

# Dynamics and Control in Component-Based Agent Models

Frances M. T. Brazier, Catholijn M. Jonker,<sup>†</sup> Jan Treur<sup>‡</sup>  
*Department of Artificial Intelligence, Vrije Universiteit Amsterdam,  
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands*

Dynamics are an important aspect of agent models. Control of dynamics requires specific methods of specification that have their own specific semantics. This paper addresses specification and semantics of dynamics and control in component-based agent models. Specification is based on a dedicated formal design specification language for agent models. Semantics of the dynamics are defined using temporal traces with composite states. It is shown in what manner control aspects can be specified, and what their semantics is in terms of the temporal traces. An agent model for controlled diagnostic reasoning processes is used to illustrate the approach. © 2002 Wiley Periodicals, Inc.

## 1. INTRODUCTION

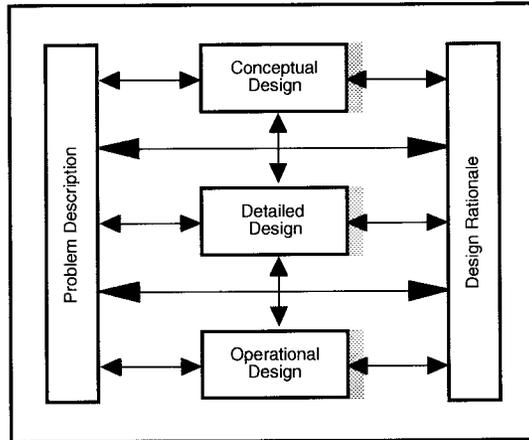
The component-based multi-agent design method DESIRE (DEsign and Specification of Interacting REasoning components) supports the design of component-based autonomous interactive agents (cf. Brazier, Jonker, and Treur<sup>1</sup>). DESIRE views both the individual agents and the overall system as component-based structures—hence all functionality is designed in terms of interacting, compositionally structured components. Complex distributed processes are the result of tasks performed by agents in interaction with their environment.

In this paper it is discussed how dynamics and the control of dynamics of processes within an agent are modeled using this component-based approach. Examples of typical dynamic reasoning patterns of intelligent agents are discussed: one reasoning pattern in which assumptions are dynamically introduced, evaluated and, if needed, retracted (temporal epistemic reflection), and another reasoning pattern in which the reasoning process is focused by dynamic generation of goals for the reasoning. First, in the remainder of this section a brief overview of the component-based modeling approach is presented. In Section 2 the notion of

e-mail: frances@cs.vu.nl.

<sup>†</sup> e-mail: jonker@cs.vu.nl.

<sup>\*</sup> Author to whom all correspondence should be addressed: e-mail: treur@cs.vu.nl.



**Figure 1.** Problem description, levels of design, and design rationale.

problem description is briefly discussed, focusing on the reasoning patterns addressed in this paper. In Section 3 details of the model of one of the reasoning patterns discussed in this paper are presented, thereby introducing the DESIRE modeling concepts needed. Section 4 is a brief discussion about the notion of design rationale. In Sections 5 and 6 the dynamics and control of the two example reasoning patterns addressed in the paper are discussed more extensively. Section 7 concludes the paper.

### 1.1. The Design Process

The design of a multiagent system is an iterative process, which aims at the identification of the parties involved (i.e., human agents, system agents, external worlds), and the processes, in addition to the types of knowledge needed. Conceptual descriptions of specific processes and knowledge are often first attained. Further explication of these conceptual design descriptions results in detailed design descriptions, most often in iteration with conceptual design. During the design of these models, partial prototype implementations may be used to analyze or verify the resulting behavior. On the basis of examination of these partial prototypes, new designs and prototypes are generated and examined, and so on and so forth. This approach to *evolutionary development* of systems is characteristic to the development of multiagent systems in DESIRE.

During a multiagent system development process, DESIRE distinguishes the following descriptions (see Fig. 1):

- problem description
- conceptual design
- detailed design
- operational design
- design rationale

The *problem description* includes the *requirements* imposed on the design. The *rationale* specifies the choices made during design at each of the levels, and assumptions with respect to its use. The relationship between the levels of design (conceptual, detailed, operational) is well defined and structure-preserving. The *conceptual design* includes conceptual models for each individual agent, the external world and the interaction between agents, and between agents and the external world. The *detailed design* of a system, based on the conceptual design, specifies all aspects of a system's knowledge and behavior. A detailed design is an adequate basis for *operational design*. Prototype implementations are automatically generated from the detailed design.

There is no fixed sequence of design: depending on the specific situation, different types of knowledge are available at different points during system design. The end result, the final multiagent system design, is specified by the system designer at the level of detailed design. In addition, important assumptions and design decisions are specified in the design rationale. Alternative design options together with argumentation are included. On the basis of verification during the design process, properties of models can be documented with the related assumptions. The assumptions define the limiting conditions under which the model will exhibit specific behaviour.

## 1.2. Compositionality of Processes and Knowledge

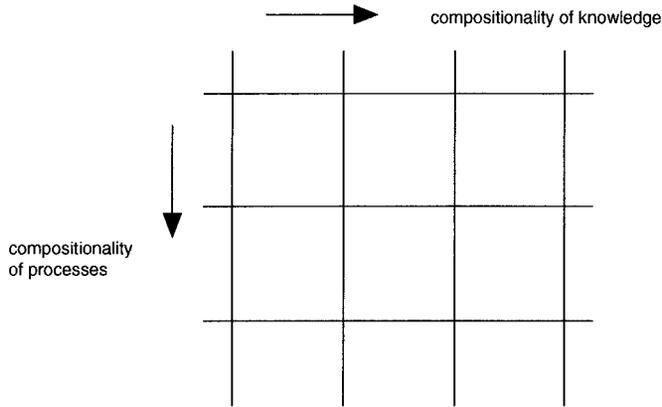
Compositionality is a general principle that refers to the use of components to structure a design. Within the DESIRE method components are often complex compositional structures in which a number of other, more specific components are grouped. During design different levels of process abstraction are identified. Processes at each of these levels (except the lowest level) are modeled as (process) *components* composed of components at the adjacent lower level.

Processes within a multiagent system may be viewed as the result of interaction between more specific processes. A complete multiagent system may, for example, be seen as one single component responsible for the performance of the overall process. Within this one single component a number of agent components and an external world can be distinguished, each responsible for a more specific process. Each agent component may, in turn, have a number of internal components responsible for more specific parts of this process. These components may themselves be composed, again entailing interaction between other more specific processes.

The *ontology* used to express the knowledge needed to reason about a specific domain may also be seen as a single (knowledge) component. This *knowledge structure* can be composed of a number of more specific knowledge structures, which, in turn, may again be composed of other even more specific knowledge structures.

As shown in Fig. 2, *compositionality of processes* and *compositionality of knowledge* are two separate, orthogonal dimensions. The compositional knowledge structures are referenced by compositional process structures, when needed.

Compositionality is a means to acquire *information and process hiding* within



**Figure 2.** Compositionality of processes and compositionality of knowledge.

a model: by defining processes and knowledge at different levels of abstraction, unnecessary detail can be hidden. Compositionality also makes it possible to *integrate* different types of components in one agent. Components and groups of components can be easily included in new designs, supporting *reuse* of components at all levels of design.

## 2. PROBLEM DESCRIPTION

Which techniques used to acquire a *problem description* is not predefined. Techniques vary in their applicability, depending on, for example, the situation, the task, and the type of knowledge on which the system developer wishes to focus. Acquisition of requirements to be imposed on the system as part of the problem description is crucial. These requirements are part of the initial problem definition, but may also evolve during the development of a system. To illustrate the dynamics and control in agent models in this paper, any of the examples used addresses an information gathering and analysis task: diagnosis of refrigerator problems.

To make decisions, agents often have to analyze a given situation by gathering information and drawing conclusions from the gathered information. Gathering (the right type of) information requires effort. To use an agent's resources economically, an information gathering process should concentrate on gathering only information that is relevant for the conclusions the agent wants to be able to draw. Ideally, an agent can focus an information gathering process by strategic reasoning. Such a reasoning process is, for example, necessary in situations in which unexpected behavior occurs in the environment, and the cause of this deviant behavior has to be analyzed (this process is sometimes called *diagnosis*). In this section an example of a strategic reasoning process to guide information gathering is discussed. The conclusions in which the agent is interested are called *hypotheses*. Information gathering is assumed to take place by *observation* (for information gathering based on, for example, communication, the pattern is similar).

Viewed from a global perspective, the agent performs a coherent and well-structured pattern of reasoning that subsequently (and iteratively) involves determining one or more hypotheses on which to focus, and confirming or rejecting these hypotheses on the basis of observations:

- determine the hypotheses on which to focus (*focus hypotheses*)
- validate these focus hypotheses:
  - determine relevant observations on which to focus (*focus observations*)
  - perform these focus observations
  - evaluate the focus hypotheses on the basis of the observation results
  - repeat this (hypothesis validation) process
  - repeat the whole process

The example diagnostic process used in this chapter is a simplified case of diagnosis of refrigerator malfunctioning. Four types of faults are considered:

- the light *bulb* is *broken*
- there is *no power supply*
- the *pump* is *broken*
- the *cooling system* is *broken*

Only three observations can be performed:

- if the door is opened there is (no) *light*
- (no) *noise* of the pump is heard
- it is (not) *cold* in the fridge

Assumptions are:

- only situations in which one type of fault occurs are considered (*single fault assumption*)

Requirements are:

- for each situation a correct diagnosis shall be determined
- the diagnostic process shall be parsimonious: The most efficient effort of observation shall be performed during the diagnostic process to reach a conclusion

Requirements can be specified, e.g., using the semiformal and formal languages discussed in Herlea, Jonker, Treur, and Wijngaards.<sup>2</sup>

### 3. CONCEPTUAL DESIGN AND DETAILED DESIGN

A conceptual and detailed design consist of specifications of the following three types:

- process composition
- knowledge composition
- the relation between process composition and knowledge composition

<i>process</i>	<i>input information type</i>	<i>output information type</i>
Diagnostic Reasoning	Observation Result Info	Assessed Hypotheses Selected Observations
External World	Required Observations	Observation Result Info

**Figure 3.** Interface information types within the Top Level.

These three types of specifications are discussed in more detail in the text that follows.

### 3.1. Process Composition

Process composition identifies the relevant processes at different levels of (process) abstraction, and describes how a process can be defined in terms of lower level processes.

#### 3.1.1. Identification of Processes at Different Levels of Abstraction

Different views can be taken: a task perspective and a multiagent perspective. The *task perspective* refers to the view in which the processes needed to perform an overall task are distinguished. These processes (or subtasks) are then *delegated* to appropriate agents and the external world. The *multiagent perspective* refers to the view in which agents and one or more external worlds are first distinguished and then the processes within them, including agent-related processes such as management of communication, or control of an agent's own processes.

*Specification of a Process.* The identified processes are modelled as *components*. For each process, the *types of information* used as input and resulting as output are identified as well, and modeled as *input and output interfaces* of the component.

To model the example refrigerator diagnosis process, a generic task model for diagnosis can be used, consisting at the top level of a reasoning process and an external world in which observation results are obtained. The reasoning process is composed of the processes hypothesis determination and hypothesis validation, where the hypothesis validation process is composed of the processes observation determination and hypothesis evaluation. Based on this generic model, the following processes are identified for the example diagnostic process: external world, and diagnostic reasoning, with subprocesses hypothesis determination and hypothesis validation. The process hypothesis validation has two subprocesses: observation determination and hypothesis evaluation. The types of information used and produced by each of these tasks are shown in Figs. 1, 2, and 3.

As shown in Fig. 3, three information types are distinguished for the process as a whole. The diagnostic reasoning process generates information on specific (focus) observations that are selected (Selected Observations), and assessments of hypotheses (Assessed Hypotheses). It requires observation results (Observation

<i>process</i>	<i>input information type</i>	<i>output information type</i>
Hypothesis Determination	Assessed Hypotheses Selected Hypotheses	Selected Hypotheses
Hypothesis Validation	Observation Result Info Focussed Hypotheses	Assessed Hypotheses Selected Observations

**Figure 4.** Interface information types within Diagnostic Reasoning.

Result Info) as input. Acquisition of information in the external world is focused on the basis of information on the observations to be performed (Required Observations) and provides the results (Observation Result Info).

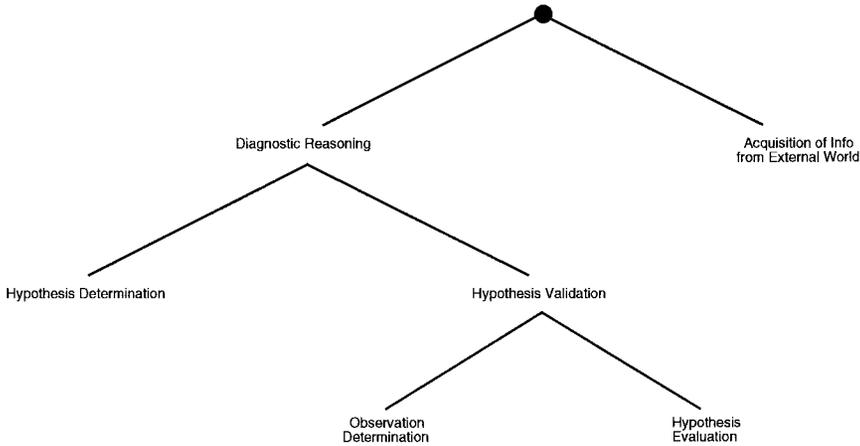
Figure 4 depicts the input and output information types of the processes within the diagnostic reasoning process. The process of determining on which hypotheses to focus (Hypothesis Determination) uses information on which hypotheses have already been assessed (Assessed Hypotheses), which hypotheses have already been in focus (Selected Hypotheses), and results of observations that have been performed. The result is a list of one or more hypotheses which may be used to focus the diagnostic process (Selected Hypotheses).

The process of validating one or more hypotheses, Hypothesis Validation, uses information on the hypotheses on which to focus (Focussed Hypotheses) and results of observations (Observation Result Info). During validation, often a need for specific information is identified (Selected Observations). Once the validation process has been completed the results—hypotheses that have been assessed (Assessed Hypotheses)—are available as output.

Figure 5 depicts the information types used and produced by the processes needed to validate hypotheses. To determine which observations to perform (Observation Determination), information is needed on the hypotheses on which to focus (Focussed Hypotheses) and the available information on observations (Observation Information). The results of this process are a list of one or more observations to be performed (Selected Observations). The process Hypothesis Evaluation, involves evaluating one or more hypotheses (Target Domain Hypotheses) on the basis of information on observations performed (Assumption Domain Info). The result is an evaluation of the hypotheses (Epistemic Domain Hypotheses) on which evaluation focussed.

<i>process</i>	<i>input information type</i>	<i>output information type</i>
Observation Determination	Focussed Hypotheses Observation Information	Selected Observations
Hypothesis Evaluation	Target Domain Hypotheses Assumption Domain Info Domain Info	Epistemic Domain Hypotheses Epistemic Domain Info Domain Info Domain Hypotheses

**Figure 5.** Interface information types within Hypothesis Validation.



**Figure 6.** Process abstraction levels in a strategic information gathering process.

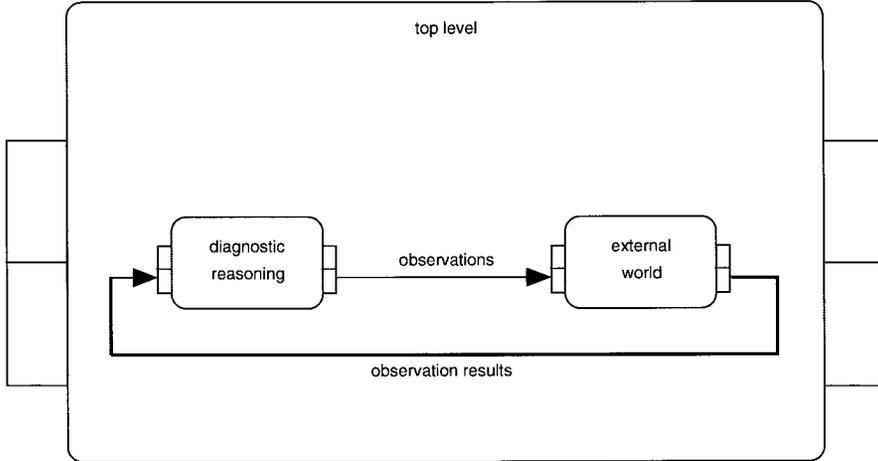
*Specification of Abstraction Levels.* The identified levels of process abstraction are modeled as *abstraction/specialization relations* between components at adjacent levels of abstraction: Components may be *composed* of other components or they may be *primitive*. Primitive components may be either reasoning components (for example, based on a knowledge base), or, alternatively, components capable of performing tasks such as calculation, information retrieval, optimization, etc.

For the example diagnostic reasoning process the process abstraction levels are depicted in Fig. 6. The identification of processes at different abstraction levels results in specification of components that can be used as building blocks, and of a specification of the subcomponent relation, defining which components are a subcomponent of a which other component. The distinction of different process abstraction levels results in process hiding.

### 3.1.2. Composition

The way in which processes at one level of abstraction are composed of processes at the adjacent lower abstraction level is called *composition*. This composition of processes is described by the possibilities for *information exchange* between processes (*static view* on the composition), and *task control knowledge* used to control processes and information exchange (*dynamic view* on the composition).

*Information Exchange.* A specification of information exchange defines which types of information can be transferred between components and the *information links* by which this can be achieved. Within each of the components *private* information links are defined to transfer information from one component to another. In addition, *mediating* links are defined to transfer information from the



**Figure 7.** Information exchange within the top level process.

input interfaces of encompassing components to the input interfaces of the internal components, and to transfer information from the output interfaces of the internal components to the output interface of the encompassing components.

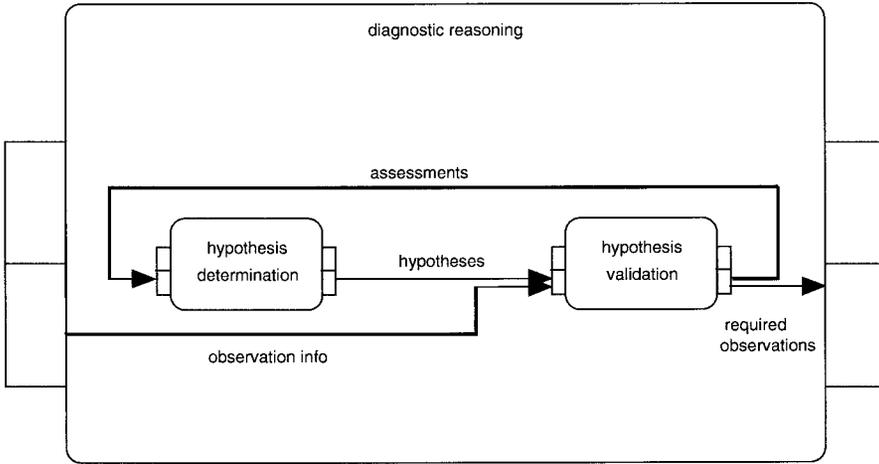
As shown in Fig. 7, two private links are defined at the top level: observations and observation results. The link observations transfers the list of observations to be performed (Selected Observations) from the lower level (called D object level) of the output interface of the component diagnostic reasoning to the lower level (called EW object level) of the input interface of the component external world. The link observation results transfers results of observations (Observation Result Info) from EW object level of the output interface of the component external world to D object level of the input interface of the component diagnostic reasoning.

In Fig. 8, four information links defined for the component diagnostic reasoning are depicted: two private and two mediating. The private links:

- The link hypotheses transfers Selected Hypotheses from the lower level (called HD object level) of the component hypothesis determination to the lower level (called HV object level) of the component hypothesis validation to become the hypotheses on which the diagnostic process is to focus.
- The link assessments transfers Assessed Hypotheses from level HV object level of the output interface of the component hypothesis validation to HD object level of the input interface of the component hypothesis determination.

The two mediating links:

- The link observation info transfers observation results from D object level of the input interface of the component diagnostic reasoning to HV object level of the component hypothesis validation.
- The link required observations transfers selected observations from HV object level of the

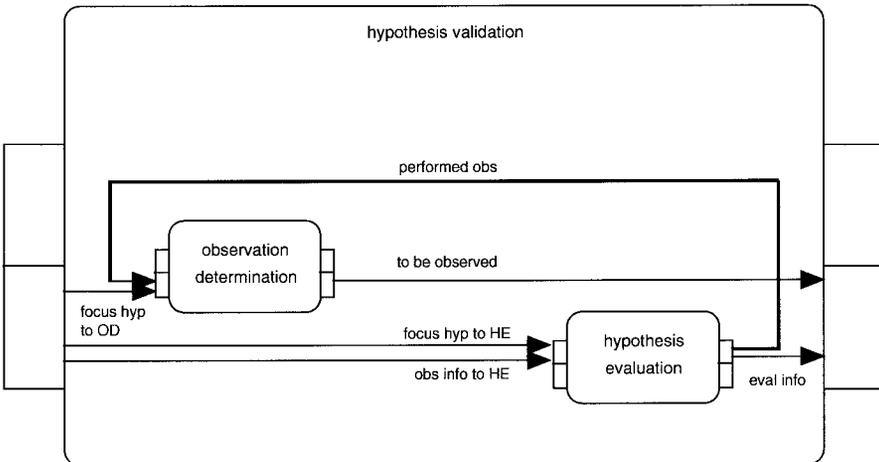


**Figure 8.** Information exchange within the diagnostic reasoning process.

component hypothesis validation to D object level of the output interface of the component diagnostic reasoning.

Within the component hypothesis validation one private link and five mediating links are defined (see Fig. 9). The private link:

- The link performed obs transfers epistemic domain information (on the results of observations) from HE meta-level (the highest level of the output interface of hypothesis evaluation) of the output interface of the component hypothesis evaluation to HD object level of the input interface of observation determination.



**Figure 9.** Information exchange within Hypothesis Validation.

The five mediating links:

- The link focus hyp to OD transfers the hypotheses to be validated, Focused Hypotheses, from HV object level of the input interface of the component hypothesis validation to OD object level of the component observation determination.
- The link focus hyp to HE transfers the hypotheses to be validated, Focused Hypotheses, from HV object level of the input interface of the component hypothesis validation to HE meta-level of the component hypothesis evaluation.
- The link obs info to HE transfers the results of observations, Observation Result Info, from HV object level of the input interface of the component hypothesis validation to HE meta-level of the input interface of the component hypothesis evaluation.
- The link to be observed transfers Selected Observations from OD object level of the output interface of the component observation determination to HV object level of the output interface of the component hypothesis validation.
- The link eval info transfers the result of hypothesis evaluation, namely, Assessed Hypotheses from HE meta-level of the component hypothesis evaluation to HV object level of the output interface of the component hypothesis validation.

*Task Control Knowledge.* Components may be activated sequentially or they may be continually capable of processing new input as soon as it arrives (*awake*). The same holds for information links: Information links may be explicitly activated or they may be awake. *Task control knowledge* specifies under which conditions which components and information links are active (or made awake). Evaluation criteria, expressed in terms of the evaluation of the results (success or failure), provide a means to guide further processing.

In a design, task control knowledge specifies when and how processes are to be performed and evaluated. Goals of a process are defined by the *task control foci* together with the *extent* to which they are to be pursued. Evaluation of the success or failure of a process's performance is specified by *evaluation criteria* together with an extent. Processes may be performed in sequence or in parallel, some may be continually performed (e.g., reacting to new input as soon as it arrives), some are to be explicitly activated.

The two main processes distinguished for the example, the diagnostic reasoning process and the external world, are both designed to react to new input as soon as it arrives. Both processes are performed in parallel. Diagnostic reasoning, however, as modeled in this example, entails determination of hypotheses and validation. These processes are performed in sequence: Once a set of hypotheses has been selected as a focus for hypothesis validation, hypothesis validation analyzes the set. The result of validation is new input for hypothesis determination. The more precise specification of task control knowledge structures is addressed in Section 3.2.

### 3.2. Knowledge Composition

Knowledge composition identifies the knowledge structures at different levels of (knowledge) abstraction, and describes how a knowledge structure can be defined in terms of lower level knowledge structures. The knowledge abstraction levels may correspond to the process abstraction levels, but this is not often the

case; often the matrix depicted in Fig. 2 shows more than a one to one correspondence between process abstraction levels and knowledge abstraction levels.

### 3.2.1. Identification of Knowledge Structures at Different Abstraction Levels

The two main structures used as building blocks to model knowledge are: *information types* and *knowledge bases*. Knowledge structures can be identified and described at different levels of abstraction. At the higher levels the details can be hidden. The resulting levels of knowledge abstraction can be distinguished for both information types and knowledge bases.

*Information Types.* An information type defines an *ontology* (lexicon, vocabulary) to describe objects or terms, their sorts, and the relations or functions that can be defined on these objects. Information types are defined as signatures (sets of names for sorts, objects, functions, and relations) for order-sorted predicate logic. Information types can be specified in graphical form (see Jonker, Kremer, et al.<sup>3</sup>), or in formal textual form. For the example diagnostic reasoning task, the following information types have been specified in textual form. The attribute information types is used to import other information types. Via the attribute meta-descriptions, all atoms that can be built by the referenced information type are imported in the indicated sort. For example,

```
meta-descriptions
    domain_hypotheses      :      HYPOTHESIS;
```

specifies that all atoms that can be formed using the information type `domain_hypotheses` are included as objects or terms in the sort `HYPOTHESIS`. This language construct is used to define object-meta relations in the reasoning process.

*Generic information types on hypotheses:*

```
information type hypotheses_sorts
    sorts      HYPOTHESIS;
end information type
```

```
information type meta_domain_hypotheses
    information types  hypotheses_sorts;
    meta-descriptions
        domain_hypotheses :      HYPOTHESIS;
end information type
```

```
information type assessed_hypotheses
    information types  meta_domain_hypotheses;
    relations  tried, rejected, confirmed :      HYPOTHESIS;
end information type
```

```

information type selected_hypotheses
  information types meta_domain_hypotheses;
  relations
    to_be_validated,
    has_been_focus : HYPOTHESIS;
    subhypothesis: HYPOTHESIS*HYPOTHESIS;
end information type

```

*Generic information types on symptoms:*

```

information type symptoms_sorts
  sorts SYMPTOM;
end information type

```

```

information type meta_domain_symptoms
  information types symptoms_sorts;
  meta-descriptions
    domain_symptoms : SYMPTOM;
end information type

```

```

information type selected_observations
  information types meta_domain_symptoms; meta_domain_hypotheses
  relations to_be_observed : SYMPTOM;
    relevant_observation_for : SYMPTOM*HYPOTHESIS;
end information type

```

```

information type observation_results
  information types domain_symptoms;
end information type

```

```

information type value_signs
  sorts SIGN
  objects pos, neg : SIGN;
end information type

```

```

information type observation_information
  information types meta_domain_symptoms, value_signs;
  relations observed : SYMPTOM*SIGN;
end information type

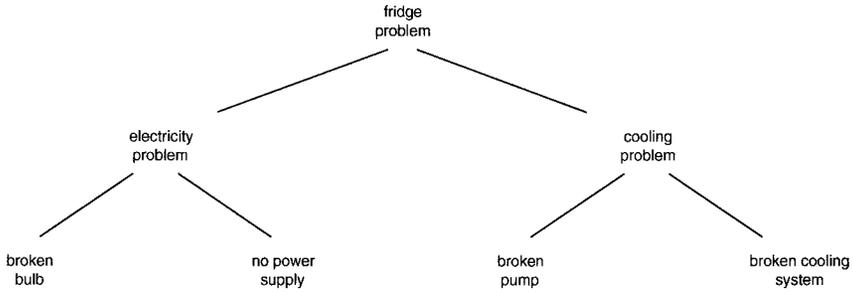
```

*Domain-specific information types on hypotheses:*

```

information type domain_hypotheses
  information types hypotheses_sort;
  objects fridge_problem, electricity_problem, cooling_problem, no_power_supply,
  broken_bulb, broken_pump, broken_cooling_system:
  HYPOTHESES;
end information type

```

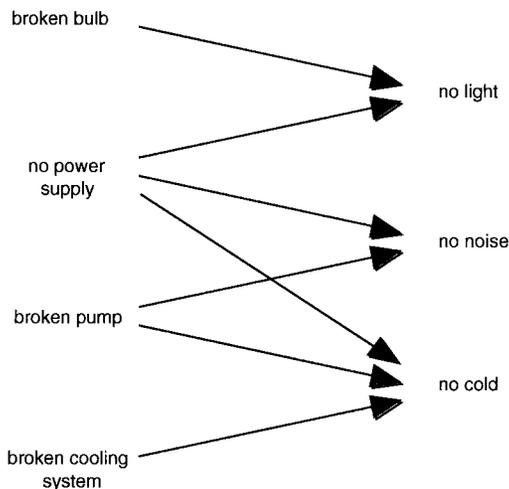


**Figure 10.** Taxonomy of types of fridge problems.

*Domain-specific information types on symptoms:*

**information type** domain\_symptoms  
**information types** symptoms\_sort;  
**objects** light, cold, noise: SYMPTOM;  
**end information type**

*Knowledge Bases.* Knowledge bases use ontologies defined in information types. Which information types are used in a knowledge base is defined by a relation between information types and knowledge bases. In detailed design, knowledge bases are specified in order-sorted predicate logic form, normalized to a classical rule format (implications between conjunctions of literals), as shown in Fig. 10. Knowledge bases can also be specified in conceptual pre-formal manners, for example in graphical forms. Graphical representations of knowledge bases for the example diagnostic reasoning task are as follows. In Fig. 11 the causal relations between faults and observations are depicted.



**Figure 11.** The causal relations between faults and observations.

As mentioned in the problem description, for simplicity's sake, only single faults are considered. If any one of the faults occurs, there is a *fridge problem*. If the fridge either has a broken bulb or there is no power supply, the fridge is said to have an *electricity problem*. If either the pump or the cooling system is broken, the fridge is said to have a *cooling problem*. In Fig. 11 this taxonomy of types of problems is depicted.

The object level domain knowledge can be modeled in a *causal* (i.e., faults imply observations, as depicted in Fig. 10) or *anticausal* (i.e., observations imply faults) manner. For this example, the anticausal manner has been chosen. The knowledge base anticausal fridge knowledge consists of:

```

if fridge_problem          and not light          then electricity_problem
if fridge_problem          and not cold           then cooling_problem
                             and light              then no_power_supply
if electricity_problem     and not cold           then broken_bulb
if electricity_problem     and cold             then broken_pump
if cooling_problem         and not noise          then broken_cooling_system
if cooling_problem         and noise              then broken_cooling_system

if light                   then not electricity_problem
if cold                    then not cooling_problem
if not cold                and not light          then not cooling_problem

if not fridge_problem     then not electricity_problem
                             and not cooling_problem

if not
  electricity_problem      then not broken_bulb
                             and not no_power_supply
if not cooling_problem     then not broken_pump
                             and not broken_cooling_
                             system
  
```

The component hypothesis determination uses the following knowledge bases:

hypothesis\_refinement\_kb (*domain-independent knowledge*)

```

if confirmed(H : HYPOTHESIS)
  and subhypothesis_of(H1:HYPOTHESIS, H : HYPOTHESIS)
  and not tried(H1:HYPOTHESIS)
  then focus_hypothesis(H1:HYPOTHESIS)
  
```

subhypotheses\_kb (*domain-specific knowledge*)

```

subhypothesis_of(broken_bulb, electricity_problem)
subhypothesis_of(no_power_supply, electricity_problem)
subhypothesis_of(broken_pump, cooling_problem)
subhypothesis_of(broken_cooling_system, cooling_problem)
subhypothesis_of(cooling_problem, fridge_problem)
subhypothesis_of(electricity_problem, fridge_problem)
  
```

The domain specific knowledge base `subhypotheses_kb` represents the knowledge depicted in a graphical form in Fig. 11 (the taxonomy). The generic rule in the domain independent knowledge base `hypothesis_refinement_kb` specifies that each of the confirmed children of a node become a focus hypothesis, if not already validated.

The component observation determination uses the following knowledge bases:

```
generic_obs_determination_kb (domain-independent knowledge)
  if focus_hypothesis(H : HYPOTHESIS)
    and relevant_observation_for(S:SYMPTOM, H : HYPOTHESIS)
  then relevant_observation(S:SYMPTOM)
  if relevant_observation(S:SYMPTOM)
    and not observed(S:SYMPTOM, pos)
    and not observed(S:SYMPTOM, neg)
  then to_be_observed(S:SYMPTOM)
```

```
obs_relevance_kb (domain-specific knowledge)
  relevant_observation_for(light, electricity_problem)
  relevant_observation_for(cold, cooling_problem)
  relevant_observation_for(cold, broken_lamp)
  relevant_observation_for(cold, no_power_supply)
  relevant_observation_for(noise, broken_pump)
  relevant_observation_for(light, broken_pump)
  relevant_observation_for(noise, broken_cooling_system)
```

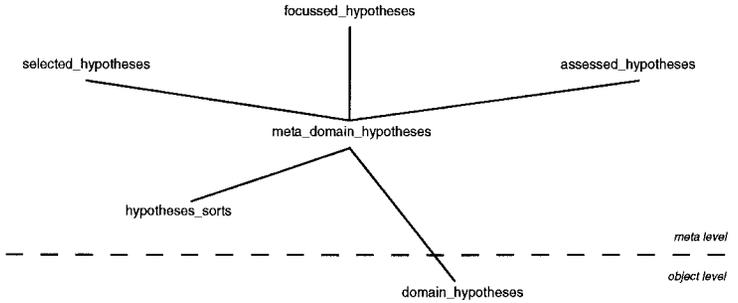
These knowledge bases specify that the observations that are relevant for at least one of the focus hypotheses are selected, unless they have already been performed.

### 3.2.2. Composition of Knowledge Structures

Information types can be composed of more specific information types, following the principle of compositionality discussed above. Similarly, knowledge bases can be composed of more specific knowledge bases. The compositional structure is based on the different levels of knowledge abstraction distinguished, and results in information and knowledge hiding.

*Composition of Information Types.* The relations between the information types distinguished above and generic information types distinguished for diagnosis, are depicted below in Figs. 12 and 13. The information types selected hypotheses, assessed hypotheses and focussed hypotheses (see Fig. 12) refer to the information type meta domain hypotheses. The information type meta-domain hypotheses refers to the information types hypotheses sorts and domain hypotheses. Note in this respect the meta-object distinction meta-domain hypotheses and domain hypotheses.

The information types predicted symptoms, observation information and selected observations refer to the information type meta-domain symptoms in a



**Figure 12.** Composition of information types: hypotheses.

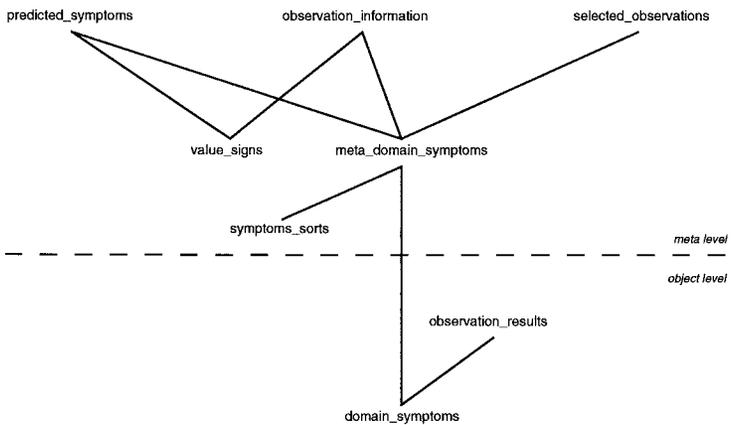
similar way (see Fig. 13). The information types predicted symptoms and observation information both refer to value signs.

The information types predicted symptoms, selected observations, and observation information all refer to the information type meta-domain hypotheses. Information type observation results refers to domain symptoms. Only domain symptoms and observation results are on the same object level; all other information types are at the same meta-level.

*Composition of Knowledge Bases.* The knowledge base hypothesis\_determination\_kb is composed of the generic knowledge base hypothesis\_refinement\_kb and the domain-specific knowledge base subhypotheses\_kb. The knowledge base observation\_determination\_kb is composed of the generic knowledge base generic\_obs\_determination\_kb and the domain-specific knowledge base obs\_relevance\_kb.

3.2.3. Task Control Knowledge

Processes at different abstraction levels can have different degrees of autonomy. For example, to constrain behavior the following forms of control can occur:



**Figure 13.** Composition of information types: observations.

<i>extent</i>	<i>to be derived</i>
<b>all p</b>	all possible targets
<b>every</b>	every target
<b>any</b>	any target
<b>any new</b>	any target not previously derived

**Figure 14.** Extents.

- *Fully decentralized control*  
Processes at the lowest process abstraction levels are all autonomous (i.e., not constrained by control from the higher levels) and the behavior of the processes at the higher abstraction levels emerges on the basis of the behavior of the processes at the lower levels (and the composition relation defined by the information links).
- *Fully centralized control*  
The top level process has control knowledge that constrains the behavior of the processes at the lower level of process abstraction (e.g., by a constraint prescribing a sequence of activation), which in their turn have control knowledge that constrains the processes at the levels below, and so on.
- *Centralized top level, decentralized lower level control*  
The top level has centralized control of the processes at the level of process abstraction level (just) below it and processes at the lowest levels of process abstraction are autonomous.
- *Decentralized top level, centralized lower level control*  
Control is minimal at the top level (e.g., agents are completely autonomous), but at the lower levels (within the agents) more centralized forms of control are used to obtain coherent behavior (the agents themselves have control over their activities). This form of control is often used within multiagent systems.

Different variants of control can be modeled within DESIRE, including all types of control mentioned above. Whether or not control knowledge is modeled to constrain the behavior of the subprocesses can be decided independently for each of the processes at each level of abstraction. Together with the information links, the control knowledge defines the process composition relation.

Task control knowledge specifies control of a component. A component can be in one of three *states*: it can be active, awake, or idle. A component in state idle is inactive. An active or awake component is actively pursuing a strategy defined by its task control focus and extent. A *task control focus* defines the focus of the process: Processes can have different foci, specified as task control foci. Dynamic selection of a task control focus is one of the ways in which a process can be controlled. To each task control focus a specific set of *targets* can be associated. These are the outputs the process tries to determine. An *extent* specifies to which extent targets associated to a task control must be derived (see Fig. 14).

If an *active* component *succeeds* or *fails* to derive the targets specified in its task control focus to a given extent, it becomes idle. An *awake component* is a component that is continually capable of deriving new information. Its task control focus and extent are used to focus the reasoning process in the same way as the task

control focus and extent are used to focus the reasoning in active components. Links can be either awake (in which case information is transferred as soon as it has become available), uptodate (in which case information has just been transferred). A link or component in state awake remains in this state continually, even if the component or link has nothing to do (in this way they are stand-by). A component or link in state awake reacts to arriving information immediately (event-driven). A component in state active and link in state uptodate becomes idle as soon as no new information can be derived (termination). A component or link in state idle does not react to arriving information. It can only react if its state is explicitly changed to awake or active, respectively uptodate.

The names of task control foci are part of the public task information in a component specification. The *initial task control focus* and the *initial extent* can be specified (the initial task information, as part of private task information) as well as which targets initially are associated to which task control focus (*initial targets*): initial kernel information, as part of private kernel information. A target specifies whether its aim is to confirm, determine, or reject a specific output atom. Evaluation criteria can be specified to assess a component's results. These may be the same as task control foci, or they may differ. Specific sets of targets also can be associated with evaluation criteria. Different evaluation criteria can be used to determine the status of a component's process, each with different implications: depending on which evaluation criteria have been successfully achieved, or failed, one or more components may be activated (made active or awake).

In the diagnostic reasoning example the component diagnostic reasoning has initial information about its task control focus diagnose fault and its extent, namely all p. This implies that the component will succeed if all possible targets associated to the task control focus diagnose fault have been derived. The definition of this information is included in the definition of diagnostic reasoning's private kernel information as initial kernel information: This specifies that the output atoms the component aims to determine are all atoms of the form diagnosis(H : HYPOTHESIS) for some instantiation of H : HYPOTHESIS for the task control focus diagnose fault. This is specified by the expression: diagnosis(H : HYPOTHESES) : confirm, where confirm is the *target type*. Note that targets are specified one level higher than the output atoms to which they refer: target information is not information about the world but about the component's process: a target describes meta-information expressed by a meta-atom of the form target(diagnose\_fault, diagnosis(H : HYPOTHESIS), confirm).

Instances of task control knowledge of the diagnostic reasoning example are used to illustrate task control specifications. Continual activity of both the component diagnostic reasoning and the component external world, and the two links observations and observation results, is specified as task control knowledge of the component top level.

```

if start
then next_component_state(diagnostic_reasoning, awake)
and next_component_state(external_world, awake)

```

**and next\_link\_state(observations, awake)**  
**and next\_link\_state(observation\_results, awake)**

The component diagnostic reasoning is made awake: The system is to continually react to new information in its effort to confirm all possible hypotheses. The external world is assumed to be continually awake, capable of executing an observation as soon as it arrives. Information provided by one of the two components is immediately transferred to the other by the respective link which is awake.

An example of *task control knowledge* within the component diagnostic reasoning is as follows:

**if start**  
**then next\_component\_state(hypothesis\_determination, active)**  
**and next\_link\_state(assessments, uptodate)**  
**and next\_task\_control\_focus(hypothesis\_determination, determine\_hypos)**  
**and next\_extent(hypothesis\_determination, any\_new)**

This rule states that when activated (in the sense of active or awake), the component diagnostic reasoning activates its subcomponent hypothesis determination with the state active, with the task control focus determine hypos and extent any new. In this rule, task control focus and extent do not actually need to be specified, as they never change during the process; they can be specified as initial task information. Another example of task control knowledge of the component diagnostic reasoning is:

**if component\_state(hypothesis\_determination, idle)**  
**and previous\_component\_state(hypothesis\_determination, active)**  
**and evaluation(hypothesis\_determination, hypos\_determined, any, succeeded)**  
**then next\_component\_state(hypothesis\_validation, awake)**  
**and next\_link\_state(hypotheses, uptodate)**

This statement specifies that if the component hypothesis determination terminates and succeeds in deriving any target associated to hypos determined (an evaluation criterion), then, at the next point in time, the component hypothesis validation has state **awake**. In addition, the link between hypothesis determination and hypothesis validation has become **uptodate**.

Task control knowledge constrains the process states of components. Typically, in the current version of the DESIRE software environment, task control knowledge is specified according to the pattern

**previous state & current state  $\Rightarrow$  next state**

This form [actually a form of executable temporal logic (cf. Barringer et al.<sup>4</sup>)] enables direct computation of the next control state from the current and previous

control state. In the near future, extensions of this format (e.g., by adding intermediate information types) will be supported.

### 3.3. Relation between Process Composition and Knowledge Composition

Each process in a process composition uses knowledge structures. Which knowledge structures (information types and knowledge bases) are used for which processes is defined by the relation between process composition and knowledge composition. The cells within the matrix depicted in Fig. 2 define these relations.

For the example diagnostic reasoning task the knowledge base related to the primitive component hypothesis evaluation is *anticausal\_fridge\_kb*. The knowledge base related to hypothesis\_determination is *hypothesis\_determination\_kb*; the knowledge base related to observation\_determination is *observation\_determination\_kb*.

## 4. DESIGN RATIONALE

One of the parts of the *design rationale* describes verification of the relevant properties of the designed system in relation to the design requirements identified in the problem description. Verification can be performed in a compositional manner as well; for example, as presented in Jonker and Treur.<sup>5</sup> Also the assumptions under which these properties hold are made explicit in the design rationale. Important design decisions are specified, together with some of the alternative choices that could have been made and the arguments in favour of and against the different options. At the operational level the design rationale includes decisions based on operational considerations, such as the choice to implement a parallel process on one or more machines, depending on the available capacity. For verification, the possible states of the world (world situations) taken into account are depicted in Fig. 15.

It is easy to verify that this anticausal knowledge is *correct* with respect to the possible world states depicted in Fig. 15 in the sense that, for given observation results, conclusions drawn by means of this knowledge are true in the world state in which the observations were performed. Moreover, the knowledge is *decisive* in the sense that if it is known whether there is a fridge problem, and sufficient observation information is available, then for each of the causes it can be derived whether or not it is true (one of the variants of completeness; see Treur and Willems<sup>6</sup> and Leemans, Treur, and Willems<sup>7</sup>).

A trivial approach would be to perform all possible observations and then draw a conclusion on the hypotheses (e.g., using the knowledge base above). For practical applications, such a trivial approach is not satisfactory: Many observations would have to be performed that do not contribute to the solution. The challenge lies mainly in the question which strategy can be followed to obtain observation information, sufficient to determine which hypotheses are true and which are false, but as economically as possible.

One strategy for the strategic reasoning processes described in the literature is called *hierarchical classification*. This strategy uses a taxonomy of types of

<i>atoms</i>	<i>possible world states</i>						
light	false	false	false	true	true	true	true
noise	false	true	false	false	true	true	false
cold	true	true	false	false	false	true	true
fridge problem	true	true	true	true	true	false	false
electricity problem	true	true	true	false	false	false	false
cooling problem	false	false	false	true	true	false	false
broken bulb	true	true	false	false	false	false	false
no power supply	false	false	true	false	false	false	false
broken pump	false	false	false	true	false	false	false
broken cooling system	false	false	false	false	true	false	false

**Figure 15.** Possible world states for the example domain.

problems (also called *abstract hypotheses*) of different levels of abstraction like the one used in the example model for diagnostic reasoning. The strategy of hypothesis determination is to first determine hypotheses at the highest level in the taxonomy, and depending on which of these abstract hypotheses are confirmed (by validation), proceed by selection of more specific hypotheses. Thus if an abstract hypothesis is confirmed, then the next hypotheses to focus on are its children in the taxonomy. The *specific hypotheses* for the diagnostic process as a whole are the hypotheses at the bottom of the taxonomy.

The abstract hypotheses in the taxonomy play two intermediary roles during the diagnostic process:

- They serve as intermediate results in the object level reasoning (e.g., using the knowledge base above).
- They play an important strategic role within the hypothesis determination task (meta-level reasoning about focusing the diagnostic process).

A confirmed abstract hypothesis defines a subtaxonomy that can be further pursued by selecting its children (subhypotheses in the taxonomy) as the next hypotheses to be validated. The children of nonconfirmed hypotheses are not further pursued. In this way each of the intermediate outcomes of the reasoning process (a confirmed abstract hypothesis) implies control of the direction of search. This hierarchical strategy of hypothesis determination is more economical than treating all specific hypotheses in sequence. Analyzing all specific hypotheses, for example, for a binary tree of depth  $n$  entails validation of  $2^n$  specific hypotheses, whereas for hierarchical classification following the taxonomy from top to bottom, two hypotheses per level need to be analysed, entailing validation of only  $2n$  hypotheses for the whole process.

## 5. CONTROL OF THE DYNAMICS OF REASONING BY DYNAMIC TARGETS

Control of reasoning processes can be imposed in different manners. One option is related to the process composition and uses task control knowledge as presented above to control the global phases (specified by different components), and task control foci in a reasoning process. In addition to this global form of control of reasoning, control at a more detailed level also is possible. An option for more fine-grained control is to dynamically generate goals (targets) for the reasoning process within a (primitive) component, thus supporting dynamic control of limited reasoning within a component. Another option for more fine-grained control is to dynamically generate additional presuppositions (assumptions) for the reasoning process in a component. In this section the control of reasoning by dynamic targets and by dynamic assumptions is discussed. The example of refrigerator diagnosis is used to illustrate the use of dynamic targets to control reasoning patterns in a fine-grained manner.

### 5.1. First Example Trace

The first example reasoning trace considered is depicted in Fig. 16. In this trace, the information states of the different components within the model are depicted. An information state is defined by a set of ground literals (those literals that have been input or derived within the component). The trace starts when the system is informed that there are problems with the fridge (1). This information is transferred to the component hypothesis determination (2), where, based on the taxonomy of hypotheses, two (abstract) hypotheses to be validated (electricity problem, cooling problem) are determined (3). These focus hypotheses are transferred to the component hypothesis validation (4), and within this component, to both the components observation determination and hypothesis evaluation (5). In the latter component they serve as targets: the reasoning process within the component is limited to deriving these outputs only. However, in the beginning there is not enough observation information available to derive either one of these targets. In observation determination, based on the hypotheses in focus, two observations to be performed (to observe light, cold) are determined (6), which are transferred to the output interface of hypothesis validation (7), and from there to the output interface of diagnostic reasoning (8).

The observations to be performed are transferred to the component external world (9), where they are actually performed (10). The observation results (light, not cold) are transferred to the component diagnostic reasoning (11), and within this component to hypothesis validation (12), and further down to the lower process abstraction level in hypothesis evaluation (13). Given the observation information (light, not cold), this component (which has electricity problem and cooling problem as targets) is able to derive that one of the abstract hypotheses (cooling problem) is true (14). This information is transferred to the component hypothesis determination, as is the information that both hypotheses in focus have been validated; based on this updated input, revision takes place, which leads to the retraction of the earlier derived conclusions on hypotheses to be validated (15).

time points	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
diagnostic reasoning	<pre>[ ]; [ confirmed(fridge_problem) ]  [ light, not cold ]; [ confirmed(fridge_problem),   selected_observation(light),   selected_observation(cold) ]  [ ]; [ confirmed(fridge_problem),   selected_observation(light),   selected_observation(cold) ]</pre>															
hypothesis determination	<pre>[ confirmed(fridge_problem) ]  [ confirmed(fridge_problem),   to_be_validated(electricity_problem),   to_be_validated(cooling_problem) ]  [ confirmed(fridge_problem),   validated(electricity_problem),   validated(cooling_problem),   confirmed(cooling_problem) ]  [ confirmed(fridge_problem),   validated(electricity_problem),   validated(cooling_problem),   confirmed(cooling_problem),   to_be_validated(broken_pump),   to_be_validated(broken_cooling_system) ]</pre>															
hypothesis validation	<pre>[ ]; [ focus_hypothesis(   electricity_problem),   focus_hypothesis(   cooling_problem),   confirmed(   fridge_problem), ]  [ light, not cold ]; [ focus_hypothesis(electricity_problem),   focus_hypothesis(cooling_problem),   confirmed(fridge_problem),   selected_observation(light),   selected_observation(cold) ]  [ ]; [ focus_hypothesis(electricity_problem),   focus_hypothesis(cooling_problem),   confirmed(fridge_problem),   selected_observation(light),   selected_observation(cold) ]</pre>															
observation determination	<pre>[ focus_hypothesis(electricity_problem),   focus_hypothesis(cooling_problem) ]  [ focus_hypothesis(electricity_problem),   focus_hypothesis(cooling_problem),   selected_observation(light),   selected_observation(cold) ]</pre>															
hypothesis evaluation	<pre>[ fridge_problem ]; [ ];  [ fridge_problem ]; [ target(he_tcf,   electricity_problem, det),   target(he_tcf,   cooling_problem, det) ]  [ light, not cold, fridge_problem ]; [ target(he_tcf, electricity_problem, det),   target(he_tcf, cooling_problem, det) ]  [ light, not cold, fridge_problem, cooling_problem ]; [ target(he_tcf, electricity_problem, det),   target(he_tcf, cooling_problem, det),   true(cooling_problem) ]</pre>															
external world	<pre>[ ]; [ target(ew_tcf, light, det),   target(ew_tcf, cold, det) ]  [ light, not cold ]; [ target(ew_tcf, light, det),   target(ew_tcf, cold, det) ]</pre>															

Figure 16. First example trace description.

After this revision, new (this time specific) hypotheses to be validated (broken pump and broken cooling system) are derived, based on the taxonomy of hypotheses (16), which again are transferred to hypothesis validation (17), and from there to the lower process abstraction levels of hypothesis evaluation and observation determination, where revision takes place (18). In observation determination a new observation to be performed (to observe noise) is found (19). Again, it is transferred to the output interface of hypothesis validation (20), and from there to the output interface of diagnostic reasoning (21).

time points	17	18	19	20	21	22	23	24	25	26	27	28	29
diagnostic reasoning	<pre>[ light, not cold , not noise ] ; [ confirmed(fridge_problem), confirmed(cooling_problem) selected_observation(noise) ]  [ light, not cold ] ; [ confirmed(fridge_problem), confirmed_cooling_problem), selected_observation(noise) ]  [ light, not cold , not noise ] ; [ confirmed(fridge_problem), confirmed(cooling_problem), confirmed(broken_pump) selected_observation(noise) ]</pre>												
hypothesis determination													
hypothesis validation	<pre>[ light, not cold ] ; [ focus_hypothesis( broken_pump), focus_hypothesis( broken_cooling_system), selected_observation(light), selected_observation(cold), confirmed(fridge_problem), confirmed(cooling_problem) ]  [ light, not cold , not noise ] ; [ focus_hypothesis( broken_pump), focus_hypothesis( broken_cooling_system), selected_observation(noise), confirmed(fridge_problem), confirmed(cooling_problem) ]  [ light, not cold ] ; [ focus_hypothesis( broken_pump), focus_hypothesis( broken_cooling_system), selected_observation(noise), confirmed(fridge_problem), confirmed(cooling_problem) ]  [ light, not cold , not noise ] ; [ focus_hypothesis( broken_pump), focus_hypothesis( broken_cooling_system), selected_observation(noise), confirmed(fridge_problem), confirmed(cooling_problem), confirmed(broken_pump) ]</pre>												
observation determination	<pre>[ focus_hypothesis(broken_pump), focus_hypothesis(broken_cooling_system), observed(light), observed(cold) ]  [ focus_hypothesis(broken_pump), focus_hypothesis(broken_cooling_system), observed(light), observed(cold) , selected_observation(noise) ]</pre>												
hypothesis evaluation	<pre>[ light, not cold, fridge_problem, cooling_problem] ; [ target(he_tcf, broken_pump, det), target(he_tcf, broken_cooling_system, det), true(cooling_problem) ]  [ light, not cold, not noise, fridge_problem, cooling_problem] ; [ target(he_tcf, broken_pump, det), target(he_tcf, broken_cooling_system, det), true(cooling_problem) ]  [ light, not cold, not noise, fridge_problem, cooling_problem, broken_pump ] ; [ target(he_tcf, broken_pump, det), target(he_tcf, broken_cooling_system, det), true(cooling_problem), true(broken_pump) ]</pre>												
external world	<pre>[ light, not cold ] ; [ target(ew_tcf, noise, det) ]  [ light, not cold, not noise ] ; [ target(ew_tcf, noise, det) ]</pre>												

Figure 16 (Continued)

The new observation to be performed is transferred to the component external world (22), where it is actually performed (23). The observation result (not noise) is transferred to the component diagnostic reasoning (24), and further down to hypothesis validation (25) and hypothesis evaluation (26). This time this component is able to derive that one of the specific hypotheses (broken pump) is true (27). This result is transferred up to the output interfaces of hypothesis validation (28) and diagnostic reasoning (29), respectively.

time points	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
diagnostic reasoning	[ ]; [ confirmed(fridge_problem) ]									[ not light, cold ] ; [ confirmed(fridge_problem), selected_observation(light), selected_observation(cold) ]						
hypothesis determination	[ confirmed(fridge_problem) ]									[ confirmed(fridge_problem), validated(electricity_problem), validated(cooling_problem), confirmed(electricity_problem) ]						
hypothesis validation	[ confirmed(fridge_problem), to_be_validated(electricity_problem), to_be_validated(cooling_problem) ]									[ confirmed(fridge_problem), validated(electricity_problem), validated(cooling_problem), confirmed(electricity_problem), to_be_validated(broken_bulb), to_be_validated(no_power_supply) ]						
hypothesis validation	[ ]; [ focus_hypothesis( electricity_problem), focus_hypothesis( cooling_problem), confirmed( fridge_problem) ]									[ not light, cold ] ; [ focus_hypothesis(electricity_problem), focus_hypothesis(cooling_problem), confirmed(fridge_problem), selected_observation(light), selected_observation(cold) ]						
observation determination	[ focus_hypothesis(electricity_problem), focus_hypothesis(cooling_problem) ]									[ focus_hypothesis(electricity_problem), focus_hypothesis(cooling_problem), selected_observation(light), selected_observation(cold) ]						
hypothesis evaluation	[ fridge_problem ] ; [ ]									[ fridge_problem ] ; [ target(he_tcf, electricity_problem, det), target(he_tcf, cooling_problem, det) ]						
hypothesis evaluation										[ fridge_problem, not light, cold ] ; [ target(he_tcf, electricity_problem, determine), target(he_tcf, cooling_problem, determine) ]						
external world										[ not light, cold, fridge_problem, electricity_problem ] ; [ target(he_tcf, electricity_problem, determine), target(he_tcf, cooling_problem, determine), true(electricity_problem) ]						
external world										[ ]; [ target(ew_tcf, light, determine), target(ew_tcf, cold, determine) ]						
external world										[ not light, cold ] ; [ target(ew_tcf, light, determine), target(ew_tcf, cold, determine) ]						

Figure 17. Second example trace description.

### 5.2. Second Example Trace

The second example reasoning trace considered in this chapter is depicted in Fig. 17. Steps (1)–(9) are exactly the same steps as for the first trace. A difference occurs at step (10). This time the observations results are (not light and cold). From (11) to (18) a similar pattern as in the first trace is followed: after transfer of the observation results (12, 13), within the component hypothesis evaluation, which has electricity problem and cooling problem as its targets, the truth of the abstract hypothesis electricity problem is derived from the observation information (not

<i>time points</i>	17 18 19 20 21
diagnostic reasoning	[ not light, cold ] ; [ confirmed(fridge_problem), confirmed(electricity_problem), confirmed(broken_bulb) ]
hypothesis determination	
hypothesis validation	[ not light, cold ] ; [ focus_hypothesis(broken_bulb), focus_hypothesis(no_power_supply), selected_observation(light), selected_observation(cold), confirmed(fridge_problem), confirmed(electricity_problem) ]  [ not light, cold ] ; [ focus_hypothesis(broken_bulb), focus_hypothesis(no_power_supply), selected_observation(light), selected_observation(cold), confirmed(fridge_problem), confirmed(electricity_problem), confirmed(broken_bulb) ]
observation determination	[ focus_hypothesis(broken_bulb), focus_hypothesis(no_power_supply), observed(light), observed(cold) ]  [ focus_hypothesis(broken_bulb), focus_hypothesis(no_power_supply), observed(light), observed(cold) ]
hypothesis evaluation	[ not light, cold, fridge_problem, electricity_problem ] ; : [ target(he_tcf, broken_bulb, determine), target(he_tcf, no_power_supply, determine), true(electricity_problem) ]  [ not light, cold, fridge_problem, electricity_problem, broken_bulb ] ; [ target(he_tcf, broken_bulb, determine), target(he_tcf, no_power_supply, determine), true(electricity_problem), true(broken_bulb) ]
external world	

Figure 17 (Continued)

light, cold) (14). This information is transferred to hypothesis determination entailing revision (15). After this the new (specific) hypotheses to be validated broken bulb and no electricity supply are derived, based on the taxonomy of hypotheses (16). These focus hypotheses are transferred down to the lower process abstraction level of hypothesis evaluation and observation determination, where revision takes place (17, 18). A difference occurs at (19). In observation determination no new observation to be performed is found (19). At the same time, the component hypothesis evaluation is able to derive one of the specific hypotheses in focus (which, actually, are targets of hypothesis evaluation), namely, broken bulb, without further observations. Therefore, this evaluation result is transferred up in the component hierarchy to hypothesis validation (20) and diagnostic reasoning (21), after which the diagnostic process stops.

Note that within the component hypothesis evaluation, the necessary

information to derive broken bulb was already available at step (13). However, at that time broken bulb was not a target of this component, and therefore it was not derived. Only after the targets of hypothesis evaluation had been changed and included broken bulb (18), was it actually derived (19). This shows how targets influence the reasoning behavior of the system.

## 6. CONTROL BY DYNAMIC GENERATION OF ASSUMPTIONS

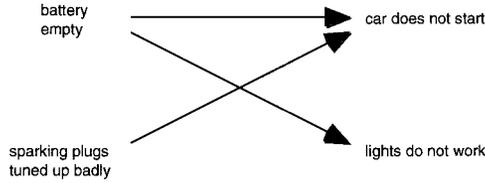
How to acquire and handle beliefs is an important but not simple task of an agent. In practice, information acquisition is often defeasible. Information acquired earlier may be found to be incorrect, and has to be retracted or revised. To support such processes, the status of information can be modeled as well, for example, whether or not a fact was assumed, or observed, or no matter which agent communicated the fact. The decision of an agent to actually believe acquired information may depend on an estimation of the degree to which the source of the information is deemed trustworthy. Therefore, different beliefs of an agent may have different status, and the agent has to be prepared to revise its beliefs in the light of newly acquired information anyway. A possible pattern of strategic reasoning is the following:

- identify the *required* information
- determine the *method to acquire* the required information; for example,
  - derive information in a *deductive* manner from available information
  - determine an appropriate *assumption*, and make this assumption
  - acquire additional information by *observation* (in interaction with the world)
  - acquire additional information by *communication* (ask another agent)
- *apply* the chosen method for information acquisition
- *verify* the obtained information in the light of other available information
- *integrate* the new information in the available information

In this section a reasoning method in which assumptions are dynamically added and retracted (sometimes called hypothetical reasoning) is discussed. The reasoning method is illustrated by a simple example of diagnostic reasoning on malfunctioning cars. Reasoning with and about assumptions entails deciding about a set of assumptions to be assumed for a while (reasoning *about* assumptions), and deriving which facts are logically implied by this set of assumptions (reasoning *with* assumptions). The derived facts may be evaluated; based on this evaluation some of the assumptions may be rejected and/or a new set of assumptions may be chosen (reasoning *about* assumptions). As an example, if an assumption has been chosen, and the facts derived from this assumption contradict information obtained from a different source (e.g., by observation), the assumption may be rejected, and the converse may be assumed.

Reasoning with and about assumptions is a reflective reasoning method. It proceeds by the following alternation of object level and meta-level reasoning, and upward and downward reflection:

- inspecting the information currently available (epistemic upward reflection)
- determining a set of assumptions (meta-level reasoning)



**Figure 18.** Causal knowledge for car diagnosis.

- temporarily assuming this set of assumptions (downward reflection of assumptions)
- deriving which facts follow from this assumed information (in the object level reasoning)
- inspecting the information currently available (epistemic upward reflection)
- evaluating the derived facts (meta-level reasoning)
- deciding to reject some of the assumptions and/or to choose a new set of assumptions based on this evaluation (meta-level reasoning)

As an example, if an assumption “a is true” has been chosen, and the facts derived from this assumption contradict information that is obtained from a different source, the assumption “a is true” may be rejected and the converse “a is false” may be assumed. This reasoning pattern also occurs in diagnostic reasoning based on causal knowledge (discussed below) this reasoning pattern occurs.

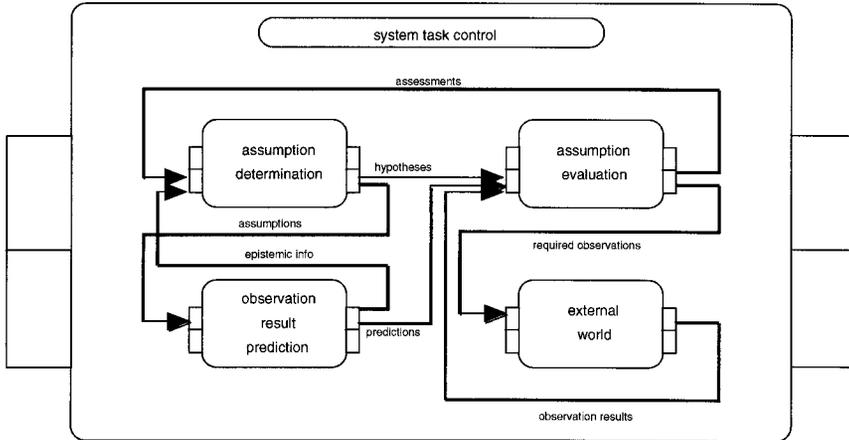
### 6.1. Car Diagnosis Based on Causal Knowledge

In this section a simple diagnostic reasoning pattern on car malfunctioning is analyzed. The causal knowledge on the domain of cars depicted in Fig. 18 is used:

- if the battery is empty  
then the lights do not work  
and the car will not start
- if the sparking-plugs are tuned up badly  
• then the car will not start

The causal knowledge could be easily extended, but for this example this knowledge suffices. At the meta-level the diagnostic reasoning process focuses on finding out whether an empty battery can be excluded as the cause of the problems. The following (simplified) meta-knowledge is used to reason about hypotheses; to propose hypotheses on which to focus, and to reject them if possible:

- if it has been observed that the car does not start  
and it is not known whether the hypothesis “the battery is empty” holds  
then “the battery is empty” is an adequate hypothesis on which to focus
- if it has been observed that the car does not start  
and it is true that the battery is nonempty



**Figure 19.** A generic model for reasoning with and about assumptions.

- and it is not known whether the hypothesis “the sparking-plugs are tuned up badly” holds
- then “the sparking-plugs are tuned up badly” is an adequate hypothesis on which to focus
- if the focus is on a hypothesis X and, assuming X, it has been derived that Y is the case and it has been observed that Y is not the case
- then the hypothesis X should be rejected

In the following section the generic model behind this reasoning pattern is introduced.

## 6.2. A Generic Model for Reasoning with and about Assumptions

The generic model for reasoning with and about assumptions consists of four primitive components: external world, predict observation results, assumption determination, and assumption evaluation (see Fig. 19). The first two of these components represent the object level, the last two the meta-level. The component observation result prediction reasons *with* assumptions, the two components assumption determination and assumption evaluation reason *about* assumptions.

The generic model is explained in this section for the domain of car diagnosis. In this example, the reasoning pattern starts with the information that the car will not start. Using this information and the knowledge described above, the following reflective reasoning pattern is followed:

### 1. component *assumption determination*

From the observation that the car does not start and that it is as yet unknown whether the battery is empty, draw the conclusion at the meta-level that “battery is empty” is an adequate hypothesis on which to focus

2. **information link assumptions (object-assumption)**  
Reflect this hypothesis downwards: introduce it at the object level as an assumption
3. **information link hypotheses (object-object)**  
Transfer the hypothesis to assumption evaluation
4. **component observation result prediction**  
Draw the conclusion at the object level that the lights do not work
5. **information link predictions (epistemic-object)**  
Reflect upwards the information that object level reasoning has predicted that the lights do not work
6. **component assumption evaluation**  
Draw the conclusion at the meta-level that the observation to find out whether the lights work is a useful observation to perform
7. **information link observation (object-target)**  
Reflect downwards the observation to be performed (as a target for the external world)
8. **component external world**  
Perform an observation in the external world to find out whether the lights work
9. **information link observation results (epistemic-object)**  
Reflect the observation result upwards
10. **component assumption evaluation**  
At the meta-level use the observation result that the lights work, and notice that the actual observation result contradicts the prediction on the observation. Draw the conclusion that the focus hypothesis “battery is empty” should be rejected
11. **information link assessments (object-object)**
12. **component assumption determination**

and so on

Note the interaction between the two levels at points 5 and 9 (epistemic upward interaction of type epistemic-object), at point 2 (downward interaction to make an assumption: type object-assumption), and at point 7 (downward interaction to set a target: type object-target). These are the points in the traces, where interaction between levels takes place.

### 6.3. Overview of the Components and Information Links

In this section a short overview is given of the components and information links of the generic model introduced in Section 1.2. and their instantiations for the domain of car diagnosis.

#### 6.3.1. Overview of the Components

The generic model is not specified in detail. Instead, the components and information links are depicted in Fig. 19, and relevant parts of the detailed design are presented.

##### *External World.*

This component is used for executing observations. It has no knowledge base.

Relevant <i>input atoms</i> (meta-level):	target(observations, A:OA, determine)
Relevant <i>output atoms</i> (object level):	car_starts, lights_work
(meta-level)	true(A:OA), false(A:OA), known(A:OA)



Relevant part of the *knowledge base*:

```

if predicted(O:OBSERVATION, S : SIGN)
then to_be_observed(O:OBSERVATION)

if assumed(H : HYPOTHESIS, S : SIGN)
  and predicted(O:OBSERVATION, pos)
  and observation_result(O:OBSERVATION, neg)
then rejected(H : HYPOTHESIS, S : SIGN)

if assumed(H : HYPOTHESIS, S : SIGN)
  and predicted(O:OBSERVATION, neg)
  and observation_result(O:OBSERVATION, pos)
then rejected(H : HYPOTHESIS, S : SIGN)

if assumed(H : HYPOTHESIS, S : SIGN)
then has_been_considered(H : HYPOTHESIS)

```

Note that the knowledge base specified above only contains domain independent knowledge. This knowledge base is small: The example reasoning pattern can be generated, but not much more. This could easily be extended.

### 6.3.2. Overview of the Information Links

The information links of the generic model for reasoning with and about assumptions are shortly described as follows.

*Assumptions (Type Object-Assumption).* The truth values of instances of output atoms `poss_assumption(H : HYPOTHESIS, S : SIGN)` of the component assumption determination are transferred to the same truth values of input meta-facts `assumption(H:IA, S : SIGN)` of the component observation result prediction. As a result, the hypothesis to which the assumption refers can be used in this component.

*Epistemic Information (Type Epistemic-Object).* The truth values of output meta-facts of the form `true(O:OA)` of the component observation result prediction are transferred to the same truth values of object level input atoms of the component assumption evaluation of the form `known_to_hold(O:OBSERVATION, pos)`. Similarly, the truth values of output meta-facts of the form `false(O:OBSERVATION)` of the component observation result prediction are transferred to the same truth values of object level input atoms of the component assumption evaluation of the form `known_to_hold(O:OBSERVATION, neg)`.

*Predictions (Type Epistemic-Object).* The truth values of output meta-facts of the form `true(O:OA)` of the component observation result prediction are transferred to the same truth values of object level input atoms of the component assumption evaluation of the form `predicted(O:OBSERVATION, pos)`. Similarly, the truth values of output meta-facts of the form `false(O:OBSERVATION)` of the component observation result prediction are transferred to the same truth values of object level

input atoms of the component assumption evaluation of the form `predicted(O:OBSERVATION, neg)`.

*Required Observations (Type Object-Target)*. The truth values of the object level output atoms `to_be_observed(O:OBSERVATION)` of the component assumption evaluation are transferred to the same truth values of meta-level input atoms `target(observations, O:OA, determine)` of the component external world.

*Observation Results (Type Epistemic-Object)*. This interaction transfers truth values of output meta-facts of the form `true(O:OA)` (resp. `false(O:OA)`) of the component external world to the same truth values of object level input atoms of the component assumption evaluation of the form `observation_result(O:OBSERVATION, pos)` (resp. `observation_result(O:OBSERVATION, neg)`).

*Hypotheses (Type Object-Object)*. The truth values of the object level output atoms of the form `poss_assumption(H : HYPOTHESIS, S : SIGN)` of the component assumption determination are transferred to the same truth values of object level input atoms `assumed(H : HYPOTHESIS, S : SIGN)` of the component assumption evaluation.

*Assessments (Type Object-Object)*. This interaction transfers truth values of object level output facts of the form `rejected(H : HYPOTHESIS)` and `has_been_considered(H : HYPOTHESIS)` of the component assumption evaluation to the same truth values of identical object level input atoms of the component assumption determination.

#### 6.4. The Dynamics of the Reasoning Method

Task control knowledge controls the activation of the components. For instance, first the component assumption determination is activated; next the component observation result prediction; if this succeeds (which is the case in the example trace below), then the component assumption evaluation is made active. After activation of the component external world to perform the observation, the component assumption evaluation is again activated. An example trace of a part of such a reasoning pattern is depicted in the example in Fig. 20 starting with the initial information in the component assumption determination that it has been observed that car starts is false and that the truth or falsity of the hypotheses battery empty and sparking plugs problem have not, as yet, been determined.

This example trace combines traces for the four components. For the object level components observation result prediction and external world the information states for two levels are depicted as:

```
[(object level information state)];  
[(meta-level information state)]
```

These meta-level information states show how the interaction between levels takes place. Their content is directly related to the content of the object level information state. For example, in the example trace, after the first activation (1)–(2) of the component assumption determination the information link assumptions is made up to date. This means that at the next moment in time the meta-statement `assumption(battery_empty, pos)` is true in the meta-information state of observation result

prediction (3). However, at the same moment the assumption is actually made: at the object level the atom `battery_empty` becomes true. This is an example of (downward) interaction between the levels. As a consequence the meta-atom `true(battery_empty)` is given truth value true in the meta-level information state (upward interaction between the levels). As soon as the information link becomes `uptodate` (i.e., at the next moment in time), this whole revision process is assumed to have finished. At this time point (3) only the result of the process of level interaction is visible, the different steps as sketched are not recorded separately. Therefore, the interaction between the levels is considered to take place instantaneously, as soon as new information arrives as input.

The same holds in the opposite direction. Based on the assumption `battery_empty` the component observation result prediction derives that `lights_work` is false (4). At the same time the related epistemic meta-atoms have been assigned their appropriate truth values; e.g., `false(lights_work)` gets truth value true, which, again, is the actual (upward) interaction between the levels. This meta-information is transferred to the component assumption evaluation by the information link predictions (5).

A next interaction between levels can be found after the component assumption evaluation has derived the atom `to_be_observed(lights_work)` (6). The information link required observations transfers this information to the meta-atom `target(observations, lights_work, determine)` of the component external world (7). This time the actual (downward) interaction between the levels immediately determines control of the component. The effect of the target is that the process focusses on `lights_work` only and does not try to find truth values for other atoms. For example, if the component external world actually stands for a human observer, then the target tells him or her what to observe. If, instead, the external world is a reasoning component, then the target focuses the component's reasoning by a goal-directed inference strategy with the target as its goal.

The result of the observation is that `lights_work` becomes true in the object level information state of the component external world (8). Again, an upward level interaction from object level to meta-level takes place instantaneously: at the same moment `true(lights_work)` becomes true. This observation result is transferred to the component assumption evaluation (9).

A fifth example of level interaction in the example trace is the second time assumptions are made in the component observation result prediction (13)–(14). This time the assumption earlier made that `battery_empty` is true is retracted. This takes place because, due to revision within the component assumption determination, the earlier drawn conclusion `poss_assumption(battery_empty, pos)` is retracted, which means that it is assigned truth value unknown. This truth value is propagated through the link assumptions: The meta-level atom `assumption(battery_empty, pos)` of the object level component observation result prediction is assigned the truth value unknown, the effect of which is that within the same component the object level atom `battery_empty` is assigned the truth value unknown (downward level interaction). At the same time the meta-level atom `assumption(battery_empty, neg)` is provided as input, which by downward level

<i>time point</i>	<i>external world</i>	<i>observation result prediction</i>	<i>assumption determination</i>	<i>assumption evaluation</i>
1			[ observation_result(car_starts, neg), not has_been_considered(battery_empty), not has_been_considered(sparking_plugs_problem)]	
2			[ observation_result(car_starts, neg), not has_been_considered(battery_empty), not has_been_considered(sparking_plugs_problem), poss_assumption(battery_empty, pos)]	
3		[ battery_empty ; [ assumption(battery_empty, pos), true(battery_empty) ]		
4		[ battery_empty, not lights_work ; [ assumption(battery_empty, pos), true(battery_empty), false(lights_work) ]		
5				[ assumed(battery_empty, pos), predicted(lights_work, neg) ]
6				[ assumed(battery_empty, pos), predicted(lights_work, neg), to_be_observed(lights_work) ]
7	[ ;	[ target(observations, lights_work, determine) ]		
8	[ lights_work ;	[ target(observations, lights_work, determine), true(lights_work) ]		
9				[ assumed(battery_empty, pos), predicted(lights_work, neg), to_be_observed(lights_work) observation_result(lights_work, pos) ]
10				[ assumed(battery_empty, pos), predicted(lights_work, neg), to_be_observed(lights_work), observation_result(lights_work, pos), rejected(battery_empty, pos), has_been_considered(battery_empty) ]

**Figure 20.** Trace of a hypothetical reasoning process.

```

11          [ observation_result(car_starts, neg),
              has_been_considered(battery_empty),
              not has_been_considered(sparking_plugs_problem)
              rejected(battery_empty, pos) ]

12          [ observation_result(car_starts, neg),
              has_been_considered(battery_empty),
              rejected(battery_empty, pos),
              poss_assumption(battery_empty, neg),
              poss_assumption(sparking_plugs_problem, pos)]

13      [ not battery_empty,
          sparking_plugs_problem ];
          [ assumption(battery_empty, neg),
            assumption(sparking_plugs_problem, pos)
            false(battery_empty),
            true(sparking_plugs_problem) ]

14      [ not battery_empty,
          sparking_plugs_problem,
          not car_starts];
          [ assumption(battery_empty, neg),
            assumption(sparking_plugs_problem, pos)
            false(battery_empty),
            true(sparking_plugs_problem),
            false(car_starts)]

```

**Figure 20.** (Continued)

interaction assigns the truth value false to the object level atom `empty_battery` (note that in this downward level interaction the truth value unknown for the object level atom `empty_battery` is overruled by the truth value false). Furthermore, the meta-level atom `assumption(sparking_plugs_problem, pos)` is provided as input, which, by downward level interaction assigns the truth value true to the object level atom `sparking_plugs_problem`. After all of these downward level interactions, the epistemic information at the meta-level is updated in an upward level interaction, finishing all level interaction, and resulting in the state at time point (13).

The generic model for reasoning with and about dynamic assumptions (based on temporal epistemic reflection), has been used to model, for example, default reasoning and reasoning based on the closed world assumption, in a large number of applications.

## 7. DISCUSSION

As shown in this paper component-based DESIRE models specify processes and knowledge at different levels of abstraction. Information exchange between processes and process sequencing are explicitly defined at each of the levels distinguished. Different levels of abstraction within the knowledge composition structure information types and knowledge bases. Reuse of generic models within DESIRE is supported by their transparent component-based structure. The basic principles behind component-based multiagent system design described in this paper (process and knowledge abstraction, compositionality, reusability, formal

semantics, and formal evaluation) are principles generally acknowledged to be of importance in both software engineering and knowledge engineering. The operationalization of these principles within a component-based design method for multiagent systems is, however, a distinguishing element.

Libraries of both generic models and instantiated components, of which some have been (re)used in this paper, support system designers at all levels of design. Generic agent models, generic task models, and generic models of reasoning patterns help structure the process of system design. Formal semantics provide a basis for methods for verification—an essential part of such a method.

The DESIRE design method is supported by the distributed DESIRE software environment. This environment includes tools to support system development during all phases of design. Graphical editors, for example, support specification of conceptual and detailed design of processes and knowledge. A detailed design is a solid basis to develop an operational implementation in any desired environment. An implementation generator supports prototype generation of both partially and fully specified models. The code generated by the implementation generator can be executed in a distributed execution environment, which runs on different platforms: UNIX- or LINUX-based and Windows-based systems.

A number of approaches to conceptual-level specification of multiagent systems have been recently proposed. On the one hand, general-purpose formal specification languages stemming from Software Engineering are applied to the specification of multiagent systems (e.g., Luck and d'Inverno<sup>8</sup>) for an approach using Z). A component-based design method such as DESIRE is committed to well-structured component-based designs that can be specified at a higher level of conceptualization than in Z or VDM and can be implemented automatically using automated prototype generators. On the other hand, new development methods for the specification of multi-agent systems have been proposed. These methods often commit to a specific agent architecture. For instance, Kinny, Georgeff, and Rao<sup>9</sup> describe a language, on the one hand, based on the BDI agent architecture (Rao and Georgeff<sup>10</sup>) and, on the other hand, based on object-oriented design methods. A more in depth comparative analysis of these methods from the perspective of compositionality and the related principles presented in this paper would be interesting further research.

The component-based approach to agent design followed in this paper has some aspects in common with object oriented design methods (e.g., Booch,<sup>11</sup> Coleman, Arnold, Bodoff, Dollin, Gilchrist, Hayes, and Jeremaes,<sup>12</sup> Rumbaugh, Blaha, Pelerlani, Eddy, and Lorensen<sup>13</sup>). However, there are differences as well. Examples of approaches to object oriented agent specifications can be found in Aridor and Lange,<sup>14</sup> and Kendall, Murali Krisna, Pathak, and Suresh.<sup>15</sup> A first interesting point of discussion is to what the difference is between agents and objects. Some tend to classify agents as different from objects. For example, Jennings and Wooldridge<sup>16</sup> compare objects with agents on the dimension of autonomy in the following way:

An object encapsulates some state, and has some control over this state in that it can only be accessed or modified via the methods that the object provides.

Agents encapsulate state in just the same way. However, we also think of agents as encapsulating *behavior*, in addition to state. An object does not encapsulate *behavior*: it has no control over the execution of methods—if an object  $\mathbf{x}$  invokes a method  $\mathbf{m}$  on an object  $\mathbf{y}$ , then  $\mathbf{y}$  has no control over whether  $\mathbf{m}$  is executed or not—it just *is*. In this sense, object  $\mathbf{y}$  is not autonomous, as it has no control over its own actions. In contrast, we think of an agent as having *exactly* this kind of control over what actions it performs. Because of this distinction, we do not think of agents as invoking methods (actions) on agents—rather, we tend to think of them *requesting* actions to be performed. The decision about whether to act upon the request lies with the recipient.

Some others consider agents as a specific type of objects that are able to decide by themselves whether or not they execute a method (objects that can say “no”) upon a received message, and that can initiate action (objects that can say “go”) without any message received.

A difference between the component-based design method DESIRE and object-oriented design methods in representation of basic functionality is that, within DESIRE declarative, knowledge-based specification forms are used, whereas method specifications (which usually have a more procedural style of specification) are used in object-oriented design. Another difference is that within DESIRE the notion of component and the composition relation are defined in a more specific manner: the static aspects by information links, and the dynamic aspects by (temporal) task control knowledge, according to a prespecified format. A similarity is the (re)use of generic structures: generic models in DESIRE, and patterns (Alexander,<sup>17</sup> Gamma, Helm, Johnson, and Vlissides,<sup>18</sup> Fowler,<sup>19</sup> Grand<sup>20</sup>) in object-oriented design methods, although their functionality and compositionality are specified in different manners, as discussed above.

The use of reflection principles to specify the control of complex dynamic reasoning patterns in diagnosis is described in Treur.<sup>21</sup> The approach to diagnosis using fine-grained dynamic control of (limited) reasoning by means of dynamic generation of targets in particular is described in Treur.<sup>22</sup> Temporal semantics for the dynamics of this reasoning pattern are introduced in Treur.<sup>23</sup> The generic model for reasoning with and about dynamic assumptions is introduced in Treur.<sup>24</sup> Semantics for this approach to reasoning with and about dynamic assumptions (temporal epistemic reflection) are introduced in Hoek, Meyer, and Treur.<sup>25</sup> In Brazier, Treur, and Wijngaards<sup>26</sup> and Brazier, Jonker, Treur, and Wijngaards,<sup>27</sup> more details about these diagnostic reasoning models can be found.

## References

1. Brazier FMT, Jonker CM, Treur J. Principles of compositional multi-agent system development. In: Cuenca J, editor. Proc 15th IFIP World Comput Cong, WCC'98, Conf Inf Technol Knowledge Syst, IT&KNOWS'98, 1998, Vienna; pp 347–360. Extended version in: Data Knowledge Eng 2002;41:1–28.
2. Herlea DE, Jonker CM, Treur J, Wijngaards NJE. Specification of behavioural requirements within compositional multi-agent system design. In: Garijo FJ, Boman M, editors.

- Multi-Agent System Engineering, Proc 9th Eur Workshop Model Autonomous Agents Multi-Agent World, MAAMAW'99, Lecture Notes in Artificial Intelligence 1647. Berlin: Springer-Verlag; 1999. pp 8–27.
3. Jonker CM, Kremer R, Van Leeuwen P, Pan D, Treur J. Visual and textual knowledge representation in DESIRE. In: Imam I, Kodratoff Y, El-Dessouki A, Ali M, editors. Multiple approaches intell syst Proc 12th Int Conf Ind Eng Appl AI Expert Syst, IEA/AIE'99), Lecture Notes in Artificial Intelligence 1611. Berlin: Springer-Verlag; 1999. pp 306–315.
  4. Barringer H, Fisher M, Gabbay D, Owens R, Reynolds M, editors. The imperative future: Principles of executable temporal logics. Chichester, United Kingdom: Research Studies Press; 1996.
  5. Jonker CM, Treur J. Compositional verification of multi-agent systems: A formal analysis of pro-activeness and reactiveness. In: de Roeper WP, Langmaack H, Pnueli A, editors. Proc Int Workshop Compositionality, COMPOS'97, Lecture Notes in Computer Science 1536. New York: Springer-Verlag; 1998. pp 350–380. Extended version in: Int J Coop Inf Syst 2002;11:51–92.
  6. Treur J, Willems M. A logical foundation for verification. In: Cohn AG, editor. Proc 11th Eur Conf Artif Intell, ECAI'94. Chichester: Wiley; 1994. pp 745–749.
  7. Leemans NEM, Treur J, Willems M. A semantical perspective on verification of knowledge. Data Knowledge Eng 2002;40:33–70.
  8. Luck M, d'Inverno M. A formal framework for agency and autonomy. In: Lesser V, editor. Proc First Int Conf Multi-Agent Syst, ICMAS'95. AAAI Press, Menlo Park, Cambridge, London; 1995. pp 254–260.
  9. Kinny D, Georgeff MP, Rao AS. A methodology and technique for systems of BDI agents. In: van der Velde W, Perram JW, editors. Agents breaking away, Proc 7th Eur Workshop Model Autonomous Agents Multi-Agent World, MAAMAW'96, Lecture Notes in Artificial Intelligence. 1038. Berlin: Springer-Verlag; 1996. pp 56–71.
  10. Rao AS, Georgeff MP. Modeling rational agents within a BDI architecture. In: Fikes R, Sandewall E, editors. Proc Second Conf Knowledge Representation Reason San Mateo, CA: Morgan Kaufman; 1991. pp 473–484.
  11. Booch G. Object-oriented analysis and design (2nd edition). Reading, MA: Addison-Wesley; 1994.
  12. Coleman D, Arnold P, Bodoff S, Dollin C, Gilchrist H, Hayes F, Jeremaes P. Object-oriented development: The FUSION method. Hempel Hempstead, United Kingdom: Prentice Hall International; 1994.
  13. Rumbaugh J, Blaha M, Pelerlani W, Eddy F, Lorensen W. Object-oriented modelling and design. Eaglewoods Cliffs, NJ: Prentice Hall; 1991.
  14. Aridor Y, Lange D. Agent design patterns: elements of agent application design. Proceedings of the Second International Conference on Autonomous Agents (Agents 98). ACM Press, New York, 1998, 10–115.
  15. Kendall EA, Murali Krishna PV, Pathak CV, Suresh CB. Patterns of Intelligent and Mobile Agents. Proceedings of the Second International Conference on Autonomous Agents (Agents 98), ACM Press, New York, 1998, 92–99.
  16. Jennings NR, Wooldridge M. Applications of intelligent agents. In: Jennings NR, Wooldridge M, editors. Agent technology: Foundations, applications, and markets. New York: Springer-Verlag; 1998. pp 3–28.
  17. Alexander C. A pattern language. Oxford: Oxford University Press; 1977.
  18. Gamma ER, Helm R, Johnson R, Vlissides J. Design patterns: Elements of reusable object-oriented software. Reading, MA: Addison-Wesley; 1995.

19. Fowler M. Analysis patterns: Reusable object models. Reading, MA: Addison Wesley; 1997.
20. Grand M. Patterns in Java: Volume 1. New York: Wiley; 1998.
21. Treur J. On the use of reflection principles in modelling complex reasoning. *Int J Intell Syst* 1991;6:277–294.
22. Treur J. Heuristic reasoning and relative incompleteness. *Int J Approximate Reason* 1993;8:51–87.
23. Treur J. Temporal semantics of meta-level architectures for dynamic control of reasoning. In: Fribourg L, Turini F, editors. *Logic Prog Synth Transformation-Meta-Program Logic, Proc Fourth Int Workshop Meta-Program Logic, META'94, Lecture Notes in Computer Science 883*. New York: Springer-Verlag; 1994. pp 353–376. Extended version in: *Int J Intell Syst* 2002;17:545–568.
24. Treur J. Interaction types and chemistry of generic task models. In: Linster M, Gaines B, editors. *Proc Fifth Eur Knowledge Acquisition Workshop, EKAW'91, GMD Studien 211, Crieff, Scotland*. 1992. pp 390–414.
25. van der Hoek W, Meyer J-J, Treur J. Formal semantics of temporal epistemic reflection, In: Fribourg L, Turini F, editors. *Logic Prog Synth Transformation-Meta-Program Logic, Proc Fourth Int Workshop Meta-program Logic, META'94, Lecture Notes in Computer Science 883*. New York: Springer-Verlag; 1994. pp 332–352.
26. Brazier FMT, Treur J, Wijngaards NJE. The acquisition of a shared task model. In: Shadbolt N, O'Hara K, Schreiber ATH, editors. *Advances in knowledge acquisition, Proc 9th Eur Knowledge Acquisition, Workshop, EKAW'96, Lecture Notes in Artificial Intelligence 1076*, Berlin: Springer-Verlag; 1996. pp 278–289.
27. Brazier FMT, Jonker CM, Treur J, Wijngaards NJE. On the use of shared task models in knowledge acquisition, strategic user interaction and clarification agents. *Int J Human-Comput Stud* 2000;52:77–110.