

# Modelling Distributed Industrial Processes in a Multi-Agent Framework<sup>\*</sup>

Frances M.T. Brazier<sup>a</sup>, Barbara Dunin-Keplicz<sup>b</sup>, Nick R. Jennings<sup>c</sup> and Jan Treur<sup>a</sup>

<sup>a</sup> 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

<sup>b</sup> University of Warsaw  
Institute of Informatics, ul. Banacha 2, 02-097 Warsaw, Poland  
Email: keplicz@mimuw.edu.pl

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

## Abstract

A declarative compositional modelling framework, DESIRE, designed to model knowledge-intensive multi-agent systems, is shown to provide a means to model distributed industrial and business processes. An agent's knowledge, reasoning processes and interaction with other agents, and the world are explicitly specified within this framework. Electricity transportation management is used to illustrate the characteristic elements of the approach, in particular with respect to dynamic aspects of distributed industrial and business processes; aspects which are of importance to knowledge management and knowledge engineering.

## 1 Introduction

Automation of industrial and business processes has focussed primarily on modelling information available within and applicable to an organisation. Large quantities of data have become not only available but also easily accessible, to appropriate (groups of) individuals within an organisation.

The processes themselves, however, have been given less attention. Different (coordinated) processes within an organisation can often be modelled as autonomous distributed processes with known goals. Goal-driven processes can be seen as tasks within an organisation. Tasks can often be decomposed into subtasks and most often involve more than one individual. Coordination between individuals, also called agents, is essential, as is interaction and cooperation. To model business processes not only the requirements tasks impose upon individual agents within an organisation are

---

<sup>\*</sup> In: Cooperative Knowledge Processing: The Key Technology for Intelligent Organisations, S. Kirn and G. O'Hare (eds.), Springer Verlag, Computer Supported Cooperative Work Series, 1996, pp. 212-229

of importance, but also the requirements tasks impose on coordination and interaction between (groups of) individual agents.

For individual agents the specific expertise required to perform tasks for which the agent is responsible should be made explicit in terms of knowledge and reasoning capabilities. Expertise, required to guide interaction, cooperation and coordination between agents is of a slightly different nature. Individual agents should know of the implications of distributed task performance with respect to interaction, cooperation and coordination, and be capable of acting appropriately.

A modelling framework should support explicit specification of both types of expertise. Transparent models of distributed agents and of the relevant interaction are essential. The framework DESIRE (framework for DEsign and Specification of Interacting REasoning components; cf. (Langevelde, Philipsen & Treur 92; Brazier, Treur, Wijngaards & Willems 94)), supports specification of knowledge, interaction and coordination of complex tasks and reasoning capabilities. Within the framework complex processes are designed as interacting task-based hierarchically structured components: as compositional architectures. The interaction (and coordination) between components, between components and the external world, as well as between components and one or more users (cf. (Brazier & Treur 94)), is specified precisely. Components can be reasoning components (including a knowledge base), but may also be subsystems which are capable of performing tasks such as calculation, information retrieval, optimisation, et cetera. As the framework inherently supports interaction between components, interaction between agents (modelled as components) is a natural application of the framework, as shown in (Brazier, Dunin-Keplicz, Jennings, and Treur, 1995; Dunin-Keplicz and Treur, 1995).

The philosophy behind the DESIRE framework is that it should support knowledge engineers in focussing on the specification of the conceptual design of a system: on both the static and the dynamic aspects to be considered. Implementation generators exist to automatically generate prototype implementations from specifications. DESIRE is currently used by a number of companies and research institutes for the development of compositional systems for complex tasks. A number of these systems are currently operational.

Specifications in DESIRE and their semantics can be made formal (on the basis of temporal logic; see (Engelfriet & Treur 94; Gavrila & Treur 94; Treur 94)). The formal basis offers the possibility to develop dedicated verification and validation methods for the domain of multi-agent systems. In contrast to general purpose formal specification languages, such as Z and VDM, DESIRE is committed to well structured compositional architectures. Such architectures can be specified in DESIRE at a higher level of conceptualisation than specifications in Z or VDM.

This paper briefly describes the framework DESIRE and its application to a multi-agent organisation. One of the few operational real-world distributed artificial intelligence applications, modelling an electricity transportation management task (Jennings et al 95), is used to illustrate how business processes are modelled within this framework. This domain of application is described below in Section 2. In Section 3 a brief description of the DESIRE framework is presented. The results of modelling and specifying the multi-agent electricity transportation management task are

presented in Section 4. Finally, in Section 5 these results are discussed and further perspectives are presented.

## 2 The Application Domain

The multi-agent system used as an illustration 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 five of them.

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.

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. Next these candidates are checked upon feasibility and relevance. Finally from the approved actions a repair plan is prepared. The execution of this plan (guided by the *human operator*) is monitored cooperatively by CSI, which groups any alarm messages coming from the network, and BAI which checks that SRA's predictions about the various intermediate states of its recover plan are in fact reflected in the real network.

### 3 A Specification Framework for Multi-Agent Systems

The architectures upon which specifications for compositional multi-agent systems are based are the result of analysis of the tasks performed by and between agents. Task (de)compositions include specifications of interaction between subtasks at each level within a task (de)composition, making it possible to explicitly model tasks which entail interaction between agents. 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.

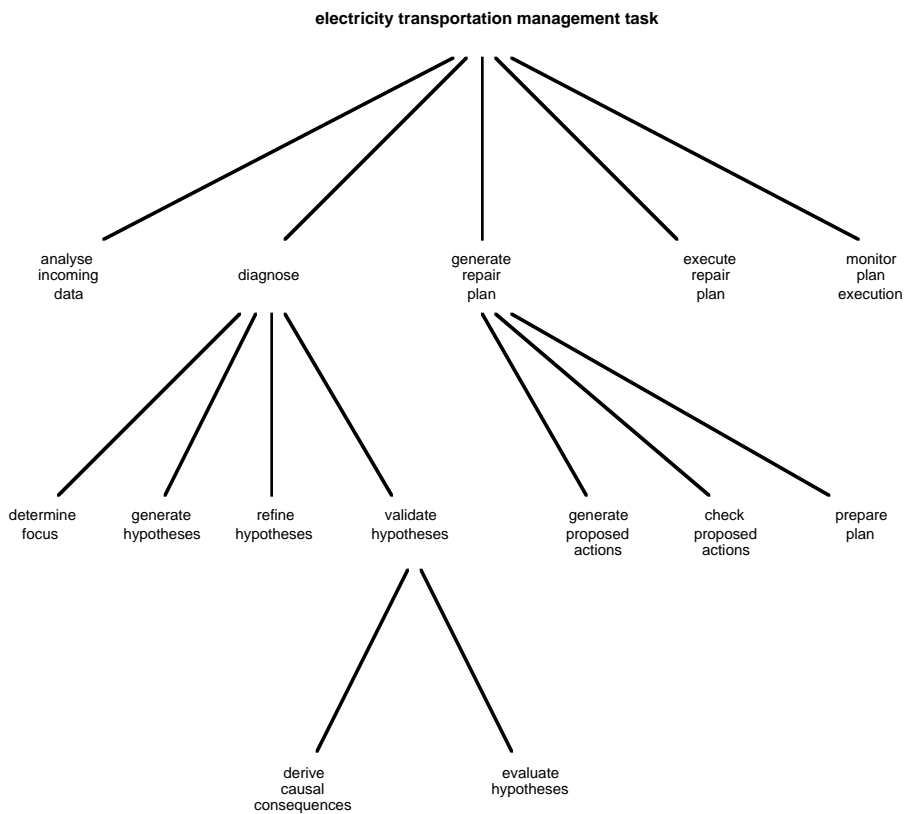
#### 3.1 Task (de)composition

To 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).

Within a task hierarchy *composed* and *primitive* tasks are distinguished: in contrast to primitive tasks composed tasks are tasks for which 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.

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



**Figure 1 Task hierarchy of electricity transportation management**

Information required/produced by a (sub)task is defined as *input* and *output signatures* 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 role information plays within reasoning is indicated by the level of an atom within a signature: different (meta)levels may be distinguished. In a two level situation the lowest level is termed *object-level information*, and the second level *meta-level information*. Meta-level information contains information about object-level information and reasoning processes; for example, for which atoms the values are still unknown (*epistemic information*). Similarly *tasks* which include reasoning about other tasks are modelled as meta-level tasks with respect to object-level tasks. Often more than two levels of information and reasoning occur, resulting in meta-meta-... 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 truth value of a specific output atom is linked with which truth value of a specific input atom. Atoms can be renamed: each component can be specified in its own language, independent of other components. The conditions for activation of information links are explicitly specified as task control information: 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 goals associated with task activation and the amount of effort which can be afforded to achieve a goal to a given extent. These aspects are specified as (sub)component and link activation together with sets of targets and requests, exhaustiveness and effort to define the component's goals. 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 component activation. Activation 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).

Task control is limited to global internal control and is independent of the content of the underlying components or knowledge.

### 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 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.

### 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.

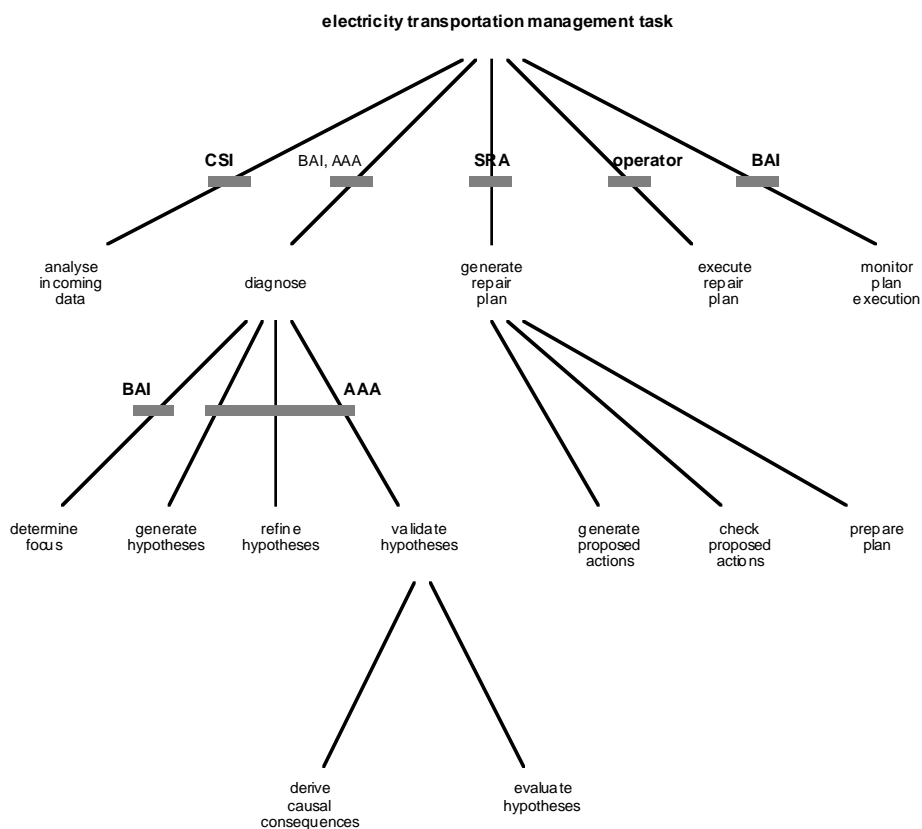


Figure 2 Delegation of tasks to agents

## 4 Formal Model and Specification of a Multi-Agent System

A *compositional agent* is a composed component with a number of subcomponents representing the agent's tasks. The types of knowledge distinguished above are described below for the compositional agents involved in the electricity transportation management task.

### 4.1 Task decomposition and role allocation

The task hierarchy presented in section 3.1 is a task hierarchy for the electricity transportation management task represented as a single composed task. In Section 3.4 subtasks are delegated to individual agents. 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 material world.

The example system described in Section 2 consists of five agents (CSI, AAA, BAI, SRA, operator) and their interactions. The main tasks of each of the agents in this example are similar; they each have the same three generic tasks (own process control, update world state information, preparing communication) and one agent-specific task to perform: e.g., diagnose fault (for the agent AAA), or identify blackout area (for the agent BAI). In Figure 3 agent AAA's first level decomposition is depicted. These generic tasks are generic in the sense that they can be (specialised and) instantiated for different agents (reuse).

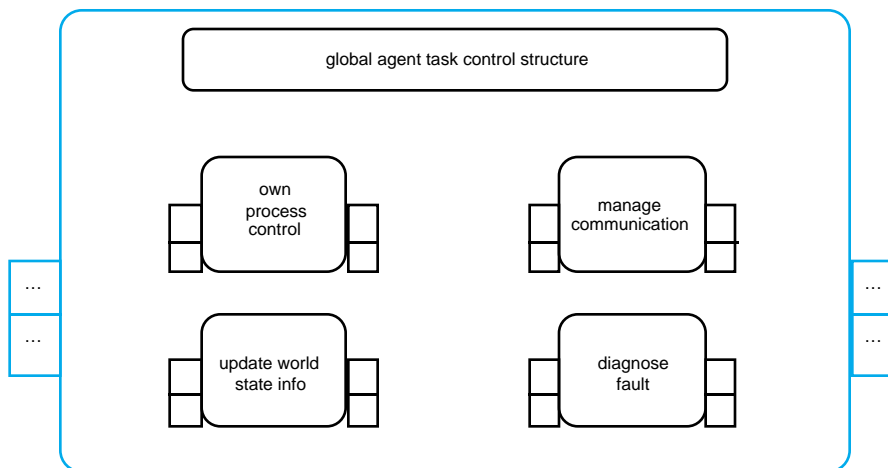


Figure 3 Top level compositional structure of agent AAA

In Figure 3 the small boxes on the left and right hand side denote the levelled input and output interface respectively, indicating object-meta distinctions. Agent AAA's complete task hierarchy as described in Section 2 is shown below in Figure 4:

1. Own process control
  - 1.1 Monitoring incoming data
  - 1.2 Evaluating the process state
2. Update world state information
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. Manage communication
  - 4.1 Examining the acquaintance model
  - 4.2 Generating requests

**Figure 4 Complete task hierarchy of agent AAA**

Task-subtask relations for AAA's top level (as shown in Figure 3) are specified as follows:

```

task structure AAA
  subcomponents own_process_control, update_world_state_info,
    diagnose_fault, manage_communication;
  .....

```

## 4.2 Information flow within an agent

*Information links* are defined to model *information exchange* between components *within an agent*. The information links of AAA's top level are depicted in Figure 5. Which information links are used within a component is specified as part of the task structure; for agent AAA, the specification of AAA's *task structure* is as follows:

```

task structure AAA
  subcomponents own_process_control, update_world_state_info,
    diagnose_fault, manage_communication;
  links incoming_snapshot_for_own_process_control,
    incoming_snapshot_for_world_info_update, grouped_alarms, blackout_area,
    current_snapshot, request_info, request_to_output, fault_results_to_output,
    request_out ;
end task structure AAA

```

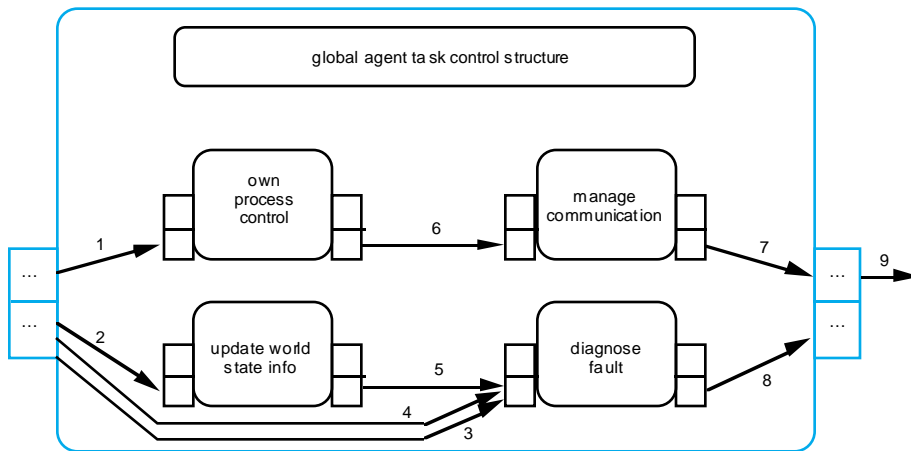
An example of an *information link specification* (see Sections 3.1.2 and 3.2.2) within AAA is link (5) between the component `update_world_state_information` and the component `diagnose_fault`:

```

private link current_snapshot: object-object
domain update_world_state_information
  output world_state_info
codomain diagnose_fault
  input world_state_info
sort links (World_state,World_state)
object links identity
term links identity
atom links (current_world_state_info(!:World_state),current_world_state_info(!:World_state)):
  <<true,true>,<false,false>>
endlink

```

This link relates output of the component `update_world_state_information` to input of the component `diagnose_fault`. The truth value (`true`, resp. `false`) of the atom `current_world_state_info(!:World_state)` is transferred from `update_world_state_information` to `diagnose_fault`.



**Figure 5 Information flow of agent AAA's top level;**

**the following links are depicted:**

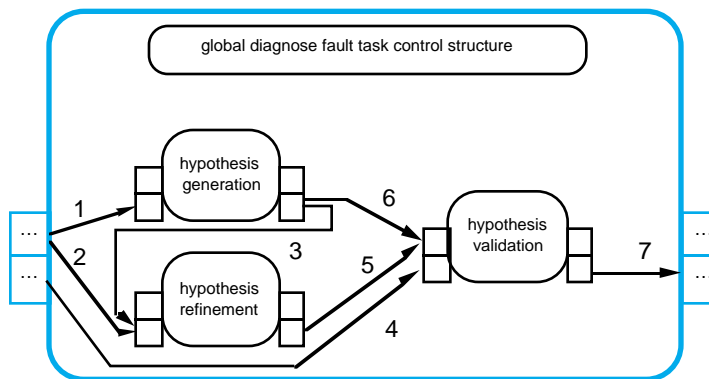
incoming\_snapshot\_for\_own\_process\_control (1), incoming\_snapshot\_for\_world\_info\_update (2), grouped\_alarms (3), blackout\_area (4), current\_snapshot (5), request\_info (6),

request\_to\_output (7), fault\_results\_to\_output (8), request\_out (9)

One level down in AAA's hierarchy the decomposition and information flow within agent AAA's component `diagnose_fault` is specified as follows:

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

This information is depicted in Figure 6.



**Figure 6 The component diagnose fault of agent AAA;**

**the following links are depicted:**

import\_disturbances (1), import\_blackout\_info (2), poss\_hyps\_to\_refine (3),  
import\_snapshot\_info (4), lim\_hyps\_to\_validate (5), poss\_hyps\_to\_validate (6),  
diagnosis\_to\_output (7)

An example of an information link specification within the component `diagnose_fault` is link (6) between the components `hypothesis_generation` and `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`, by which 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)`. Note that this link actually renames atoms between components.

The components `hypothesis_generation` and `hypothesis_refinement` are meta-level reasoning components (with respect to the object level reasoning about the world). They reason about the reasoning process. Which hypotheses to consider, for example, requires information about which hypotheses have already been examined: whether they have been confirmed, rejected, or are yet unknown, in the current information state of the reasoning process (epistemic information). As this is information about the state of the process itself and not about the world, the meta-level components `hypothesis_generation` and `hypothesis_refinement` use meta-information from outside the component `diagnose_fault` as their input: their input links start one (meta-)level higher in the input interface (see links (1), (2) and (4) in Figure 6).

In the specification the information links have names that can be used in the task control knowledge to specify under which conditions to transfer the up-to-date information.

### 4.3 Task control within an agent

Task control knowledge (for complex and primitive tasks alike), specifies 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; Gavrilu & Treur 94; 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 explicitly. 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.

Below AAA's task control knowledge (Section 4.3.1) and its subcomponent `diagnose_fault` (Section 4.3.2) are discussed.

#### 4.3.1 Agent AAA's task control

Control over AAA's four subtasks is limited: the seven 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`. From the start both AAA's component `own_process_control` and the information link `incoming_snapshot_for_own_process_control` have to become and remain awake. This is expressed by:

```
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)
```

A typical example of a component's task control knowledge rule in which the success of one component is required (whether or not incoming snapshot data have been monitored by the component `own_process_control`) before a following component (`update_world_state_info`) can be activated with the required information, is the following:

```
if evaluation(own_process_control, ts_update, succeeded)
then next-component-state(update_world_state_info, active)
and next-target-set(own_process_control, ts_oss_hyps)
and next-link-state(incoming_snapshot_for_world_info_update, up_to_date)
```

This knowledge rule states that

```
if the component own_process_control has succeeded in accomplishing
the targets defined by its target set ts_update,

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_world_state_info, given
information which has been recently updated by activation of the
link incoming_snapshot_for_world_info_update.
```

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

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 `manage_communication` is activated if the component `own_process_control` determines that a request for blackout area information is needed (because this information is still lacking):

```

if    evaluation(own_process_control, ts_requests, succeeded)
then  next-component-state(manage_communication, active)
and   next-target-set(manage_communication, outgoing_requests)
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, 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(manage_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

Task control knowledge in a lower level component is expressed in precisely the same way as task control knowledge of a higher level component. The control of the three subcomponents of the component `diagnose_fault` (`hypothesis_generation`, `hypothesis_refinement` and `hypothesis_validation`) begins after 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_oss_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_oss_hyps` and up to date information about disturbances.

If blackout information is available and hypotheses have been generated successfully, `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_oss_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(import_blackout_info, up_to_date)
and   next-link-state(oss_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_oss_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(oss_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_oss_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(lim_hyps_to_validate, up_to_date)

```

#### 4.4 Control and communication between agents

In addition to task control within agents, limited *global task control* is required to initially awaken all agents involved in task performance. Once agents are active, their agent task control knowledge determines the sequencing of task execution. Therefore, task 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)

```

*Communication between agents* is modelled by activation of information links. Specific types of interaction can be modelled explicitly. 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 interaction 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.

An example of agent communication between AAA and BAI is the request AAA issues to BAI for blackout area information. The 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

Industrial and business processes most often entail interaction between multiple autonomous agents. Modelling such processes is essential from the perspective of both knowledge management and knowledge engineering. To be able to provide support to organisations in which distributed processes are inherent, distributed models of parts and their interaction are essential. Declarative distributed models, models in which strategic, procedural and factual knowledge of the participating agents and their interaction are explicitly represented, provide a basis for such support.

The declarative compositional framework DESIRE (Langevelde, Philipsen & Treur 92; Brazier, Treur, Wijngaards & Willems 94) supports the specification of multiple agents and of interaction and coordination between agents, as presented in this paper. The cooperative interaction between agents in the example domain of electricity transportation management, used to illustrate our approach, was well-defined. The framework provided the expressiveness required to model the agents and their interaction. Knowledge of other agents, coordination of parallel processes and event-driven processing, for example, are aspects characteristic to multi-agent systems, which have been explicitly modelled for this example. Other aspects, such as different states of awareness, willingness to communicate, and different (sets of) goals (for a more extensive list of agent characteristics, see, e.g., (Dieng, Corby & Labidi 94)) were not relevant to the example at hand. Current joint research focusses on a number of such aspects.

To verify and validate DESIRE models (cf. Treur & Willems 95), not only have formal semantics been devised on the basis of temporal logic (cf. Brazier, Treur, Wijngaards & Willems 94), but also automated implementation generators have been constructed. The implication of both synchronous and asynchronous activation of agents is currently another topic of research.

## Acknowledgements

This research was partly supported by the ESPRIT III Basic Research project 6156 DRUMS II on Defeasible Reasoning and Uncertainty Management Systems.

## References

- F.M.T. Brazier, B. 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*, MIT Press, 1995, pp. 25-32.
- 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: J.C. Bioch, Y.H. Tan (eds.), *Proc. Seventh Dutch AI Conference, NAIC'95*, 1995, pp. 307-316. 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-Classical 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. 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.
- 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*. Lecture Notes in Computer Science, Vol. 883, Springer Verlag, pp. 353-376.
- 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, pp. 189-199
- Treur, J. and Th. Wetter (eds.) (1993). *Formal Specification of Complex Reasoning Systems*, Ellis Horwood, pp. 282.