

DESIRE: MODELLING MULTI-AGENT SYSTEMS IN A COMPOSITIONAL FORMAL FRAMEWORK*

FRANCES M.T. BRAZIER

*Department of Mathematics and Computer Science, Vrije Universiteit Amsterdam
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands. Email: frances@cs.vu.nl*

BARBARA M. DUNIN-KEPLICZ

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

NICK R. JENNINGS

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

JAN TREUR

*Department of Mathematics and Computer Science, Vrije Universiteit Amsterdam
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands. Email: treur@cs.vu.nl*

Abstract

This paper discusses an example of the application of a high-level modelling framework which supports both the specification and implementation of a system's conceptual design. This framework, DESIRE (framework for DEsign and Specification of Interacting REasoning components), explicitly models the knowledge, interaction, and coordination of complex tasks and reasoning capabilities in agent systems. For the application domain addressed in this paper, an operational multi-agent system which manages an electricity transportation network for a Spanish electricity utility, a comprehensible specification is presented.

Keywords: Multi-agent system; Modelling framework; Compositional modelling

1. Introduction

As multi-agent technology begins to emerge as a viable solution for large-scale industrial and commercial applications, there is an increasing need to ensure that the systems being developed are robust, reliable and fit for purpose. To this end, it is important that the basic principles and lessons of software engineering are applied to the development and deployment of multi-agent systems. At present, the majority of the extant agent applications are developed in an ad hoc fashion - following little or no

* To appear in the International Journal of Cooperative Information Systems, vol. 6, no 1, Special Issue on Formal Methods in Cooperative Information Systems: Multiagent Systems, M. Huhns, M. Singh (eds.), 1997

rigorous design methodology and with limited a priori specification of the agents or of the system as a whole. This lack of rigour is one of the major factors hampering the wide-scale adoption of agent technology.

This situation is, however, beginning to change as a number of researchers recognise the importance of verifying and validating the properties and characteristics of agent systems. Within this field, two broad camps can be identified: (i) those who use advanced logics to specify their systems^{1,2,3}; and (ii) those who use tools adapted from software and knowledge engineering to specify their systems.^{4,5,6,7} This paper describes work in the latter camp - a modelling framework, called DESIRE (framework for DEsign and Specification of Interacting REasoning components),^{8,9} is introduced and then used to specify an operational multi-agent system.

DESIRE allows the system designer to explicitly and precisely specify both the intra-agent functionality (i.e., the expertise required to perform the domain tasks for which the agent is responsible in terms of the knowledge requirements and the reasoning capabilities) and the inter-agent functionality (i.e., the expertise required to perform and guide coordination, cooperation and other forms of social interaction in terms of the knowledge requirements and the reasoning capabilities). DESIRE views both the individual agents and the overall system as a compositional architecture - hence all functionality is designed as a series of interacting, task-based, hierarchically structured components. Tasks are characterised in terms of their inputs, their outputs and their relationship to other tasks. Interaction and coordination between components, between components and the external world, and between components and users¹⁰ is specified in terms of information exchange, sequencing information and control dependencies. The components themselves can be of any complexity (from simple functions and procedures up to whole knowledge-based systems) and can perform any domain function (e.g. numerical calculations, information retrieval, optimization, et cetera).

Although DESIRE was originally conceived as a means of specifying complex knowledge-based systems, its philosophy of viewing the system as a series of interacting components means it is ideally suited to the specification of multi-agent systems.^{6,7,11} DESIRE's philosophy contrasts with those of general purpose specification languages such as Z^{4,12,13} or VDM¹⁴ at precisely this point: by committing to a specific type of architecture for the design specification of multi-agent systems, DESIRE provides more structure and thus more support. General purpose specification languages, such as Z and VDM, provide less support in this respect. One of the goals in the design and (continual) development of the DESIRE framework is to provide constructs with which reasoning patterns can be explicitly modelled. This is an additional advantage with respect to general purpose specification languages in modelling multi-agent systems, as agents are most often capable of reasoning about both internal and external processes. The constructs included in DESIRE support modelling and specification of agents in this respect.

The main contribution of this work is that it presents an easy to use and expressive framework which allows multi-agent system designers to focus on the conceptual design and specification of their system (rather than having to worry about low-level system

programming issues). Tools such as graphical editors support the designer in this process. Moreover, DESIRE's high-level modelling environment can automatically generate prototype applications directly from the specifications.

DESIRE is an industrial strength system - it has been used by a number of companies and research institutes (such as chemical industry, financial sector, software industry, institutes for environmental studies) to develop operational systems for a number of complex tasks (including systems for diagnosis, design, routing, scheduling and planning). DESIRE has the additional advantage that the specifications and their semantics can be made formal, using temporal logic as a base.^{15,16,17} This allows to prove various properties about the system during the verification and validation phases of the software lifecycle.

This paper describes the DESIRE framework and shows how it can be applied to model and specify multi-agent systems. The exemplar application is that of electricity transportation management¹⁸ which is one of the few operational multi-agent systems currently in existence (Section 2). Section 3 introduces the DESIRE framework and Sections 4 to 6 show how the framework can be used to model and specify the electricity transportation application. In particular, Section 4 presents a generic formal model and specification of a compositional agent, Section 5 details the specific agents involved in the scenario in terms of their specific tasks, while Section 6 deals with interaction between the agents and between agents and the world. Finally, Section 7 discusses the results of this modelling exercise and highlights those aspects which require further work.

2. An Example Application Domain

The multi-agent system used as an illustration in this paper has been developed for the domain of electricity management in general and the management of an electricity transportation network in particular. The system described was developed in the ARCHON project and is currently running on-line in a control room in the North of Spain.^{18,19} 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. This paper focusses on four of these agents.

The *Control System Interface* agent (called 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 agent CSI periodically produces a snapshot which describes the entire system state at the current instant in time. Additionally, CSI supports the overall system's diagnosis and the restoration functionality by doing some pre-processing on the alarm messages. In the former case, it performs a preliminary analysis of the data it receives to determine whether a fault has occurred. This analysis is then used to initiate the system's diagnosis functionality. In the

latter case, CSI monitors the network to determine whether significant changes have occurred. This functionality ensures that the assumptions upon which the restoration phase is based are still valid as the restoration plan is executed.

The system's diagnosis functionality is provided by two separate agents - an *Alarm Analysis Agent* (called AAA) and a *Blackout Area Identifier* agent (BAI). Upon receipt of notification of possible faults from the agent CSI, both AAA and BAI become active. These two agents use the snapshot information transferred by CSI to update the models of the network they construct and maintain. The agents' diagnoses are based on the information represented in these models. The agent 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. The agent 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). Each of these hypotheses is taken in turn and a time consuming validation task is performed to determine the likelihood that the given hypothesis is the cause of the network fault.

Cooperation occurs between the agents 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. This task is performed by AAA's hypothesis refinement 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 - the agent BAI maintains a model of all the agents in the system (its acquaintance models) and its model of AAA specifies 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 the agent AAA will generate an information request to have the initial blackout area produced - this will, in fact, result in a request being directed to the agent BAI because AAA's acquaintance model of BAI indicates that it has a task which produces the initial blackout area as a result.

The final agent considered is a *Service Restoration Agent* (SRA) which generates a plan of action which can be used to repair the network once the cause and location of the fault have been determined. To this end, first candidate actions are proposed based on information about the nature of the fault (provided by AAA) and its extent (the black out area provided by BAI). Next these candidates are checked for feasibility and relevance. Finally, from the approved actions a repair plan is prepared. The execution of this plan is guided by the human control engineer (or *operator*) and is monitored by the agent CSI to ensure that any significant changes in the network state are reported to SRA so that it can make a decision about how to respond (e.g., to carry on regardless, to generate a new restoration plan from scratch or to modify the existing plan).

3. A Specification Framework for Multi-Agent Systems

Task models define the structure of *compositional architectures*: components in a compositional architecture are directly related to (sub)tasks in a task (de)composition. The hierarchical structures of tasks, interaction and knowledge, are fully preserved within compositional architectures. Often more than one agent is involved in the performance of a given task. Task coordination between agents then becomes essential. As agents, however, often are capable of performing one or more (sub)tasks, either sequentially or in parallel, task coordination within the agents themselves is also essential.

Below a formal compositional framework for modelling multi-agent tasks is introduced, in which

- (1) a task (de)composition,
- (2) information exchange,
- (3) sequencing of (sub)tasks,
- (4) subtask delegation, and
- (5) knowledge structures,

are explicitly modelled and specified.

3.1. Task (de)composition

In order to adequately model and specify (de)composition of tasks, knowledge is required of:

- a *task hierarchy*,
- information a task requires as *input*,
- information a task produces as a *result* of task performance
- *meta-object relations* between (sub)tasks (which (sub)tasks reason about which other (sub)tasks).

For each task in a task hierarchy a set of subtasks may be specified. Within a task hierarchy *composed* and *primitive* tasks are distinguished: in contrast to primitive tasks, composed tasks are tasks for which (a non-empty set of) subtasks are identified. Subtasks, in turn, can be either composed or primitive. Tasks are directly related to components: composed tasks are specified as composed components, and primitive tasks as primitive components, respectively.

An example of a *task hierarchy* for the task of electricity transportation management, independent of the agents involved, is shown below in Figure 1. The leaves represent the primitive tasks.

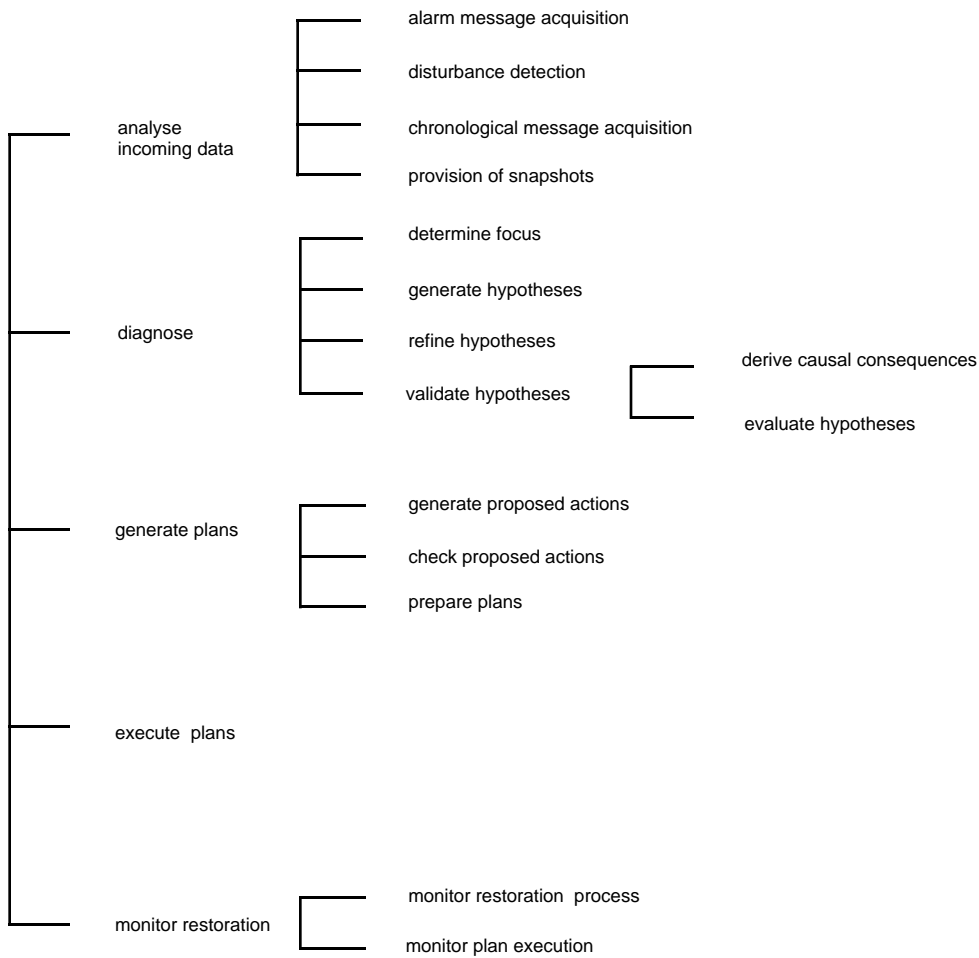


Fig. 1 Task hierarchy of electricity transportation management

Information required/produced by a (sub)task is defined as *input/output signature* of a component. The signatures used to name the information are defined in a predicate logic with a hierarchically ordered sort structure (*order-sorted predicate logic*). Units of information are represented by the (ground; i.e., instantiated) *atoms* defined in the signature.

The different roles information can play within reasoning can be distinguished by different (meta)levels, specified by the level of an atom within a signature. In a two level situation the lowest level is termed *object-level information*, and the second level *meta-level information*. Meta-level information expresses information about object-level information and reasoning processes; for example, for which atoms the values are still

unknown (*epistemic information*), or for which determination of the values is a goal for the reasoning process (*target information*). Accordingly, tasks that include reasoning about other tasks are indicated as meta-level tasks with respect to object-level tasks. Often more than two levels of information and reasoning are involved, resulting in meta-meta-level information and reasoning.

3.2. *Information exchange between tasks*

Information exchange between tasks is specified as *information links* between components. Each information link relates output of one component to input of another, by specifying which specific output atom and truth value are linked with which specific input atom and truth value. Atoms can be renamed; therefore, each component can be specified in its own language, independent of other components. The conditions for activation of information links are explicitly specified as part of the task control knowledge: knowledge of sequencing of tasks.

3.3. *Sequencing of tasks*

Task sequencing is explicitly modelled within components as *task control knowledge*. Task control knowledge includes not only knowledge of which subtasks should be activated when and how, but also knowledge of the control information associated with task activation (targets and requestables) and the amount of effort which can be afforded to achieve a goal to a given extent (exhaustiveness and effort). Subcomponents are, in principle, black boxes to the task control of an encompassing component: task control is based purely on information about the success and/or failure of the subcomponents' processes. The process of a component is considered to have been successful, for example, with respect to one of its target sets if it has reached the goals specified by this target set (and specifications of the number of goals to be reached (e.g., any or every) and the effort to be afforded).

3.4. *Delegation of tasks*

During knowledge acquisition a task as a whole is modelled. In the course of the modelling process decisions are made as to which (sub)tasks are best performed by which *agent*. This process, which in general may also be performed at run-time, results in the delegation of (sub)tasks to the parties involved in task execution. For electricity transportation management tasks can be divided over the participating agents as shown below in Figure 2.

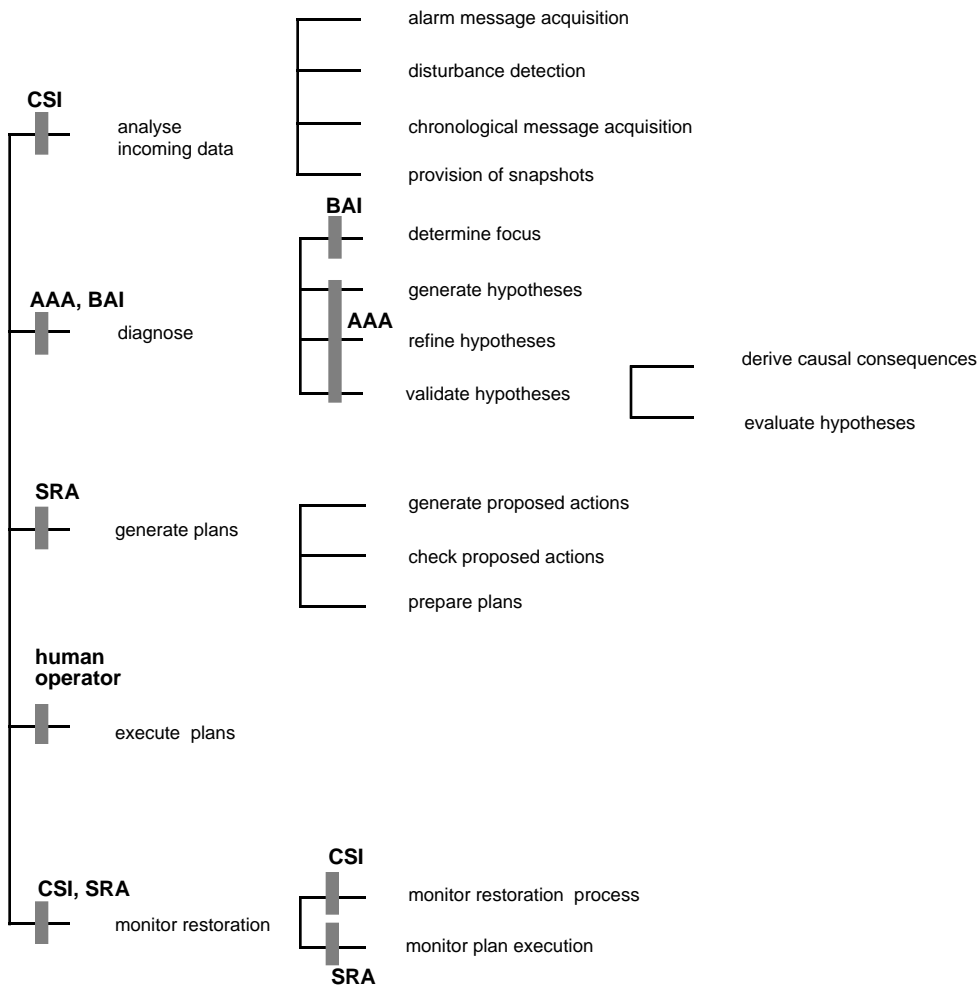


Fig. 2 Delegation of tasks to agents

3.5. Knowledge structures

During knowledge acquisition an appropriate structure for domain knowledge must be devised. The meaning of the concepts used to describe a domain and the relations between concepts and groups of concepts, must be determined. Concepts are required to identify objects distinguished in a domain, but also to express the methods and strategies employed to perform a task. Concepts and relations between concepts are defined in *hierarchies* and *rules* (based on *order-sorted predicate logic*). In a specification document references to appropriate knowledge structures (specified elsewhere) suffice.

4. A Formal Model of a Compositional Agent

The task hierarchy presented in Section 3.1 is a task hierarchy for the electricity transportation management task represented as a single composed task. Agents, however, not only perform tasks directly related to electricity transportation management; they also perform tasks related to their own internal process management and tasks related to interaction with other agents and with the world outside the agents (the external world). A generic decomposition of an agent is presented below in Section 4.1, a description of information exchange within an agent is described in Section 4.2 and in Section 4.3 task control knowledge within an agent is depicted.

4.1. A Generic Model of an Agent

The agents described in Section 2 have a number of tasks in common: control of their own processes, update of world state information, and management of communication with other agents. In addition each agent has one or more specific tasks to perform: e.g., diagnose fault (for the agent AAA), or identify blackout area (for the agent BAI), as shown in Figure 3. The levelled input and output interfaces depicted on the outer edges of the components in this figure indicate object-meta level distinctions in input and output. The agent task control (depicted at the top in the component) has interaction with each and every subcomponent. The task control links through which such interaction is achieved (between task control and subcomponents) are not explicitly specified nor depicted in graphical representations as shown in Figures 3 and 5. These links are predefined for all components in the DESIRE framework.

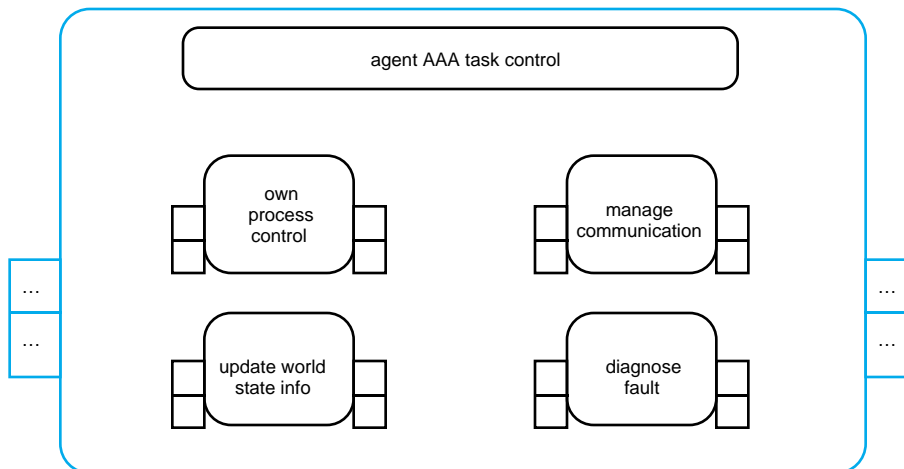


Fig. 3 Generic top level compositional structure of an agent

The specialisation and instantiation of each of the three generic tasks may differ significantly between agents: not only do agents reason on the basis of different types of information and interaction, they differ with respect to their models of the external world and the tasks they are capable of performing. A decomposition of the three generic tasks, one level deep, is shown in Figure 4.

The generic component responsible for the task of the central coordination of an agent's reasoning processes and other activities, `own_process_control`, uses (either internal or input) knowledge of:

- (1) its own abilities, beliefs, desires, intentions, motivations, et cetera,
- (2) the status of available knowledge about interaction with other agents,
- (3) the current state of affairs in the world outside the agent and
- (4) subtask performance to reason about its current "state of mind": current processes and knowledge.

This entails both monitoring the (status of) available knowledge and reasoning about the current state (often in relation to the past).

1. Own process control
 - 1.1 Monitor incoming data
 - 1.2 Evaluate process state
2. Update world state information
3. Agent specific tasks
4. Manage communication
 - 4.1 Examine agent model
 - 4.2 Generate information
 - 4.3 Receive information

Fig. 4 Generic task decomposition of an agent

The task of managing communication with other agents is a task in itself: the generic task assigned to the component `manage_communication`. To this purpose agents may have representations of other (relevant) agents: often of agents with whom interaction is known to be required. Agents are capable of examining these models, of generating information for other agents, and of receiving information from other agents. This information may be object-level information (facts about the world), but it may also be meta-information such as requests for specific object level information.

The task of maintaining a representation of the current state of the world outside an agent, is also a generic agent task, which all agents should be capable of performing, however limited the representation may be. The component `update_world_state_info`, includes knowledge of how an agent's "personal" representation of the world is acquired. An agent's representation will often differ from the real state of the world, and from other agent's representations of the world.

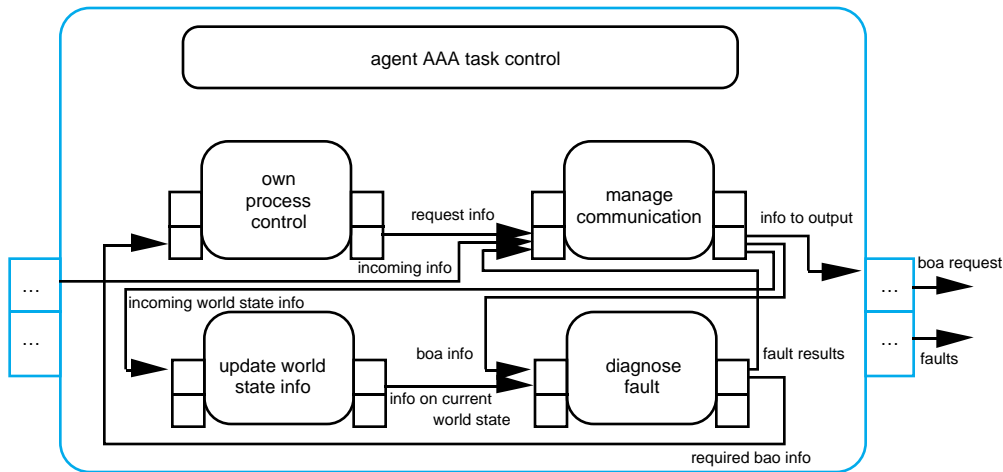


Fig. 5 Information exchange of agent AAA's top level

4.2. Information exchange at the top level of an agent

Information links are defined to model information exchange between components within an agent. In general, information communicated to an agent will be transferred directly from the input interface to the agent's component `manage_communication`. Comparably, information meant for other agents will be transferred from the component `manage_communication` to the outer interface of an agent. In more extended models of agents, an agent's `own_process_control` may frequently be informed of progress of the agent's processes: in the models of information exchange within an agent presented in this paper, these links have been omitted. The nature of the individual agent's processes in the electricity management task are relatively straightforward and do not require extended process control. To illustrate the way in which information exchange is modelled for this task the information links of one specific agent, namely AAA are described. AAA's top level tasks and links are depicted in Figure 5. As described in Section 4.1, task control links are not explicitly specified nor depicted.

The information links used within a component are specified as part of the task structure; for agent AAA, the specification of its *task structure* is as follows:

```

task structure AAA
  subcomponents own_process_control, update_world_state_info,
    diagnose_fault, manage_communication;
  links incoming_info, incoming_world_state_info, info_on_current_world_state,
    boa_info, request_info, required_bao_info, fault_results, info_to_output,
    boa_request, faults;
end task structure AAA

```

Agent AAA receives two types of information: (1) information about the state of the network and (2) information about the blackout area. This information is transferred from the outer input interface of AAA through the link `incoming_info` to the component `manage_communication`. Information about the state of the network is then transferred from the component `manage_communication` to the component `update_world_state_info` through the link `incoming_world_state_info` and the information about the blackout area is transferred to the component `diagnose_fault` through the link `boa_info`. Information about the state of the network is forwarded from the component `update_world_state_info` to the component `diagnose_fault` through the link `info_on_current_world_state`.

The component `diagnose_fault` may recognize a lack of information on a blackout area. The (meta)fact that there is no blackout area information is transferred to the component `own_process_control` through the link `required_bao_info`. The component `own_process_control` establishes that the lacking blackout area information should be acquired through communication; this conclusion is transferred to the component `manage_communication` through the link `request_info`. After determination of the agent(s) to be addressed, the request is then transferred by the link `info_to_output` to the output interface of the component. The link `boa_request` transfers this information to the agent BAI. Newly acquired information on a blackout area is transferred by the component `manage_communication` to the component `diagnose_fault` as described above.

A resulting fault, as diagnosed by `diagnose_fault` is transferred through the link `fault_results` to the component `manage_communication` and to interested agents (in particular, SRA) through the links `info_to_output` and `faults`.

Note the level difference between, on the one hand, the components `own_process_control` and `manage_communication`, and, on the other hand, the components `update_world_state_info` and `diagnose_fault`. The components `own_process_control` and `manage_communication` reason about the status of the processes within `update_world_state_info` and `diagnose_fault`. The level distinction between the origin of a link and its destination is expressed in the interfaces of the relevant components. E.g., the link `required_bao_info` links the higher level of the output interface of `diagnose_fault` with the lower level of the input interface of `own_process_control`.

Links and components are formally specified within DESIRE. An example of an *information link specification* (see Sections 3.1.2 and 3.2.2) within AAA is the specification of the link `info_on_current_world_state` between the component `update_world_state_information` and the component `diagnose_fault`:

```

link   info_on_current_world_state: object-object
domain   update_world_state_information
output   world_state_obs
codomain diagnose_fault
input    world_state_obs
sort links (World_state_info,World_state_info)
object links identity
term links   identity
atom links

```

```
(obs_in_current_world_state(!:World_state_info),obs_in_current_world_state(!:World_state_info)):  
  <<true,true>,<false,false>>
```

endlink

This link is defined to transfer information observed in the current world state from the component `update_world_state_information` (the domain of the link) to the component `diagnose_fault` (the codomain of the link). The truth value (`true`, resp. `false`) of the atom `obs_in_current_world_state(!:World_state_info)` is transferred as specified by the atom links from `update_world_state_information` to `diagnose_fault`. The information observed is of the sort `World_state_info`, known in both domains (linked by sort links). Object and term links relate the syntactic entities within the sorts. In this example the linked entities are syntactically identical.

4.3. Task Control of Agents

Control of agents and interaction between agents is specified in precisely the same way as control of components and interaction between components within an agent. Depending on the autonomy of agents and their components, specification of control will differ. In 4.3.1 an example of autonomous agents is described. In 4.3.2 an example of control within an agent is discussed.

4.3.1. Autonomous agents

Task control of agents in the electricity transportation management task at the highest (global) level is minimal. An example of a specification of a rule for agent activation for AAA is shown below.

```
if start  
then next_component_state(AAA, awake)
```

At the very beginning of system activation, agent AAA is awakened. The agent is fully autonomous, capable of processing any new input it receives, as soon as it arrives, and capable of transferring its output on to other agents as soon as this is required.

4.3.2. Control within agents

Within agents, components can be autonomous, or they may be controlled. In the electricity transportation management task, control within agents is well-defined. The agents' task control knowledge specifies when and how components and links are to be activated (and whether activation is continuous or temporarily instantiated). Evaluation criteria are used for this purpose within temporal rules. Each component is assumed to have a (local, linear) discrete time scale.

The specification of agent AAA's task control knowledge illustrates the way in which such knowledge is specified. Control over AAA's four subtasks is limited: the rules presented in this section express the knowledge required to specify interaction between

the four subcomponents. Activation of components does not always depend on the completion of a specific component. In some cases receipt of 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 the keyword *awake*. AAA's components `own_process_control` and `manage_communication`, and all internal links become and remain awake once AAA has been activated. This is expressed by:

```
if start
then next-component-state(own_process_control, awake)
and next-component-state(manage_communication, awake)
and next-target-set(manage_communication, new_world_info)
and next-link-state(incoming_info, awake)
and next-link-state(incoming_world_state_info, awake)
and next-link-state(info_on_current_world_state, awake)
and next-link-state(boa_info, awake)
and next-link-state(request_info, awake)
and next-link-state(required_bao_info , awake)
and next-link-state(fault_results, awake)
and next-link-state(info_to_output, awake)
and next-link-state(boa_request, awake)
and next-link-state(fault, awake)
```

Here `next-target-set(manage_communication, new_world_info)` specifies that the target set `new_world_info` is the focus of the activity of the component `manage_communication`. A typical example of a component's task control knowledge rule in which the success of one component is required (whether or not incoming world state information has been monitored by the component `manage_communication`), before a following component (`update_world_state_info`) can be activated with the required information, is the following:

```
if evaluation(manage_communication, new_world_info, succeeded)
then next-component-state(update_world_state_info, active)
and next-target-set(update_world_state_info, new_world_state_info)
```

This knowledge rule states that

```
if the component manage_communication, has succeeded in accomplishing the
   targets defined by its target set new_world_info,

then the component update_world_state_info is assigned a new set of targets,
   namely new_world_state_info, and activated.
```

Note that the output of the component `manage_communication` is transferred automatically to `update_world_state_information` by the link `incoming_world_state_info` that is awake.

The component `diagnose_fault` is activated when input information is received on alarms as specified below:

```
if    evaluation(update_world_state_info, new_alarms, succeeded)
then  next-component-state(diagnose_fault, active)
and   next-target-set(diagnose_fault, faults)
```

5. Task Models for the Agent Specific Tasks

As discussed above in Section 4 each agent in the electricity transportation management has 3 generic tasks and an agent specific task. In this section (parts of) the models of the agent specific tasks are presented: task decomposition and information exchange for all agent specific tasks and task control for AAA's agent specific task.

5.1. Task model of the Agent Specific Task of AAA

The agent AAA is responsible for identifying possible faults in the electricity network on the basis of information from CSI and blackout area information provided by BAI. The agent SRA analyses these faults to devise a restoration plan.

1. Own process control
 - 1.1 Monitor incoming data
 - 1.2 Evaluate process state
2. Update world state information
3. Diagnose fault (agent specific task)
 - 3.1 Generate hypotheses
 - 3.2 Refine hypotheses
 - 3.3 Validate hypotheses
 - 3.3.1 Derive causal consequences
 - 3.3.2 Evaluate hypotheses
4. Manage communication
 - 4.1 Examine agent model
 - 4.2 Generate information
 - 4.3 Receive information

Fig. 6 Complete task hierarchy of agent AAA

The task hierarchy of AAA's agent specific task - diagnose fault - is shown in Figure 6, as part of the complete task hierarchy of agent AAA.

5.1.1. Decomposition and information exchange of diagnose_fault

In Figure 7 the task decomposition of the agent specific task diagnose fault is depicted together with the information exchange between subtasks. The component's task control is depicted at the top, but, as explained in Section 4.1, the task control links are neither specified nor graphically depicted. This model shows the link `incoming_world_state_info_to_gh`

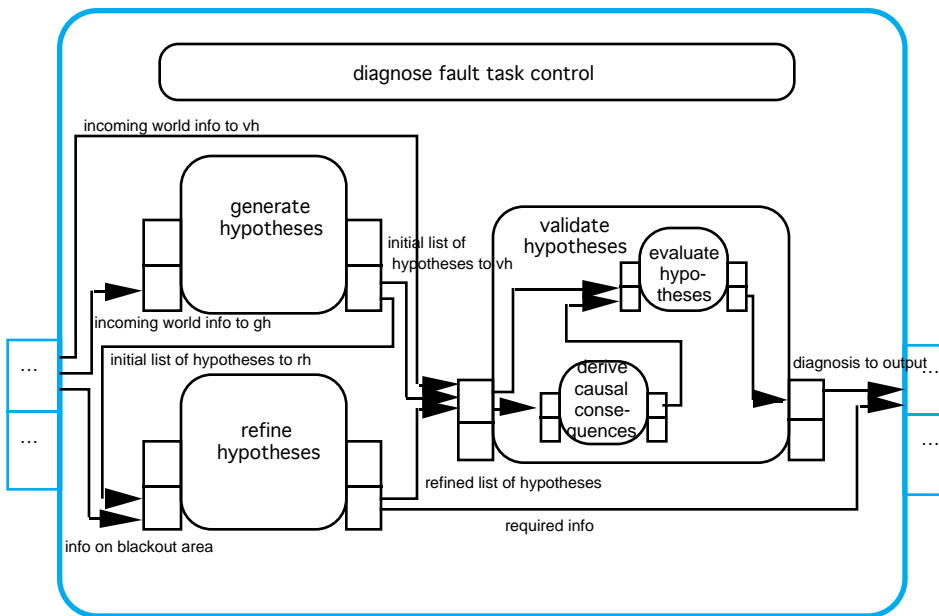


Fig. 7 The component diagnose fault of agent AAA

through which world state information is transferred from the outer input interface to the component `generate_hypotheses`. The component `generate_hypotheses` produces an initial list of hypotheses which is forwarded to two components: to the component `refine_hypotheses` (through the link `initial_list_of_hypotheses_to_rh`) and to the component `validate_hypotheses` (through the link `initial_list_of_hypotheses_to_vh`). If additional information on the blackout area is available (transferred through the link `info_on_blackout_area`) the component `refine_hypotheses` prunes the list of hypotheses on the basis of this information, providing a more specific list of hypotheses to be transferred to the component

validate_hypotheses through the link refined_list_of_hypotheses. If not, the component refine_hypotheses recognizes the need for additional information and the required information is transferred to the output interface of the component diagnose_fault through the link required_info. The component validate_hypotheses examines the hypotheses it has received one by one (either from generate_hypotheses or refine_hypotheses), and transfers plausible hypotheses (hypotheses that have not been rejected) to the output interface of the component diagnose_fault through the diagnosis_to_output link. The structure of the component diagnose_fault and the information links are specified in the same way as the structure of agent AAA was specified in Section 4.1, as shown below.

```

task structure diagnose_fault
subcomponents generate_hypotheses, refine_hypotheses, validate_hypotheses;
links          incoming_world_info_to_gh,
                incoming_world_info_to_vh, info_on_blackout_area,
                initial_list_of_hypotheses_to_rh, initial_list_of_hypotheses_to_vh,
                refined_list_of_hypotheses, required_info, diagnosis_to_output;
end task structure diagnose_fault

```

The information links within the component diagnose_fault are also specified in precisely the same way as the links within AAA. An example of an information link specification is depicted below.

```

link          initial_list_of_hypotheses_to_vh: object-object
domain        generate_hypotheses
output        poss_hyps
codomain      validate_hypotheses
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 transfers output of the component generate_hypotheses to input of the component validate_hypotheses. The truth values of the atoms transferred remain the same, but the atoms are renamed. An output atom of generate_hypotheses, the atom poss_hyp(H:Hyps) - is linked to the input atom of validate_hypotheses, the atom hyp(H:Hyps).

The component validate_hypotheses has two subcomponents: derive_causal_consequences and evaluate_hypotheses. The component derive_causal_consequences reasons about the world hypothetically, on the basis of causal knowledge and the hypothesis to be validated. The component evaluate_hypotheses reasons about the outcome of this hypothetical reasoning and the information observed in the external world. If discrepancies are identified the hypothesis is rejected. The component evaluate_hypotheses models a meta-level task

(namely, reasoning about the process of rejection of hypotheses) with respect to the component `derive_causal_consequences` (which performs reasoning about the world). This object-meta relation is depicted by the arrows at different levels within the component `validate_hypotheses`.

5.1.2. Task control knowledge of `diagnose_fault`

Specification of task control knowledge of the component `diagnose_fault` is comparable to the specification of task control knowledge of AAA. Activation of `diagnose_fault` results in activation of the subcomponent `generate_hypotheses`.

```
if    component-state(diagnose_fault, start)
then  next-component-state(generate_hypotheses, active)
and   next-target-set(generate_hypotheses, poss_hyps_found)
and   next-link-state(incoming_world_state_info_to_gh, awake)
and   next-link-state(incoming_world_state_info_to_vh, awake)
and   next-link-state(initial_list_of_hypotheses_to_rh, awake)
and   next-link-state(info_on_blackout_area, awake)
and   next-link-state(required_info, awake)
and   next-link-state(diagnosis_to_output, awake)
```

This rule states that once the component `diagnose_fault` has been activated, the component `generate_hypotheses` is to be activated with target set `poss_hyps_found` and that all links but two (namely `initial_list_of_hypotheses_to_vh` and `initial_list_of_hypotheses_to_rh`) are awakened.

If hypotheses have been generated successfully, `refine_hypotheses` is awakened:

```
if    evaluation(generate_hypotheses, poss_hyps_found, succeeded)
then  next-component-state(refine_hypotheses, awake)
and   next-target-set(refine_hypotheses, status_ref_hyps_found)
```

If no blackout area information is available, the component `refine_hypotheses` succeeds for the target set `no_blackout_area_info` and this fact is transferred through the link `required_info` to the outer interface of the component `diagnose_fault` and the component `validate_hypotheses` is activated with the initial list of hypotheses produced by the component `generate_hypotheses`. The initial list of hypotheses is transferred by explicit activation of the link `initial_list_of_hypotheses_to_vh` and explication of the target set faults.

```
if    evaluation(refine_hypotheses, no_blackout_area_info, succeeded)
then  next-component-state(validate_hypotheses, active)
and   next-target-set(validate_hypotheses, faults)
and   next-link-state(initial_list_of_hypotheses_to_vh, up_to_date)
```

If blackout information is available the component `refine_hypotheses` succeeds for the target set `ref_hyps_found` and the component `validate_hypotheses` is activated, as specified below. In addition, the refined list of hypotheses is transferred to `validate_hypotheses` by explicit activation of the link `refined_list_of_hypotheses`.

```
if    evaluation(refine_hypotheses, ref_hyps_found, succeeded)
then  next-component-state(validate_hypotheses, active)
and   next-target-set(validate_hypotheses, faults)
and   next-link-state(refined_list_of_hypotheses, up_to_date)
```

If blackout area information is made available while the component `validate_hypotheses` is active, `validate_hypotheses` is reactivated (by the rule specified above) and the new list of hypotheses is transferred (by activation of the link `refined_list_of_hypotheses`), replacing the initial list of hypotheses provided by `generate_hypotheses`. This new input to the component `validate_hypotheses` leads to revision of the validation process.

Task control of `validate_hypotheses` is straightforward: it specifies that the components `derive_causal_consequences` and `evaluate_hypotheses` are activated sequentially. The specification has been omitted.

5.2. Task Model of the Agent Specific Task of CSI

The agent CSI (the Control System Interface) periodically produces a snapshot which describes the entire state of the network 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. The agent CSI's specific task can be subdivided into the following subtasks:

3. Analyse incoming data (agent specific task)
 - 3.1 Alarm messages acquisition
 - 3.2 Chronological messages acquisition
 - 3.3 Disturbance detection
 - 3.4 Snapshot provision
 - 3.5 Monitoring restoration process

The information exchange between these tasks is depicted in Figure 8. CSI receives network snapshots every 15 minutes. This information is transferred directly to the component `provision_of_snapshots`. The component `alarm_messages_acquisition` receives alarm messages from the external world through the link `world_info_to_ama` and combines them into blocks of alarms. The component `disturbance_detection` receives these blocks of alarms through the link `alarms` and identifies indications of failure. These indications are transferred to the output interface of `analyse_incoming_data`.

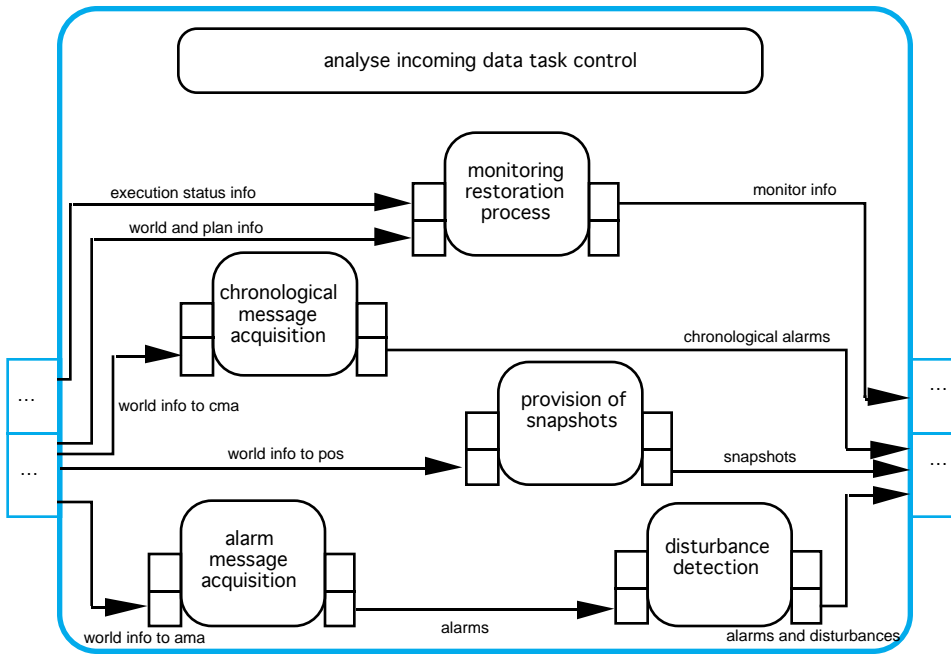


Fig. 8 The component analyse incoming data of the agent CSI

The component `chronological_message_acquisition` uses both chronological and non-chronological alarms (for missing information) received from the external world to produce blocks of chronological alarms. These blocks of chronological alarms are transferred to the output interface of `analyse_incoming_data`. Once the component `monitoring_restoration_process` has received information stating that the operator has started plan execution, it monitors the restoration process on the basis of incoming snapshot information and the current restoration plan.

5.3. Task Model of the agent specific task of BAI

The agent BAI (blackout area identifier) is a fast and relatively unsophisticated system which can pinpoint the approximate region of a fault. This region, the blackout area, is used to prune the list of possible hypothetical faults generated by AAA. The agent BAI's specific task is `determine_focus`. This task is subdivided into two subtasks, as shown below.

3. Determine focus (agent specific task)
 - 3.1 Determine initial focus
 - 3.2 Determine final focus

These tasks are depicted below in Figure 9 together with information exchange.

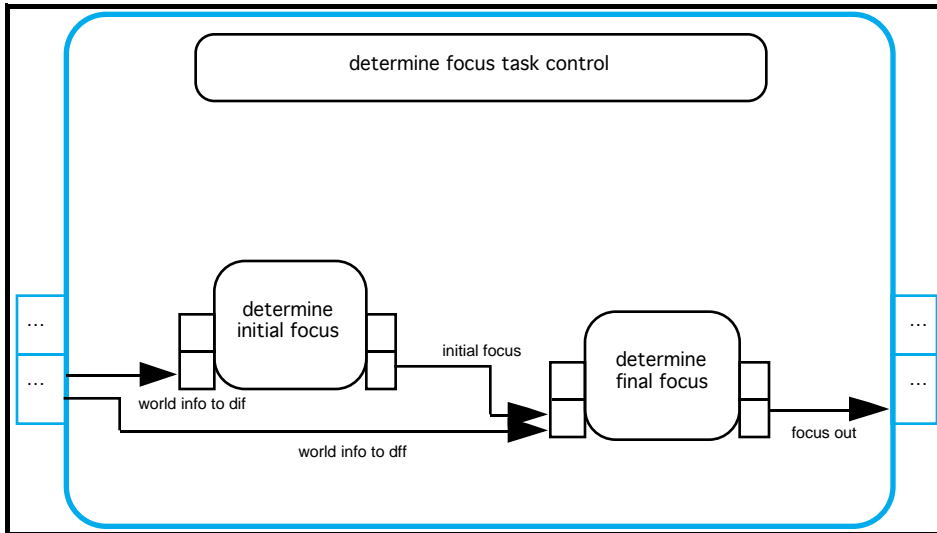


Fig. 9 The component determine focus of the agent BAI

The component `determine_initial_focus` uses part of the world state information transferred to `determine_focus`, namely indications of failure, to determine which elements are in the blackout area during the first notification of a disturbance. The link `initial_focus` transfers the result, an initial focus, to the component `determine_final_focus`. The component `determine_final_focus` uses other world state information, namely snapshot information, transferred through the link `world_info_to_dff`, to confirm whether elements are still in the initial focus area during the phase after the disturbance took place. The result, a final focus, is transferred to the output interface of `determine_focus` through the link `focus_out`.

5.4 . Task Model of the Agent Specific Task of SRA

The agent SRA (the System Restoration Agent) generates a plan of action which can be used to repair the network once the cause and location of the fault have been determined. The task hierarchy of SRA's agent specific tasks is the following:

3. Determine plans and monitor plan execution (agent specific tasks)

- 3.1 Determine Plans
 - 3.1.1 Generate proposed actions.
 - 3.1.2 Check proposed actions.
 - 3.1.3 Prepare plans.
- 3.2 Monitor plan execution.

The hierarchical structure and information exchange depicted in Figure 10 shows how the two main tasks of this component are clearly distinguished.

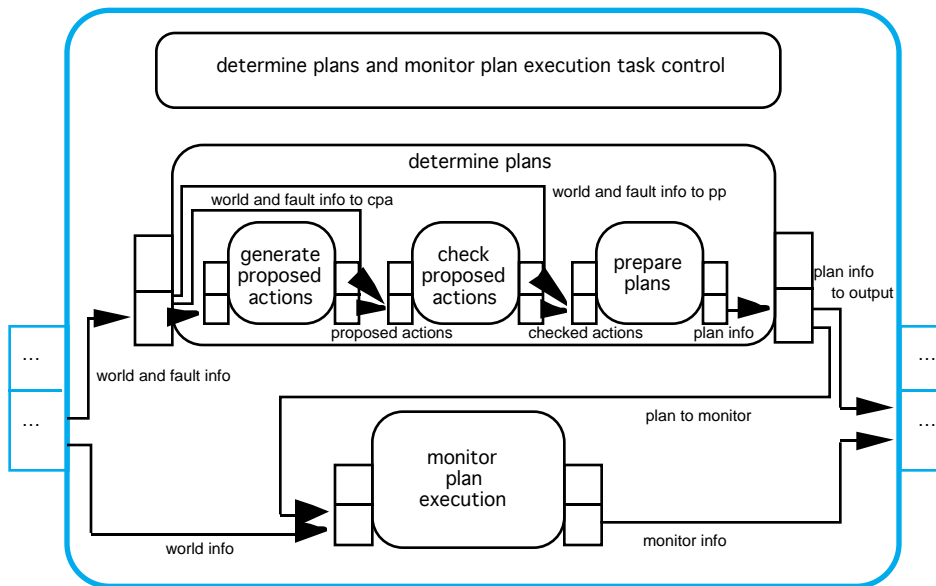


Fig. 10 The component determine plans and monitor plan execution of the agent SRA

The component `determine_plans` receives information on the faults detected, the blackout area identified, and the current state of the world which it uses to prepare a plan to restore the network. This plan is transferred to the output interface of the component `determine_plans_and_monitor_plan_execution`. Once `determine_plans_and_monitor_plan_execution` receives information that the operator has started plan execution, the component `monitor_plan_execution` evaluates the status of the network (information received on the state

of the world) in relation to the plan. This information is transferred to the output interface of the component.

6. Interaction between Agents and between Agents and the External World

Agents are not only capable of exchanging information with other agents, they are also capable of exchanging information with the external world. Although the two types of interaction can be realised by essentially the same mechanisms, they differ conceptually. Interaction with the external world is discussed below in Section 6.1, interaction between agents in Section 6.2.

In Figure 11 the information links between agents, and between agents and the external world, are depicted, together with distinctions between the levels of the information transferred. An example scenario is used in Section 6.3 to describe the patterns of interaction modelled for electricity transportation management.

6.1. Interaction between Agents and the External World

Interaction between an agent and the external world is modelled almost identically from the agent's point of view to interaction between agents. Information links are defined between specific agents and the external world for the purpose of either observation or action performance. Observation of the external world may be modelled as an agent's specific request for information from the external world. The agent's component `own_process_control` determines that such information is required and after activation of the agent's component `manage_world_interaction` (which has not been modelled in the electricity transportation management example) the request is transferred to the agent's output interface. Once this meta-information has reached the agent's interface, it is transferred to the external world. As a result of the request, object-level information may be transferred through another link back to the requesting agent. The external world includes information on the current state of the world, so that the agent can be made aware of any changes, if it so requests.

Performance of a specific action may be modelled as an update of the external world state. Once an agent has determined that an action is to be performed, information about action performance is transferred to the external world, resulting in a change of the state of the external world.

6.2. Communication between Agents

Patterns of communication between agents are made explicit by specification of information links, together with agent-specific knowledge of when information is to be transferred. An agent may, for example, decide to send specific information to one or more other agents once every 15 minutes, instigated by the agent's component `manage_communication`. This information is transferred through a link to its output interface, after which the information is transferred through one or more other links to one or more other agents. The component `manage_communication` uses knowledge of other agents,

knowledge of its own plans and knowledge of the information available/requested, to determine which information to communicate to which agent. The receiving agent may not choose to use the information immediately, but once the information has been sent, it is assumed to have been received by the other agent.

Another pattern of communication frequently encountered in multi-agent environments is modelled for situations in which agents realise that additional information on a specific topic is required. Again, on the basis of an agent's knowledge of other agents (including knowledge of the types of information individual agents can possibly provide) an agent determines which information to request from which other agent(s). The links between the agents and the mechanisms within individual agents (an agent's component `manage_communication` and links for information exchange within the agent) required to issue and receive requests, are explicitly defined. The component `own_process_control` of an agent recognizes the need for additional information and informs the component responsible for managing communication. This component reasons about other agents and prepares specific requests to one or more agents. The requests are transferred to the agent's output interface, after which the information is transferred through relevant links to other agents. Note that requests for specific information are transferred as meta-information (about object-level information) to other agents. If one of these agents is capable of providing the information requested and is willing to do so, this information is transferred back through another information link.

6.3. An example scenario

The information links modelled for the electricity transportation management task both between agents, and between agents and the external world, are shown in Figure 11. The interaction patterns are described below.

The control system of the network (part of the external world) sends alarms and snapshots of the network to CSI. This agent groups alarms, detects indications of disturbances, and provides snapshot information to interested agents. CSI's component `manage_communication` has models of other agents on which it bases the decision to forward the information to BAI, AAA and SRA. Agents AAA and BAI use this information to analyse the status of the network. In addition to information acquired from CSI, the agent AAA may also acquire information on the blackout area from BAI. The agent AAA uses this information to focus its process of diagnosis. If this information was not already provided, AAA notices this and specifically sends a request; this example of agent communication between AAA and BAI is explained below in more detail.

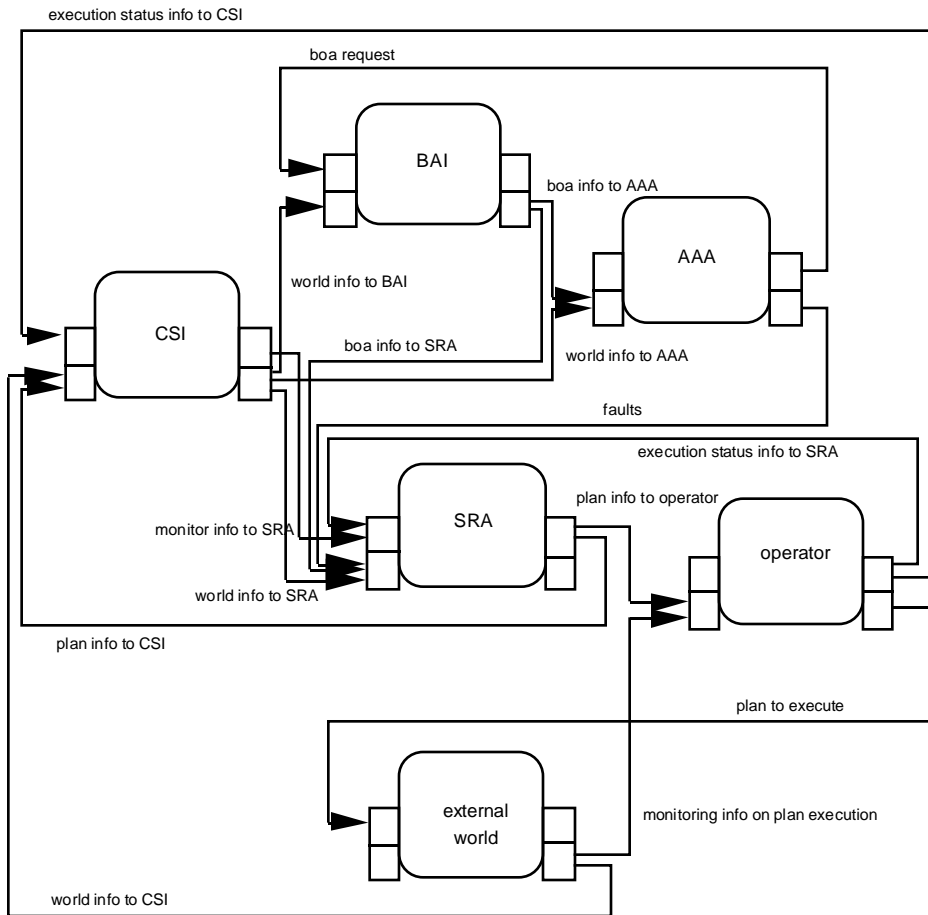


Fig. 11 Interaction between the agents and between agents and the world

The information link `boa_request` in Figure 11, linking meta-information stating that blackout area information is needed is specified by:

```

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

```

The agent AAA initiates this information exchange: the update of this information link is specified by task control knowledge of AAA on the basis of the results of the component `manage_communication` (see Section 4). AAA's `own_process_control` realises that it does not have any information about a possible blackout area, and that this information has to be acquired from another agent. This information is transferred to the component `manage_communication`. This component consults the models of other agents to determine from which agents this information can be acquired and decides to request this information from BAI. The agent AAA's task control knowledge transfers the output of `manage_communication` to AAA's outer interface. The agent BAI receives the output through the information link `boa_request`. The information link from BAI to AAA, `boa_info_to_AAA`, is updated if BAI has blackout area information to transfer to AAA. BAI's task control knowledge specifies that blackout area information produced by the component `determine_focus` (modelling BAI's agent specific task) and transferred to BAI's component `manage_communication`, is made available to the agents determined by `manage_communication`. This is effectuated by transferring the information to BAI's output interface, and from the output interface to the other agent's (i.e., AAA) input interface. AAA uses this information together with the other information it has acquired, and its own knowledge, to produce a list of hypotheses about possible faults. This list is transferred to SRA.

In addition to the information received from CSI on the state of the world, SRA receives the faults diagnosed by AAA from AAA and information about the blackout area from BAI. SRA uses this information to prepare a plan which it sends to the operator for execution and to CSI to monitor at network level. The operator executes the plan in the external world, informing SRA and CSI that execution has commenced. CSI detects when changes have occurred in the network which are likely to require a replanning endeavour and informs SRA of the deviations. This exchange of information is depicted by the the meta-level information link `monitor_info_to_SRA` between CSI and SRA.

7. Discussion

The declarative compositional framework DESIRE provides a principled architecture concept for agent design in which complex (reflective) reasoning within agents is explicitly modelled, as are communication patterns between agents and interaction with the external world. The framework supports conceptual design and specification of both dynamic and static aspects of agent behaviour and of the interaction between agents, and between agents and the external world. Depending on the situation at hand agents can be designed to be autonomous or fully controlled or anything in between. Interaction can be modelled by explicit activation of information links between agents, or as continuous processes of information exchange between agents, depending on the type of information and the role it plays in the processes modelled. In principle, the framework should be able to support the analysis and specification of a wide variety of agents: from simple agents to more complex agents, from weak agents to strong agents, et cetera.

Explication of the knowledge involved in reasoning, the information exchanged between agents and knowledge involved in task control within and between agents, is essential in modelling and specifying multi-agent systems. Earlier papers^{6,7} focussed on modelling and specifying task control within agents within the framework DESIRE. In this paper, in addition the exchange of information within agents and the communication between agents have been more closely analysed, modelled and specified, for a number of different agents in an industrial application.

There were two main perceived benefits of using DESIRE to help specify the electricity management application. Firstly, it provided a much clearer and more readily comprehensible description of the application than the more informal and descriptive technique which was originally used. It is felt that if the original specification had been produced using DESIRE then the subsequent implementation could have proceeded more rapidly. The second benefit was that the DESIRE specification highlighted a number of oversights in the original specification. Some cases of interactions and control decisions were found to be missing from the informal specification. Unfortunately this evidence is only anecdotal at present and it is also unclear as to how much of the aforementioned benefit is attributable to using a formal specification tool as opposed to using DESIRE in particular. Future work aims to place this assessment on a more substantial grounding.

In on-going research, informational and motivational aspects of agent characteristics such as beliefs, desires, intentions, commitments, and cooperation are being analysed, modelled, and specified within the DESIRE framework. A cooperation model for project coordination based on joint intentions and a model for collective user satisfaction in cooperative environments have also been developed within the framework.^{20,21}

Formal semantics of the specification language are based on temporal logic.^{9,15,16,17} Recent research on validation and verification based on these temporal semantics identifies some useful notions²²; an interesting and important topic in particular for safety critical systems.

Acknowledgements

An earlier version of this paper was read and commented upon by Catholijn Jonker. The research reported here was partly supported by the ESPRIT III Basic Research project 6156 DRUMS II on Defeasible Reasoning and Uncertainty Management Systems and by the Polish KBN Grants 3 P406 019 06 and 8T11C03110, and the programme MeDiCiS.

References

1. Rao, A.S. and M.P. Georgeff, Modeling rational agents within a BDI- architecture. In: R. Fikes, E. Sandewall (eds.), *Proc. of the Second Conference on Knowledge Representation and Reasoning, KR'91*, Morgan Kaufman, 1992, pp. 473-484.
2. Fisher, M. and Wooldridge, M., (1993) Specifying and Verifying distributed intelligent systems In: *Proc of Sixth Portugese Conf. on AI*, Portugal, pp. 13-28.
3. Burkhard, H. D., (1993) Liveness and Fairness Properties in Multi-Agent Systems. *Proc. 13th Int. Joint Conf. on AI, IJCAI'93*, Morgan Kaufman, pp. 325-330.
4. Luck, M., d'Inverno, M. (1995). A formal framework for agency and autonomy. In: V. Lesser (ed.), *Proc. of the First International Conference on Multi-Agent Systems, ICMAS'95*, AAAI Press/MIT Press, Cambridge, pp. 254-260
5. O' Hare, G. M. P., (1996) Agent Factory: An Environment for the Fabrication of Multi-Agent Systems. In: G. M. P. O'Hare and N. R. Jennings (eds.), *Foundations of Distributed Artificial Intelligence* , Wiley Interscience, pp. 449-484.
6. Brazier, F.M.T., B. M. Dunin-Keplicz, N.R. Jennings, J. Treur (1995). Formal Specification of Multi-Agent Systems: a Real World Case. In: V. Lesser (ed.), *Proc. of the First International Conference on Multi-Agent Systems, ICMAS-95*, AAAI Press/MIT Press, pp. 25-32.
7. Brazier, F.M.T., B. M. Dunin-Keplicz, N.R. Jennings, J. Treur (1996). Modelling Distributed Industrial Processes in a Multi-Agent Framework. In: S. Kirm, G.M.P. O'Hare (eds.), *Cooperative Knowledge Processing*, Springer Verlag. (to appear)
8. Langevelde, I.A. van, A.W. Philipsen and 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.
9. 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.
10. 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 AI, vol. 867, pp. 283-300.
11. 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
12. Diller, A. Z. (1992) *An Introduction to Formal Methods*. Wiley, .
13. Spivey, J.M. (1992) *The Z Notation. A Reference Manual*, 2nd Edition. Prentice-Hall, .
14. Jones, C.B. (1990). *Systematic Software Development Using VDM*, 2nd Edition. Prentice-Hall, 1990.

15. 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. Lecture Notes in AI, vol. 838, Springer Verlag, pp. 279-299. Also in: *Journal of Applied Non-Classical Logics*, vol. 5, Special issue with selected papers from JELIA'94, 1995, pp. 239-261.
16. 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.
17. Treur, J. (1994). Temporal Semantics of Meta-Level Architectures for Dynamic Control of Reasoning. In: L. Fribourg, F. Turini (eds.), *Logic Program Synthesis and Transformation - Meta-Programming in Logic, Proc. META'94*. Lecture Notes in Computer Science, Vol. 883, Springer Verlag, pp. 353-376.
18. Jennings, N. R. , J. Corera, I. Laresgoiti, E. H. Mamdani, F. Perriolat, P. Skarek and L. Z. Varga (1996). Using ARCHON to develop real-world DAI applications for electricity transportation management and particle accelerator control, *IEEE Expert - Special issue on Real World Applications of DAI* (to appear).
19. Cockburn, D. and N. R. Jennings (1995). ARCHON: A Distributed Artificial Intelligence System for Industrial Applications. In: G. M. P. O'Hare and N. R. Jennings (eds.), *Foundations of Distributed Artificial Intelligence*, Wiley & Sons, pp. 319-344.
20. Brazier, F.M.T., Jonker, C.M., Treur, J., (1996) Modelling Project Coordination in a Multi-Agent Framework. In: *Proc. Fifth IEEE Workshops on Enabling Technology: Infrastructure for Collaborative Enterprises, WET ICE'96*, IEEE Computer Society Press. (to appear)
21. Brazier, F.M.T., Ruttkay Zs. (1993). Modelling collective user satisfaction. In: *Proc. of HCI International'93*, Elsevier, Amsterdam, 1993, pp. 672-677.
22. 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*, Chambéry, pp. 189-199.