Formal Specification of Multi-Agent Systems:
a Real-World Case

Frances Brazier[a], Barbara Dunin Keplicz[b], Nick R. Jennings[c] and Jan Treur[a]

[a] Vrije Universiteit Amsterdam, Department of Mathematics and Computer Science, Artificial Intelligence Group,
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands. Emails: {frances,treur}@cs.vu.nl

[b] University of Warsaw, Institute of Informatics, ul. Banacha 2, 02-097 Warsaw, Poland. Email: keplicz@mimuw.edu.pl

[c] University of London, Queen Mary & Westfield College, Department of Electronic Engineering,
Mile End Road, London E1 4NS, United Kingdom. Email: N.R.Jennings@qmw.ac.uk

Abstract
In this paper the framework DESIRE, originally designed for formal specification of complex reasoning systems is used to specify a real-world multi-agent application on a conceptual level. Some extensions to DESIRE are introduced to obtain a useful formal specification framework for multi-agent systems.

1 Introduction

In many areas of software engineering and knowledge-based system design, formal specifications of the conceptual design of complex systems are devised before systems are implemented; for an overview in the area of complex (knowledge-based) reasoning systems, see (Treur & Wetter 93). Such specifications describe the semantics of systems without concern for implementation details, providing a basis for verification and validation of the functionality of the systems. Most specification frameworks, however, do not provide adequate means to describe the dynamics of reasoning behaviour and acting behaviour (e.g., guided reasoning, observation, communication and execution of actions) of complex systems: a crucial characteristic of multi-agent systems.

One formal specification framework, DESIRE (framework for DEsign and Specification of Interacting REasoning components; cf. (Langevelde, Philipsen & Treur 92; Brazier, Treur, Wijngaards & Willems 94)), originally designed for complex reasoning systems, does focus on the specification of the dynamics of reasoning and acting behaviour. Within this framework complex reasoning systems are designed as interacting task-based hierarchically structured components, as compositional architectures. The interaction between components, between components and the external world, as well as between components and one or more users (cf. (Brazier & Treur 94)), is formally specified. Components can be reasoning components (for example based on a knowledge-base), but may also be subsystems which are capable of performing tasks such as calculation, information retrieval, optimisation, et cetera. Formal semantics of such compositional reasoning systems are defined on the basis of temporal logic (cf. (Engelfriet & Treur 94; Gavrila & Treur 94; Treur 94)). As implementation generators exist to automatically generate prototype implementations from formal specifications, a system designer can focus on the specification of the conceptual design of a system: on both the static and the dynamic aspects of the required functionality. The formal specification framework DESIRE is currently used by a number of companies and research institutes for the development of knowledge-based systems for complex reasoning tasks. A number of knowledge-based systems developed using DESIRE have become operational.

The aim of the current paper is to introduce a specification framework based on DESIRE tuned to multi-agent tasks. Given the basic assumptions behind DESIRE, the extension of the formal specification framework to multi-agent systems is relatively straightforward and natural. Agents are most often autonomous entities, designed autonomously. Agents can be activated in parallel and control resides (essentially) in the agents themselves.

This paper briefly describes the formal specification framework DESIRE and its application to multi-agent systems in general, illustrated on the basis of one of the few operational real-world distributed artificial intelligence applications, a system for electricity transportation management (Cockburn & Jennings 95). The paper has the following structure. In Section 2 the application domain is described; in Section 3 the specification framework is introduced. In Section 4 the specification of the example multi-agent system is presented. Finally, in Section 5 a discussion of some conclusions and further perspectives is presented.

2 The Application Domain

The multi-agent system described in this paper was developed in the ARCHON project (see (Cockburn & Jennings 95)) and is currently running on-line in a control room in the North of Spain (see (Jennings et al. 95)). An electricity transportation network carries electricity from generation sites to the local networks where it is distributed to customers. Managing this network is a complex activity which involves a number of different subtasks: monitoring the network, diagnosing faults, and planning and carrying out maintenance when such faults occur. The running application involves seven agents. In this paper we will focus on the three most representative agents.

The *Control System Interface agent* (CSI) continuously receives data from the network - e.g., alarm messages about unusual events and status information about the network's components. From this information, the CSI periodically produces a snapshot which describes the entire system state at the current instant in time. It also performs a preliminary analysis on the data it receives from the network to determine whether there may be a fault.

Two diagnosis agents are also considered - an *Alarm Analysis Agent* (AAA) and a *Blackout Area Identifier agent* (BAI). Both of these agents are activated by the receipt of information from CSI which indicates that there might be a fault. They both use CSI's snapshot information to update their model of the network on which their diagnosis is based. BAI is a fast and relatively unsophisticated diagnostic system which can pinpoint the approximate region of the fault (the initial *blackout area*) but not the specific element which is at fault. AAA, on the other hand, is a sophisticated model-based diagnosis system which is able to generate and verify the cause of the fault in the network. It does this in a number of different phases. Firstly, it performs an approximate *hypothesis generation* task which produces a large number of potential hypotheses (the knowledge used here guarantees that the actual fault is always contained in this initial list). It then takes each of these hypotheses in turn and performs a time consuming *validation* task to determine the likelihood that the given hypothesis is the cause of the problem. Cooperation occurs between AAA and BAI in that BAI's initial blackout area can be used to prune the search space of AAA's hypothesis validation task. It can do this because the fault will be contained in the initial blackout area - hence any hypotheses produced by AAA's generation task which are not in the blackout area can be removed from the list which needs to be considered by AAA's validation task. The blackout area can be received by AAA in two different ways. The most usual route is that BAI will volunteer it as unsolicited information - BAI maintains a model of all the agents in the system (its *acquaintance models*) and its model of AAA will specify that it is interested in receiving information about the blackout area. Hence when this information is produced it will automatically send it after making reference to its acquaintance models. The other route is that AAA will generate an information request to have the initial blackout area produced - this will, in fact, result in a request being directed to BAI because AAA's acquaintance model of BAI indicates that it has a task which produces the initial blackout area as a result.

3 A Formal Specification Framework for Multi-Agent Sytems

The formal specification of compositional architectures for multi-agent systems is based on a task-based approach to multi-agent systems' design. As a result of task analysis, hierarchical task models are specified at different levels of abstraction as is the interaction between tasks. A close relation exists between the different levels of task decomposition and the specification of the interaction between tasks: the interaction is specified for each level within the task decomposition. Each task is assigned to one or more agents. Agents themselves perform one or more (sub)tasks, either sequentially or in parallel. The knowledge that agents have of themselves and of other agents and the world is explicitly specified.

3.1 Formal specification framework

The task hierarchy devised during task analysis is the basis for the structure of a compositional architecture: *components* distinguished within a formal specification are defined according to the task hierarchy. Interaction between components, the basis for modelling complex behaviour, is formally specified by *information links* between components. These two aspects of the framework, components and information links, are addressed below, followed by an overview of the types of knowledge included in a formal specification (see also (Brazier, Treur, Wijngaards & Willems 94)).

3.1.1 Specification of components

During task analysis a hierarchy of tasks is composed within which *complex* and *primitive tasks* are distinguished. Complex tasks are described by one or more subtasks, which, in turn, may be described by even more specific subtasks, etc. The most specific tasks are the primitive tasks: those which are not further decomposed.

Within a compositional framework, in which components are directly related to tasks, components can be either (1) composed, or (2) primitive. *Composed components* specify the knowledge required to perform the related complex task in the task hierarchy. For each composed component the following types of knowledge are specified: the input and output interface, the task control structure, the specific subtasks of the complex task and their information links, and the relevant domain knowledge structures. *Primitive components* are similarly related to primitive tasks: a primitive component specifies the knowledge required to perform a primitive task. Primitive tasks can be performed by primitive (knowledge-based) reasoning components, but

also by, for instance, neural networks, OR-algorithms and conventional components.

The language in which a component is specified is based on an *order-sorted predicate logic* (i.e., predicate logic with a hierarchically ordered sort structure) within which *signatures* are defined. These signatures distinguish sorts, predicate symbols (relations), function symbols and constants (objects). The input and output interface of a component are defined in the appropriate *interface signature*. For each component two interface signatures are defined:

- the *input signature* describing the facts (possibly at different (meta-)levels) given as input;
- the *output signature* describing the resulting facts (possibly at different (meta-)levels) derived.

In addition a component may have an *internal signature*. The *information state* of a component is dynamic: it provides a repository for all domain information related to a component generated during task execution and received as a result of interaction with other components.

3.1.2 Interaction between components

Information exchange between components is based on *information links*. An information link specifies which truth value of an atom in the order-sorted logic used in one component is to be identified with which truth value of which atom in another component. Transfer of information may also entail *renaming* of terms. The mapping then in principle defines a kind of translation table between the two signatures involved (see also (Brazier, Treur, Wijngaards & Willems 94)). The language employed to specify an individual component is therefore independent of other components.

Within compositional architectures both complex reasoning behaviour (changing ones own information states internally) and acting behaviour (changing world or agent states externally) is modelled as combinations of nontrivial (and dynamic) patterns of interaction between components. These dynamic patterns of interactions are modelled as interactions between different *levels of reasoning* (or other computational processes) distinguished within the architecture as a whole but also within components: object level, meta-level, meta-meta-level, et cetera. At the lowest level, *object-level reasoning* entails reasoning about the state of the world; reasoning about object-level reasoning is *meta-level reasoning*. This object-meta distinction can be repeated: the meta-level is in fact the object level for the meta-meta-level, et cetera. The result of a component's reasoning at the meta-level is used to guide or influence the related object-level reasoning. This includes, for example, specification of the *assumptions* which the object-level reasoning should use, or of the *target* facts which the object-level reasoning should (try to) derive. The actual transfer of the output of meta-level reasoning to influence object-level reasoning is known as a *downward reflection*.

Meta-level reasoning uses as input information about the truth, falsity or undefinedness of object-level facts (*epistemic information*), but also notification of the fact that a component reasoning at the object-level requires additional information from other components to be able to derive specific results (*requests*). The transfer of this information from a component reasoning at an object-level to provide input for a component reasoning at a meta-level is known as an *upward reflection*.

3.1.3 The elements in the formal specification of a compositional system

A specification document for a (hierarchical) compositional architecture contains specifications of the components and the relations between components. Five types of knowledge are modelled during task acquisition:

(1) knowledge of the task structure
(2) knowledge of sequencing of (sub)tasks
(3) knowledge of information exchange between (sub)tasks
(4) knowledge of knowledge structures and knowledge decomposition
(5) knowledge of role delegation

These types of knowledge are explicitly modelled within the DESIRE framework. A specification document for a (hierarchical) compositional architecture contains specifications of the five types of knowledge specified in a formal document as:

(1) a *task decomposition*: a task hierarchy together with specification of input and output signatures for each of the (sub)tasks;
(2) agent *task control knowledge*, specifying activation of (sub)tasks within each individual agent, but also specifying initial activation of agents;
(3) *information links* between components to enable information flow;
(4) *task-knowledge allocation* with (references to) appropriate (domain) knowledge structures;
(5) *task allocation* (between agents).

3.2 Formal specification of compositional agents

Compositional architectures are clearly based on the notion of component described above. A *compositional agent* is a composed component with a number of subcomponents representing the agent's tasks to be performed and additional knowledge of the world and other agents and how to interact with them. The types of knowledge distinguished above can also be distinguished with respect to compositional agents.

3.2.1 Task hierarchy and task allocation

For design from scratch, including the design of agents themselves, tasks distinguished within a task hierarchy can be assigned to different agents on the basis of the task decomposition: both complex and primitive tasks alike. Task allocation at this level is not one-to-one: in many situations the same task may be assigned to more than one agent. In other situations, agents already have certain capabilities and characteristics. Task allocation involves distinguishing which tasks within a task hierarchy can be performed by

which agent. The level of collaboration and cooperation involved between particular agents will strongly influence task allocation.

3.2.2 Information flow within and between agents and between agents and the world

Information exchange between components *within an agent* is information exchange within a composed component: information links between components define which information can be transferred from one component to another (see Section 3.1.2). Modelling agents as composed components necessarily implies that the exchange of information *between agents* is specified in information links: specifying which information is to be exchanged.

Information links support modelling of specific types of interaction. For example, in a given situation an agent may require specific information to be able to complete a reasoning task. The agent transfers this request as meta-information to one or more other agents through information links. The information requested may, as a result, be transferred back to the agent through other information links. This mechanism is an essential element in modelling *communication between agents*.

Interaction between an agent and the external world is modelled almost identically from the agent's point of view. For example: an *observation of the external world* may be modelled as an agent's specific request for information about the external world, transferred as meta-information to the external world through an information link. As a result of the request information may be transferred through another link back to the requesting agent. The external world includes information on the current state of the world.

Another form of communication between an agent and the external world is the *performance of a specific action*. An agent performs an action by transferring information to this purpose to the external world, upon which the external world state changes.

3.2.3 Task control within an agent

Task control knowledge specified in complex and primitive tasks alike, makes it possible to specify an agent's reasoning and acting patterns distributed over the hierarchy of agent components. Within our compositional framework such knowledge is expressed in temporal rules (see (Engelfriet & Treur 94; Gavrila & Treur 94)). Each component is assumed to have a (local, linear) discrete time scale. When and how a component will be activated (and whether activation is continuous or not) is specified. This most often includes the specification of at least:
- the *interactions* required to provide the necessary input facts,
- the set of facts for which truth values are sought (*target set*)

Evaluation of the status of other components is often required to determine when a specific component is to be activated. A component is considered to have been successful with respect to one of its target sets if it has reached its goal, specified by this target set (given default specifications of the number of targets to be reached (e.g., any, or every) and the effort to be afforded). If not, it is considered to have failed. A typical example of a component's task control knowledge rule in which the success of one component is required (own_process_control) before a following component (update_snapshot) can be activated with the required information, is the following:

if   evaluation(own_process_control, ts_update, succeeded)
then next-component-state(update_snapshot, active)
   and   next-target-set(own_process_control, ts_poss_hyps)
   and   next-link-state(incoming_info_for_snapshot_update, up_to_date)

This knowledge rule states that

if   the component update_snapshot has succeeded in accomplishing every target which it was assigned,

then the component own_process_control is assigned a new set of targets to accomplish, and the next component to be activated is specified, namely the component update_snapshot, given information which has been recently updated by activation of the link incoming_info_for_ snapshot_update.

Note that next-link-state(incoming_info_for_snapshot_update, up_to_date) indicates that the link incoming_info_for_snapshot_update has been activated in order to transfer the information required for the next component, update_snapshot. This is not a guarantee that the information itself is new: it is only a guarantee that the link has been activated.

The activation of components does not always depend on the completion of another component. In some cases the input causes a component to become active. The specification of the fact that a component is to be continually capable of performing its subtask during task execution (in parallel with other components), depending on the availability of new input, is expressed by:

if   start
then next-component-state(D, awake)

3.2.4 Task control between agents and between agents and the external world

Minimal global task control is required to initially activate all agents (and possibly links) involved in task performance. Once agents are active their agent task control knowledge determines the sequencing of task execution.

4 Formal specification of the example multi-agent system

4.1 Task decomposition and role allocation

The example system described in Section 2 consists of three agents (CSI, AAA, BAI) and interactions between the agents

and between agents and the world. Within our formal framework DESIRE agents are specified as specific types of composed components (see Section 3.1.1). The main tasks of each of the agents in this example are similar; they each have the same three generic tasks (own process control, snapshot update, preparing communication) and one agent-specific task (e.g., diagnose fault (for the agent AAA), or identify blackout area (for the agent BAI)) to perform; see Figure 1 for agent AAA's first level decomposition. These generic tasks are generic in the sense that they can be (specialised and) instantiated for different agents (reuse).
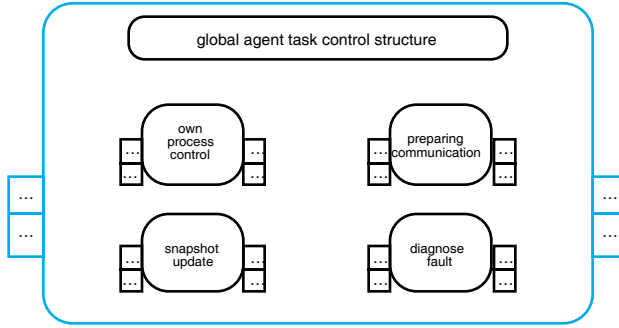

Figure 1 Top-level compositional structure of agent AAA

In Figure 1 the small boxes on the left and right hand side denote the levelled input and output interface respectively. Here the levels indicate object-meta distinctions. The agent AAA's complete task hierarchy as described in Section 2 is given by Figure 2:

1. Own process control
    1.1 Monitoring incoming data
    1.2 Evaluating the process state
2. Update snaphot
3. Diagnose fault   (agent specific task)
    3.1 Hypothesis generation
    3.2 Hypothesis refinement
    3.3 Hypothesis validation
        3.3.1 Evaluating hypothesis
        3.3.2 Deriving causal consequences
4. Managing communication
    4.1 Examining the acquaintance model
    4.2 Generating requests

Figure 2 Complete task hierarchy of agent AAA

To illustrate the way in which task hierarchies (see Section 3.2.1) are specified, in Figure 3 the task diagnose fault is decomposed one step further (compared to Figure 1); here the information links are depicted as well. In the formal specification the task-subtask relations and link names (as shown in Figure 2) are expressed as follows:

```
task structure  diagnose_fault
 subcomponents  hypothesis_generation, hypothesis_refinement,
                hypothesis_validation ;
 links     import_disturbances, import_snapshot_info,
           import_blackout_info, poss_hyps_to_refine,
           poss_hyps_to_validate, lim_hyps_to_validate,
           export_diagnosis ;
end task structure diagnose_fault
```

The link names specified above refer to information links between subcomponents of the component diagnose fault; detailed specifications of these links are of the form given in Section 4.2.

4.2 Information flow within an agent

An example of an information link specification (see Sections 3.1.2 and 3.2.2) within agent AAA is the link between the component hypothesis_generation and the component hypothesis_validation:

```
private link     poss-hyps-transfer: object-object
   domain        hypothesis generation
      output     poss-hyps
   codomain      hypothesis validation
      input      hyps
   sort links    (Hyps,Hyps)
   object links  identity
   term links    identity
   atom links    (poss-hyp(H:Hyps), hyp(H:Hyps)):
                 <<true,true>,<false,false>>

endlink
```

This link relates output of the component hypothesis_generation to input of the component hypothesis_validation, where the truth value true (resp. false) of an atom of the form poss-hyp(H:Hyps) is translated into the truth value true (resp. false) of an atom of the form hyp(H:Hyps).

The components hypothesis_generation and hypothesis_refinement represent meta-level reasoning components (with respect to the object level reasoning about the world). These meta-level components use epistemic information from outside the component diagnose_fault as their input; their input arrows start one (meta-)level higher in the input interface (see Figure 3). In the specification these information links have names that can be used in the task control knowledge, to specify under which conditions they have to transfer the up-to-date information.

4.3 Task control within an agent

Before describing task control knowledge (see Section 3.2.3) for the component diagnose_fault, AAA's own task control knowledge will be addressed, expressing control over its four main tasks.

4.3.1 Agent AAA's task control

From the start both the agent component own_process_control and the information link incoming_snapshot_for_own_process_control have to become and remain awake. This is expressed by:
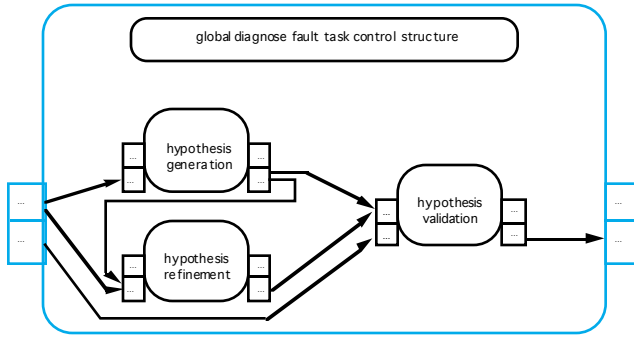
Figure 3 The component diagnose fault of agent AAA

```
if     start
then   next-component-state(own_process_control, awake)
   and   next-target-set(own_process_control, all_targets)
   and   next-link-state(incoming_snapshot_for_own_process_control, awake)
```

The component update_snapshot is only active under certain conditions (depending on whether or not incoming snapshot data have been monitored by the component own_process_control). In this case the snapshot information has to be transferred to this component as well:

```
if     evaluation(own_process_control, ts_update, succeeded)
then   next-component-state(update_snapshot, active)
   and   next-target-set(own_process_control, ts_poss_hyps)
   and   next-link-state(incoming_info_for_snapshot_update, up_to_date)
```

The component diagnose_fault is activated if the component own_process_control determines that a fault should be diagnosed (because alarms are monitored); input information is provided on grouped alarms, the current snapshot and (if available) on the blackout area:

```
if     state(diagnose_fault, idle)
   and   evaluation(own_process_control, diagnose_target_set, succeeded)
then   next-component-state(diagnose_fault, active)
   and   next-target-set(diagnose_fault, faults)
   and   next-link-state(grouped_alarms, up_to_date)
   and   next-link-state(current_snapshot, up_to_date)
   and   next-link-state(blackout_area, up_to_date)
```

The component managing_communication is activated if the component own_process_control determines that a request for black area information is needed (because it is noticed that this information is still lacking):

```
if     evaluation(own_process_control, ts_requests, succeeded)
then   next-component-state(prepare_communication, active)
   and   next-target-set(prepare_communication, communications)
   and   next-link-state(request_info, up_to_date)
```

The actual communication is performed if the component manage_communication succeeds in generating outgoing requests; note that no component states are changed, but only two links are activated in sequence: one to AAA's output interface, and, subsequently, another one from the output interface of AAA to the input interface of the agent BAI (note that the order of activation of links is expressed by the list notation):

```
if     evaluation(prepare_communication, outgoing_requests, succeeded)
then   next-link-state([request_to_output, request_out], up_to_date)
```

If fault results were found, these are transferred to AAA's output interface:

```
if     evaluation(diagnose_fault, fault_results, succeeded)
then   next-link-state(fault_results_to_output, up_to_date)
```

If blackout information has arrived, then the diagnose fault task should be activated, with extra information that blackout information is available:

```
if     evaluation(own process control, ts_blackout_area, succeeded)
then   next-link-state(blackout_area, up_to_date)
   and   next-component-state(diagnose_fault, active)
   and   next-target-set(diagnose_fault, faults)
   and   extra_info(diagnose_fault, blackout_info_available)
```

4.3.2 Task control knowledge for diagnose fault

The control of the three subtasks of the task diagnose fault (hypothesis generation, hypothesis refinement and hypothesis validation) begins by the activation of the component diagnose_fault:

```
if     component-state(diagnose_fault, start)
then   next-component-state(hypothesis_generation, active)
   and   next-target-set(hypothesis_generation, ts_poss_hyps)
   and   next-link-state(import_disturbances, up_to_date)
```

This rule states that once the component diagnose_fault has been activated, the component hypothesis_generation is to be activated with target set ts_poss_hyps and up to date information about disturbances.

If blackout information is available and hypotheses have been generated succesfully, hypothesis_refinement has to be activated, and information on the blackout area and the generated hypotheses has to be provided.

```
if     evaluation(hypothesis_generation, ts_poss_hyps, succeeded)
   and   extra_control_info(diagnose_fault, blackout_info_available)
then   next-component-state(hypothesis_refinement, active)
   and   next-target-set(hypothesis_refinement, ts_ref_hyps)
   and   next-link-state(blackout_info, up_to_date)
   and   next-link-state(poss_hyps_to_refine, up_to_date)
```

If, however, no blackout information is available, the component hypothesis_validation has to be activated, using updated snapshot information and the generated hypotheses:

```
if     evaluation(hypothesis_generation, ts_poss_hyps, succeeded)
   and   not extra_info(diagnose_fault, blackout_info_available)
then   next-component-state(hypothesis_validation, active)
   and   next-target-set(hypothesis_validation, ts_faults)
   and   next-link-state(import_snapshot_info, up_to_date)
   and   next-link-state(poss_hyps_to_validate, up_to_date)
```

The next rule expresses that if blackout information becomes available while the component hypothesis_validation is active, it has to be interrupted and cleared (in order to be able to first refine the generated hypotheses).

```
if      evaluation(hypothesis_generation, ts_poss_hyps, succeeded)
   and   extra_info(diagnose_fault, blackout_info_available)
   and   component-state(hypothesis_validation, active)
then  next-component-state(hypothesis_validation, idle)
   and   next-info-state(hypothesis_validation, clear)
```

After hypothesis_refinement has succeeded, hypothesis_validation has to be activated (again), using input information on the snapshot and the limited set of hypotheses obtained by the refinement:

```
if      evaluation(hypothesis_refinement, ts_lim_hyps, succeeded)
then  next-component-state(hypothesis_validation, active)
   and   next-target-set(hypothesis_validation, ts_faults)
   and   next-link-state(import_snapshot_info, up_to_date)
   and   next-link-state(ilim_hyps_to_validate, up_to_date)
```

4.4 Control and communication between agents

Control at the highest (central) level, between agents, is minimal. Only very simple start rules are specified to awaken the agents. For example:

```
if      start
then  next-component-state(AAA, awake)
```

An example of agent communication between AAA and BAI is the request AAA issues to BAI for blackout area information. An information link request_out from AAA to BAI exists for this purpose: to transfer meta-information stating that blackout area information is needed:

```
private link  request_out: object-object
   domain       AAA
      output    request_output
   codomain    BAI
      input     request_input
   atom links      (boa_info_needed, boa_info_needed): <<true,true>>
endlink
```

The control of this information link is specified in the task control knowledge of the sending agent (see the fifth rule in Section 4.3 for the control of request_out). From BAI to AAA there is an information link blackout_area_transfer to provide AAA with blackout area information; this link is controlled by BAI's task control knowledge.

5 Discussion

The advantages of formal specifications of complex systems have been recognized both within information system design and knowledge-based system design. Such specifications provide an implementation-independent description of the functionality of a system, providing a means to increase maintainability and support reuse of system components (Langevelde, Philipsen & Treur 92; Brazier, Treur, Wijngaards & Willems 94), but also to verify and validate system behaviour (cf. (Treur & Willems 95)).

The application of a formal specification framework, originally developed for complex reasoning systems, to multi-agent systems has been explored in this paper. The compositional nature of architectures designed within the formal specification framework DESIRE, clearly supports the structure required for modelling multi-agents. Agents are composed components; interaction between agents is modelled as interaction between composed components. These principles have, in the past, been succesfully employed for the design and specification of, for example, decision support systems, in which the interaction between a user, a decision support system and an external world (often represented by knowledge stored in a database), have been modelled, specified and used to design and implement prototype systems. Both static and dynamic aspects of the interaction and required system behaviour were specified, defining the semantics of task performance.

Extension of the framework to model multi-agent systems in which agents are not necessarily cooperative and willing to interact, in which agents are not necessarily aware of each others goals, knowledge, et cetera, requires reconsideration of the expressiveness provided within the framework. Aspects such as: knowledge of other components (their capabilities, willingness to communicate, etc), coordination of parallel processes, handling interrupts, different timing schemes, different states of awareness, et cetera (see also (Dieng, Corby & Labidi 94)), need to be formally specified. A number of these aspects were easily incorporated in the framework for the example domain in the paper: the extension of the framework required was minimal.

The incorporation of control information within agents and not at the global level as in the original version of DESIRE, described in (Langevelde, Philipsen & Treur 92), was, for example, relatively straightforward. A first step in this direction had already been taken by adding hierarchical component structures to DESIRE, including hierarchically decentralized control (see (Brazier, Treur, Wijngaards & Willems 94)). For the multi-agent case the framework has been extended to allow parallel activation of composed components and to allow agents to directly control their communication with other agents. An additional component state status distinction was also required: the distinction between active and idle no longer sufficed. In addition to being active or idle, agents and also information links can be awake (capable of processing incoming information as it arrives) or asleep. The same holds for information links: links can likewise be continually awake, although not continually active.

The interaction between the autonomous agents in the case study was well-defined: each agent clearly had its own tasks and the types of information exchange required were known. Duplication of tasks across agents was easily implemented requiring no further extensions of the framework. More extensive forms of collaboration, such as explored in (Brazier & Treur, 94) in which the user and an intelligent system have a shared task model, modelled in DESIRE, required further consideration for multi-agent situations. Collective agent satisfaction as an extension of the concept of collective user satisfaction as modelled in (Brazier & Ruttkay 93) is another concept of interest for further research. Current research focusses on specification

of different types of cooperation (see, for example, (Brazier, Eck & Treur 95)).

In conclusion, the employment of the formal specification framework to multi-agent systems has proven to be promising. The electricity transportation management example has been formally specified within the formal framework DESIRE with minor extensions for the parallel and event-driven processing required. Further research will address the expressiveness required for more extensive forms of parallel and event-driven interaction and the implications of the adaptations of the DESIRE syntax and semantics for the software environment supporting automated generation of executable code out of specifications.

Acknowledgements

References

Brazier, F.M.T., P.A.T. van Eck, J. Treur (1995), Modelling Exclusive Access to Limited Resources within a Multi-Agent Environment: Formal Specification. Technical Report, Vrije Universiteit Amsterdam, Department of Mathematics and Computer Science

Brazier, F.M.T., Ruttkay Zs. (1993), Modelling collective user satisfaction, *Proc. of HCI International'93*, Elsevier, Amsterdam, 1993, pp. 672-677.

Brazier, F.M.T., J. Treur, N.J.E. Wijngaards and M. Willems (1994). Temporal semantics and specification of complex tasks. Technical Report IR-375, Vrije Universiteit Amsterdam, Department of Mathematics and Computer Science. Shorter version in: *Proc. Dutch AI Conference, NAIC'95*, 1995. Preliminary version in: D. Fensel (ed.), *Proceedings of the ECAI '94 Workshop on Formal Specification Methods for Knowledge-Based Systems*, 1994, pp. 97-112.

Brazier, F.M.T. and J. Treur (1994). User centered knowledge-based system design: a formal modelling approach. In: L. Steels, G. Schreiber and W. Van de Velde (eds.), "A future for knowledge acquisition," *Proceedings of the 8th European Knowledge Acquisition Workshop*, EKAW '94. Springer-Verlag, Lecture Notes in Artificial Intelligence 867, pp. 283-300.

Cockburn, D. and N. R. Jennings (1995) "ARCHON: A Distributed Artificial Intelligence System for Industrial Applications". In: *Foundations of Distributed Artificial Intelligence* (eds. G. M. P. O'Hare and N. R. Jennings), Wiley & Sons.

Dieng, R., O. Corby, S. Labidi (1994), Agent-based knowledge acquisition. In: L. Steels, G. Schreiber and W. Van de Velde (eds.), "A future for knowledge acquisition," *Proceedings of the 8th European Knowledge Acquisition Workshop*, EKAW '94. Springer-Verlag, Lecture Notes in Artificial Intelligence 867, pp. 63-82

Dunin Keplicz, B. and J. Treur (1995). Compositional formal specification of multi-agent systems. In: M. Wooldridge, N. Jennings (eds.), *Intelligent Agents*, Proc. of the ECAI'94 Workshop on Agent Theories, Architectures and Languages, Lecture Notes in AI, vol. 890, Springer Verlag, 1995, pp. 102-117

Engelfriet, J. and J. Treur (1994). Temporal Theories of Reasoning. In: C. MacNish, D. Pearce, L.M. Pereira (eds.), *Logics in Artificial Intelligence*, Proc. of the 4th European Workshop on Logics in Artificial Intelligence, JELIA '94. Springer Verlag, pp. 279-299. Also in: *Journal of Applied Non-Classsical Logics*, Special Issue with selected papers from JELIA'94, 1995, to appear

Gavrila, I.S. and J. Treur (1994). A formal model for the dynamics of compositional reasoning systems. In: A.G. Cohn (ed.), *Proc. 11th European Conference on Artificial Intelligence, ECAI'94*, Wiley and Sons, pp. 307-311

Jennings, N. R. , J. Corera, I. Laresgoiti, E. H. Mamdani, F. Perriolat, P. Skarek and L. Z. Varga (1995) "Using ARCHON to develop real-word DAI applications for electricity transportation management and particle accelerator control" *IEEE Expert - Special Issue on Real World Applications of DAI*

Langevelde, I.A. van, A.W. Philipsenand J. Treur (1992). Formal specification of compositional architectures, in B. Neumann (ed.), *Proceedings of the 10th European Conference on Artificial Intelligence, ECAI'92*, John Wiley & Sons, Chichester, pp. 272-276.

Treur, J. (1994). Temporal Semantics of Meta-Level Architectures for Dynamic Control of Reasoning. In: F. Turini (ed.), *Proceedings of the Fourth International Workshop on Meta-Programming in Logic, META '94*. Springer Verlag, Lecture Notes in Computer Science.

Treur, J. and M. Willems (1995). Formal Notions for Verification of Dynamics of Knowledge-Based Systems. In: M.C. Rousset and M. Ayel (eds.), *Proc. European Symposium on Validation and Verification of KBSs, EUROVAV'95*, Chambery

Treur, J. and Th. Wetter (eds.) (1993). *Formal Specification of Complex Reasoning Systems*, Ellis Horwood.