that situation in which no actions have yet occurred; there is a distinguished binary function symbol *do* where $do(a, s)$ denotes the successor situation to $s$ resulting from performing the action $a$; relations whose truth values vary from situation to situation are called *relational fluents*, and are denoted by predicate symbols taking a situation term as their last argument; similarly, functions varying across situations are called *functional fluents* and are denoted

[2] Sequence links were not present in the original SNet, but were since added for convenience (see also [WL01] for a similar construct).

[3] Strictly speaking, a small dose of second-order logic is required as well, an issue which should not concern us here.

analogously; finally, there is a special predicate $Poss(a, s)$ used to state that action $a$ is executable in situation $s$.

Within this language, we can formulate theories which describe how the world changes as the result of the available actions. One possibility is a *basic action theory* of the following form [Rei01]:

- Axioms describing the initial situation, $S_0$.
- Action precondition axioms, one for each primitive action $a$, characterizing $Poss(a, s)$. For example, the fact that a robot can only pick up an object if it is next to the object and it is not holding anything can be formalized as follows: $Poss(pickup(r, x), s) \equiv NextTo(r, x, s) \wedge \forall y.\neg Holding(r, y, s)$. We use the convention that free variables are implicitly universally quantified.
- Successor state axioms, one for each fluent $F$, stating under what conditions $F(x, do(a, s))$ holds as a function of what holds in situation $s$. These take the place of the so-called effect axioms, but also provide a solution to the frame problem. As an example, consider a simple model of time which progresses in a discrete fashion by 1 unit as a result of a special action *clocktick*. The time of a situation can then be specified with the help of a fluent $time(s)$ and the following successor state axiom:

$$time(do(a, s)) = t \equiv a = clocktick \wedge t = time(s) + 1 \vee a \neq clocktick \wedge t = time(s)$$

  We remark already here that we are using a temporal version of the situation calculus with discrete linear time, where time advances as specified by this axiom.
- Domain closure and unique-name axioms for actions.

ConGolog [dGLL00], an extension of Golog [LRL+97], is a language for specifying complex actions (high-level plans). It comes equipped with an interpreter which maps these plans into sequences of atomic actions assuming a description of the initial state of the world, action precondition axioms and successor state axioms for each fluent.

| | |
|---|---|
| $\alpha$ | primitive action |
| $\phi?$ | test action |
| $[\sigma_1, \sigma_2, \ldots, \sigma_n]$ | sequence |
| if $\phi$ then $\sigma_1$ else $\sigma_2$ | conditional |
| while $\phi$ do $\sigma$ | loop |
| $ndet(\sigma_1, \sigma_2)$ | nondeterministic choice of actions |
| $pi(x, \sigma)$ | nondeterministic choice of arguments |
| $star(\sigma)$ | nondeterministic iteration |
| $conc(\sigma_1, \sigma_2)$ | concurrent execution |
| $pconc(\sigma_1, \sigma_2)$ | prioritized concurrent execution |
| $interrupt(\phi, \sigma)$ | triggers $\sigma$ whenever $\phi$ holds |
| $proc(\beta(\vec{x}), \sigma)$ | procedure definition |

We will not go over the formal semantics of ConGolog here except to note that it uses a conventional transition semantics defining single steps of computation

and where concurrency is interpreted as an interleaving of primitive actions and test actions. For details see [dGLL00].

### 2.3 Mapping the i* Model to a ConGolog Program

The mapping of the i* elements considered so far results in a purely reactive program. A complex task is transformed into a procedure whereby the body is derived from the subelements. The sequential relations between the subelements are reflected via the use of sequence and `conc`. There are primitive actions preceding and following the body, so that the preconditions to and effects of this element can be reflected in the program. In this simple model all primitive tasks have the same duration (1) and are thus mapped directly to primitive actions. Resources are reflected by fluents. Sequence links, precondition/effect and resource elements are mapped to precondition axioms and effect axioms respectively.

Here is an excerpt of the transformation into ConGolog of the *SeatProducerA*.

```
proc(produce_seats(seatProducerA),
     [pre_produce_seats(seatProducerA),
      conc([buy_material(seatProducerA),
            make_padding(seatProducerA)],
           build_frame(seatProducerA)),
      put_it_together(seatProducerA),
      post_produce_seats(seatProducerA)]).

causes_val(build_frame(seatProducerA), frame, seatProducerA, true).
poss(put_it_together(seatProducerA), and(frame=seatProducerA,
                          executed(make_padding(seatProducerA)))).
```

The main task *produce_seat* is turned into a ConGolog procedure which first checks whether its preconditions are satisfied (*pre_produce_seat*), then calls its subtasks, using concurrency whenever possible, and finally handles effects that are specific to the main task (*post_produce_seat*). (Sub-)Tasks which are not decomposed further are turned into primitive actions such as *build_frame* or *put_it_together*, for which precondition axioms (*poss*) and effect axioms (*causes_val*) need to be specified. For example, the effect axiom for *build_frame(seatProducerA)* simply sets the value of *frame* to *seatProducerA* unconditionally.

When given three agents *CarProducer*, *SeatProducerA*, and another seat producer *SeatProducerB*, the main program of the simulation would simply be the following.

```
proc(main,
     conc(interrupt(true, carProducer_proc),
          conc(interrupt(true, seatProducerA_proc),
               interrupt(true, seatProducerB_proc)))).
```

Here we concurrently start all three agent programs. The interrupt mechanism makes sure that the programs are restarted immediately after termination. The termination of a simulation is achieved by disabling the interrupts.

# 3 Modeling Deliberative Agents in Inter-organizational Networks

The agents we introduced in the previous section are purely reactive. They neither have the opportunity to choose between different ways to pursue their goals nor to decide in which way, or if at all, to execute certain tasks other agents expect them to do. In this section we show how this can be included by taking a particular stance on interpreting goals and softgoals in i*, by adding an explicit account of delegation, and by incorporating a form of decision-theoretic planning into the formalism.

## 3.1 The Use of Goals and Softgoals for Deliberation

Goals in i* can be viewed in two ways. One view is forward-directed where goals are simply postconditions of those tasks that fulfill them. This is how the original SNet interprets them. The other is backward-directed or intentional, that is, starting from the goal, one asks what would be the best way of achieving a goal given a number of possible alternatives. This is the view we will follow in this paper.[4]

To illustrate our approach, let us consider an SR model for the actor *CarProducer* (Figure 2), focusing on tasks, goals (denoted as ovals), and so-called *softgoals*, which are also part of Yu's original i*.

Here the goal *engage_a_seat_supplier* can be achieved in two ways. On one hand, the agent *SeatProducerA* is able to fulfill the goal by executing the action *produce_seats*. On the other hand the agent *SeatProducerB* could execute a task with the same name to achieve the goal, too. Therefore the agent *CarProducer* has the choice between these two alternatives.

Since agents ultimately need to choose among the possible alternatives, we need criteria which the agent can use to distinguish among them. For this purpose we use *softgoals* such as *sportivity* or *comfort*. In i*, the semantics of softgoals is left somewhat vague, but the intuition is that these, like goals, are desirable properties, but that, unlike goals, their satisfaction is a matter of degree. For example, there usually is no clear-cut definition of when a seat is comfortable. Here we keep this intuition, but also provide an unambiguous quantitative interpretation, that is, a softgoal takes a numeric value. Moreover, this value reflects the utility of the property represented by this softgoal.

To see how we arrive at the utility, consider the *contribution links* in Figure 2 that are drawn between the *produce_seats* task of the two seat producers and the softgoals *sportivity* and *comfort*. The idea is that the seat producers supply their respective values for those softgoals. The car producer will then compute, separately for each seat producer, a *utility* value in the range between 0 and 1 using a utility function attached as an attribute to the softgoals (not shown in

[4] Note that i* itself does not commit itself to a particular view.

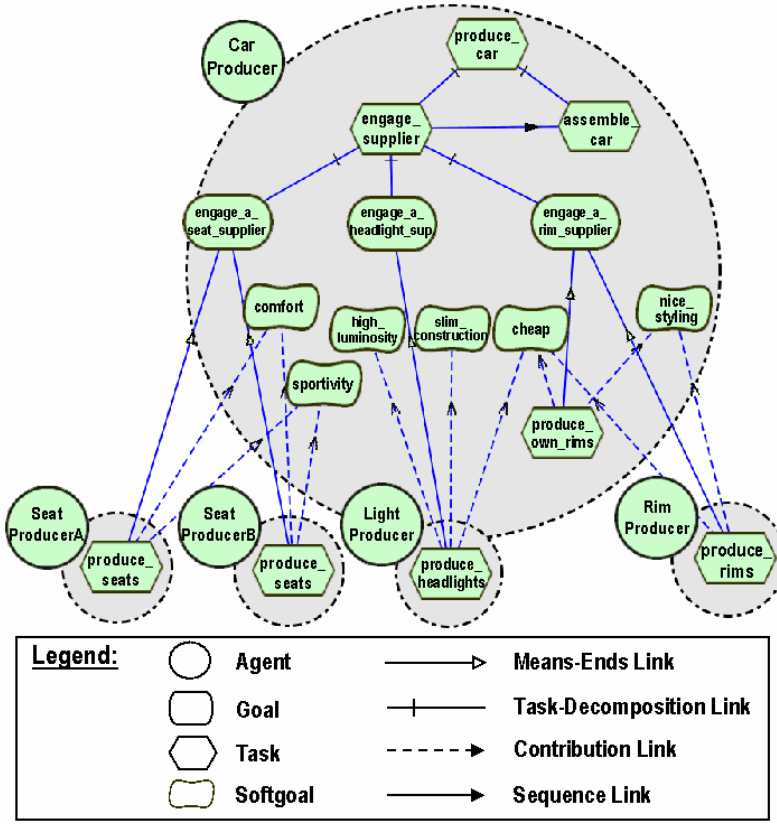


**Fig. 2.** Modeling Deliberation in i*/SNet

the figure). Finally, the utilities are combined using some weighting scheme,[5] which may differ depending on the type of car, say a sports car versus a sedan, and the seat producer with highest overall utility is chosen for the task. For more details on how utilities are computed and compared see the description of the planning component in Section 3.4.

We end this section with a brief remark on the role of *trust* in inter-organizational networks. Clearly, trust has an impact on whether and how agents interact with each other. Based on our previous work [GJK+01], the original SNet already includes basic mechanisms to model and update trust between two agents with respect to a task. In particular, trust is interpreted as a subjective probability and hence can be quantified.[6] In the current version of SNet, this

[5] We remark that our weighted-sum model for decision making is only one possibility among a large number of other multi-criteria decision making methods (see, for example, [Tri00]).

[6] Note that, in contrast to [YL00], we do not model trust in terms of softgoals and contribution links. See [GJK+01] for the reasons why.

treatment of trust is retained. In addition, trust is also used when computing the utility of delegation, the details of which are left out for reasons of space.

### 3.2 Mapping Goals and Softgoals into ConGolog

The transformation of goal elements and their fulfilling tasks is rather similar to the one of complex tasks, but the subelements are combined by nondeterministic choice operators (`ndet`) to reflect the fact that one of these alternatives has to be chosen. The decision-theoretic planner (Section 3.4) will evaluate all the choices implied by the `ndet`-constructs and return the best one in the form of the corresponding deterministic program.

Softgoals are mapped into functional fluents containing the contributions. The contribution links themselves are mapped into action effects on these fluents.

For example, the subgraph determined by the goal *engage_a_seat_supplier* (see Figure 2) will be roughly mapped into the following ConGolog-Code:

```
proc(engage_a_seat_supplier(carProducer),
     [pre_engage_a_seat_supplier(carProducer),
      ndet(delegate(produce_seats(seatProducerA)),
           delegate(produce_seats(seatProducerB))),
      post_engage_a_seat_supplier(carProducer)]).
```

For the softgoal *sportivity* and the contribution of the *produce_seats* task of *seatProducerA* the following code is generated:

```
prim_fluent(sportivity).
causes_val(post_produce_seats(seatProducerA),
           sportivity,
           150,
           true).
```

Here the value 150 is taken from a given scale, say 0 to 200. The planner will take this value and convert it to a utility in the range 0 to 1.

### 3.3 Delegation

The delegation process is modeled using a simple communication protocol.[7] The protocol consists of three steps: In the first step, the agent who wants to delegate a task (the delegator) sends a request to the agent whom he wants to perform the task (delegatee). This request includes the delegator's preferences by mentioning relevant criteria (softgoals) with a suitable weighting and the earliest possible starting time (EPST) when the delegatee can start this job. The delegatee then answers with an offer specifying to what extent the softgoals will be fulfilled and when the job will be finished. Finally the delegator must inform the delegatee

[7] For more sophisticated scenarios, other protocols involving several rounds of negotiations may be more suitable (see [SQ01] for example). We leave this for future work.

whether in fact he is given the job or not, thus sending a confirmation respectively cancellation message.

So if the car producer wants to produce a new sports car it could start a request towards *SeatProducerA* for sportive seats:

```
send_mail(carProducer,    /* From:    */
          seatProducerA,  /* To:      */
          produce_seats,  /* Subject: */
          ask((sportivity, 0.8), (comfort, 0.2), 10))
                   /* (criteria, weights) and EPST */
```

*SeatProducerA* might answer as follows:

```
send_mail(seatProducerA,  /* From:    */
          carProducer,    /* To:      */
          produce_seats,  /* Subject: */
          answer((sportivity, 150), (comfort, 100), 14))
                   /* contributions and end time */
```

Then the car producer could confirm this delegation by sending:

```
send_mail(carProducer,    /* From:    */
          seatProducerA,  /* To:      */
          produce_seats,  /* Subject: */
          confirm)
```

### 3.4 Decision-Theoretic Planning

The need to reason about how to perform a job results either from the agent's own initiative like producing a car[8] or from a newly received delegation request such as the request from the car producer to *SeatProducerA* for the production of car seats.

Since the description of how to do a job may contain several alternatives to choose from, the agent must be given the ability to decide on them. For that purpose we already introduced criteria and a utility model to evaluate the alternatives. Here we propose a planner that completely evaluates all possibilities leading to a globally optimal decision for the given utility functions. Roughly, the planner does the following. For each possible course of actions specified by a nondeterministic program, it generates the deterministic program corresponding to this course of actions. The utility of this program is computed and compared with all the other alternatives. Finally, the program representing the best alternative is returned.

Given that we only need to deal with ConGolog programs that are the result of a mapping from an i* model, the planning component can be restricted to cope with a small set of ConGolog elements: primitive actions, non-recursive

[8] In our simulations the *initiative* to perform a task is itself simulated using so-called exogenous events provided by ConGolog.

procedures, `;` (sequence), `conc`, `ndet`, and `delegate` (as a special mark for tasks to be delegated).

As a simple example, one of the deterministic alternatives of the ConGolog procedure for the goal *engage_a_seat_supplier* (see page 250) is the following:

```
[pre_engage_a_seat_supplier(carProducer),
 delegate(produce_seats(seatProducerA)),
 post_engage_a_seat_supplier(carProducer)]
```

To compute the utilities and the duration of the job the resulting deterministic program is then processed. Attention must only be paid to handle `delegate`. Once the planning process meets such an element, a message is sent to the agent containing a weighting of relevant criteria and an earliest possible starting time for the delegated task resulting from the planning so far. The answer of the agent, i.e. alleged criteria contributions and finishing time, can be modified according to the trust the agent has in his partner. The planning process is continued with this (possibly) modified information taken into account.

After processing an alternative completely the utility can be computed by applying utility functions to the corresponding fluents representing the relevant criteria. The agents involved in the loser of the comparison with the up to now best solution are immediately informed about the cancellation of their jobs (see the delegation protocol).

Once all alternatives are processed the best one is known. Since the non-determinisms are eliminated in advance the processed alternative itself can be used as the policy of how to proceed. The instantiation of the time parameter resulting from the processing is used in the plan monitoring component which checks for delays during execution of this job (see also Section 4.1). Allowing for `conc` elements in the policy is also important because of possibly occurring delays. A pure sequence of primitive actions would be too restrictive, since a delayed execution of a delegated task could then block the agent's own actions.

If planning results from an agent's own initiative, it can commit to the chosen alternative by sending confirmation messages to all agents involved and adding the policy to its schedule. If planning results from a delegation request the computed data can be used to answer this request. In this case the agent has to wait for a final confirmation before adding the job to its schedule. Both cancellation or confirmation are propagated towards the agents involved in the chosen alternative.

Returning to our example (see Figure 2) and assuming that the chosen alternative reflects that for fulfilling the goal *engage_a_seat_supplier SeatProducerA* has proven to be best and for *engage_a_rim_supplier* it occurs that the car producer should do it himself the following policy is created for the task *produce_car* of the car producer:

```
[pre_produce_car(carProducer),
 pre_engage_supplier(carProducer),
 conc([pre_engage_a_seat_supplier(carProducer),
       delegate(produce_seats(seatProducerA)),
       post_engage_a_seat_supplier(carProducer)],
```

```
      conc([pre_engage_a_headlight_supplier(carProducer),
            delegate(produce_headlights(lightProducer)),
            post_engage_a_headlight_supplier(carProducer)],
           [pre_engage_a_rim_supplier(carProducer),
            produce_own_rims(carProducer),
            post_engage_a_rim_supplier(carProducer)])),
 post_engage_supplier(carProducer),
 assemble_car(carProducer),
 post_produce_car(carProducer)]
```

In [BRST00], a related decision-theoretic variant of Golog (DTGolog) was proposed. Our planning component differs in that it is a special purpose planner which does not allow for all Golog actions or stochastic actions. On the other hand, we consider concurrency which DTGolog does not.

## 4 Modifications in SNet: Implementation Aspects

### 4.1 Realization of Autonomous Agents in ConGolog

Since we focus on deliberative autonomous agents as representatives for members of an inter-organizational network, this autonomy of an agent has to be reflected in the implementation. We implemented this multi agent framework within Golog itself:

```
proc(agent_simulator(Agents),
     while(true, [/* deliberative phase */
                  check_delay(Agents),
                  check_mails(Agents),
                  /* acting phase */
                  start_actions(Agents),
                  clock_tick(system),
                  end_actions(Agents),
                  reset_phase(system)])).
```

The parameter to this main program is a list of all agents in the simulation. In every time unit two phases can be identified: First a *deliberative* phase where all agents check their current job for unexpected delays. If an unacceptable delay occurs an agent might decide to cancel the execution to prevent future jobs from being affected.[9] Handling mails leads to the use of the planning component and might add future jobs to the agent's schedule (fluent `schedule(Agent)`).[10] An agent starts a new planning process whenever he detects a new `ask` message in his mailbox. Within this planning process the agent can send requests to other agents, if he depends on a delegation, thus activating their planning process. Planning and so handling mails is finished as soon as every planning process of

[9] We do not provide a general recovery mechanism like re-planning yet. Only the agent who initiates this task has the obligation to re-initiate it.

[10] Since exogenous actions are masked as mails to the agent itself, these actions initiate the planning rounds.

every agent is finished and none of the agents has got a mail in its inbox any more.

During the *acting* phase each agent is allowed to execute its current job which is stored in the fluent `prog_in_execution(Agent)`. This phase is sub-divided into two phases (starting and finishing phase[11]) by `clock_tick`, which also increases the `time` by 1. To realize the execution of an agent's current job, a special primitive action `execute(Agent)` is introduced, after which the history is not only extended by this action but also by an action taken by the given agent.

Executing the primitive action `reset_phase` allows for the starting phase again and finishes the run for the current time unit. The program then loops back to start with the deliberative phase for the next time unit.

In comparison to the old implementation (especially the old main program see page 247) the following observations can be made:

- Providing a *deliberative* phase is completely new. The old agents were reactively waiting for the preconditions of their tasks to become true to be able to execute them.
- The agents now can communicate with each other about delegations.
- The sub-division of the *acting* phase corrects the old implementation.
- In the old version we need not store agent programs in fluents but resulting from this the agent itself is given more control over his jobs e.g. cancellation of the current job, adding new jobs (from delegations).

Communication is realized just as in [LLL+95] via primitive (system) actions `send_mail`, `sense_mail`, and `remove_mail`. The four types of messages can be derived from the delegation protocol: ask, answer, confirm, cancel. We could not adopt the model provided as a successor to this one by [SLL97] because of its different approach towards delegation. In contrast to our approach in their model already a request adds a non-removable goal to the asked agent to fulfill the request.

### 4.2 Architecture of the SNet Tool

The extension towards more deliberative agents has no impact on the software architecture of the original SNet (see Figure 3) as a whole. We use OME3 (Object Modeling Environment) [LY] to build up the static extended i* model (.tel file). The plugins for OME are modified to reflect the new semantics of the revised goal and softgoal elements and to provide the correct transformation to ConGolog code (.pl file).

ConceptBase [JEG+95] as the database in which the model is stored and which is used for static analysis and the translation process is not affected at all.

The implementation of the ConGolog interpreter (IndiGolog) in which the simulation runs is also adapted for example to the process model, the planning

[11] The sub-division is due to a technical reason resulting from the process model for primitive tasks in combination with the simple interleaved concurrency model.

component, and the new agent model, e.g. the handling of the special primitive action `execute(Agent)`.

The simulator uses ConceptBase to retrieve the graphical representation of a model, shows a step by step view of the simulation run and provides access to control the simulation for example via exogenous actions. Thus it must accomodate the changes in the IndiGolog interpreter and the ConGolog code resulting from the transformation.

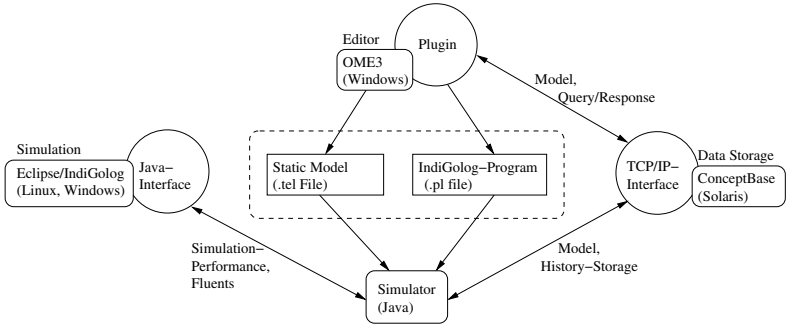


**Fig. 3.** SNet Software Architecture

## 5 Discussion

The usefulness of the approach proposed here can be discussed using an analogy with static and dynamic simulation techniques in engineering systems. For example, in chemical engineering, static equilibrium analysis tells you how a chemical plant will behave in steady state. Unfortunately, history knows several plants where such steady-state simulations showed beautiful results but the desired steady state could never be reached after actually building the plant (or, even worse, could not be shut down once started, without major damage).

Similarly, many failures of business process reorganizations or attempts to set up inter-organizational information and cooperation systems demonstrate that brittle social forms such as inter-organizational networks or inter-organizational information systems may work beautifully in a stable state, but there might be no way to get them started or significantly changed from an actual state. SNet aims at simulating such dynamic settings. Currently, several application domains are under investigation. For example, we have made a comparative static analysis of high-tech entrepreneurship networks in a German and an American region and will try to formally reconstruct these results using our approach, with the aim of evaluating different strategies for improving the entrepreneurship situation in Germany [FM03,JKM03]. As another example, we have developed negotiation support technology for eCommerce, and will apply our approach to forecast the impact of this technology on the regional architecture and construction industry network where this technology is being tried out [Sch02]. As a third example, a collaborative research center in Aachen is studying the optimization of innovation chains in chemical engineering across organizational boundaries. This

requires information sharing across organizational boundaries without losing the key trade secrets of the involved partners [JM02]. Again, it will be interesting to analyze the goal settings, dependencies, and inter-organizational cooperation and information exchange rules under which such a cross-organizational business innovation can be successfully set up and maintained.

From the point of view of SNet itself, many further extensions seem desirable. For example, as was argued in [GJK+01], distrust between agents plays an important role in that it leads to an increased level of monitoring of the activities of distrusted partners. We would like to incorporate an appropriate model of such behavior in SNet. As a simplification for modeling big (realistic) models we plan to make use of i*'s *role concept.* Instead of modeling similar agents separately, agents will be assembled during transformation by assigning roles to them. In this regard agent properties become important since they then provide the only differences between agents capable of fulfilling the same role. Of course, equally important is an in-depth evaluation of our methodology in a real-world scenario.

**Acknowledgment.** This work was supported in part by the Deutsche Forschungsgemeinschaft in its Priority Program on Socionics, and its SFB 476 (IMPROVE).

## References

[BRST00] C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-theoretic, high-level agent programming in the situation calculus. In *AAAI-2000, 17th National Conference on Artificial Intelligence, Austin, Texas*, 2000.

[dGLL00] G. de Giacomo, Y. Lespérance, and H.J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1-2):109–169, 2000.

[FM03] C. Funken and M. Meister. Netzwerke als Single Bars, affinity groups und interorganisationales Regime. Freiburg, 2003.

[FPMT01] A. Fuxman, M. Pistore, J. Mylopoulos, and P. Traverso. Model checking early requirements specifications in Tropos. In *Proceedings Fifth IEEE International Symposium on Requirements Engineering (RE01), Toronto, Canada*, August 27–31 2001.

[GJK+01] G. Gans, M. Jarke, S. Kethers, G. Lakemeyer, L. Ellrich, C. Funken, and M. Meister. Requirements modeling for organization networks: A (dis-) trust-based approach. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering (RE01)*, pages 154–163, Toronto, Canada, August 2001.

[GJLV02] G. Gans, M. Jarke, G. Lakemeyer, and T. Vits. SNet: A modeling and simulation environment for agent networks based on i* and ConGolog. In *Proceedings of the 14th International Conference on Advanced Information Systems Engineering (CAiSE02)*, LNCS 2348, pages 328–343, Toronto, Canada, May 2002.

[JEG+95] M. Jarke, S. Eherer, R. Gallersdörfer, M. A. Jeusfeld, and M. Staudt. ConceptBase - a deductive object base for meta data management. *Journal of Intelligent Information Systems, Special Issue on Advances in Deductive Object-Oriented Databases*, 4(2):167–192, 1995.

[JKM03] M. Jarke, R. Klamma, and J. Marock. *Zu den Wirkungen des regionalen Kontexts auf Unternehmensgründungen*, chapter Gründerausbildung und Gründernetze im Umfeld technischer Hochschulen: ein wirtschaftsinformatischer Versuch, pages 115–154. EUL-Verlag, 2003.

[JM02] M. Jarke and H. C. Mayr. Mediengestütztes Anforderungsmanagement. *Informatik Spektrum*, 25(6):452–464, 2002.

[LLL+95] Y. Lespérance, H. J. Levesque, F. Lin, D. Marcu, R. Reiter, and R. B. Scherl. Foundations of a logical approach to agent programming. In *ATAL-95, Intelligent Agents II. Workshop on Agent Theories, Architectures, and Languages*, 1995.

[LRL+97] H.J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1):59–84, 1997.

[LY] L. Liu and E. Yu. Object Modeling Environment (OME). http://www.cs.toronto.edu/km/ome.

[McC63] John McCarthy. Situations, actions and causal laws. Technical report, Stanford University, 1963. Reprinted 1968 in Minsky, M.(ed.): Semantic Information Processing, MIT Press.

[OSS94] A. Oberweis, G. Scherrer, and W. Stucky. INCOME/STAR: Methodology and tools for the development of distributed information systems. *Information Systems*, 19(8):643–660, 1994.

[PJ96] P. Peters and M. Jarke. Simulating the impact of information flows on networked organizations. In *Proceedings of the 17th International Conference on Information Systems, Cleveland, Ohio, USA*, pages 421–439, December 1996.

[Rei01] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems.* MIT Press, 2001.

[Sch94] A.-W. Scheer. *Business Process Engineering - Reference Models for Industrial Companies.* Springer Verlag, Berlin, 2 edition, 1994.

[Sch02] M. Schoop. Electronic markets for architects - the architecture of electronic markets. *Information Systems Frontiers*, 4(3):285–302, 2002.

[SLL97] S. Shapiro, Y. Lespérance, and H. J. Levesque. Specifying communicative multi-agent systems with congolog. In *Working Notes of the AAAI Fall 1997 Symposium on Communicative Action in Humans and Machines*, pages 72–82, Cambridge, MA, Novemeber 1997. AAAI Press.

[SQ01] M. Schoop and C. Quix. Doc.com: a framework for effective negotiation support in electronic marketplaces. *Computer Networks*, 37(2):153–170, 2001.

[Tri00] E. Triantaphyllou. *Multi-Criteria Decision Making Methods: A Comparative Study.* Kluwer Academic Publishers, 2000.

[WL01] Xiyun Wang and Yves Lespérance. Agent-oriented requirements engineering using ConGolog and i*. In *Working Notes of the Agent-Oriented Information Systems (AOIS-2001) Workshop, Montreal, QC*, May 2001.

[YL00] E. Yu and L. Liu. Modelling trust in the i* strategic actors framework. In *Proceedings of the 3rd Workshop on Deception, Fraud and Trust in Agent Societies, Barcelona, Catalonia, Spain (at Agents2000)*, June 3–4 2000.

[Yu95] E. Yu. *Modelling Strategic Relationships for Process Reengineering.* PhD thesis, University of Toronto, 1995.