ELSEVIER

Data & Knowledge Engineering 41 (2002) 1–27

DATA & KNOWLEDGE ENGINEERING

www.elsevier.com/locate/datak

Principles of component-based design of intelligent agents

Frances M.T. Brazier, Catholijn M. Jonker, Jan Treur *

Department of Artificial Intelligence, Vrije Universiteit Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, Netherlands

Received 24 July 2001; received in revised form 2 October 2001; accepted 24 October 2001

Abstract

Compositional multi-agent system design is a methodological perspective on multi-agent system design based on the software engineering principles process and knowledge abstraction, compositionality, reuse, specification and verification. This paper addresses these principles from a generic perspective in the context of the compositional development method DESIRE. An overview is given of reusable generic models (design patterns) for different types of agents, problem solving methods and tasks, and reasoning patterns. Examples of supporting tools are described. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Component-based; Design; Agent; Reuse; Generic model

1. Introduction

The area of Component-Based Software Engineering is currently a well-developed area of research within Software Engineering; e.g., [14,25,40,41]. More specific approaches to component-based design of *agents* are often restricted to object-oriented implementation environments, usually based on Java [2,22,34]. In these approaches, agents are often kept simple; rarely knowledge-based architectures are covered, and if so, only with agents that are based on one knowledge base [36]. Techniques for complex, knowledge-intensive tasks and domains developed within Knowledge Engineering play no significant role. In contrast, this paper addresses the design of component-based intelligent agents in the sense that: (1) the agents can be specified on

* Corresponding author. Tel.: +31-20-444-7763; fax: +31-20-444-7653.
E-mail addresses: frances@cs.vu.nl; http://www.cs.vu.nl (F.M.T. Brazier), jonker@cs.vu.nl; http://www.cs.vu.nl (C.M. Jonker), treur@cs.vu.nl; http://www.cs.vu.nl (J. Treur).

PII: S0169-023X(01)00058-1

a conceptual (design) level instead of an implementation level, and (2) specifications exploit knowledge-based techniques as developed within Knowledge Engineering, enabling the design of more complex agents, for example for knowledge-intensive applications.

The compositional multi-agent design method DESIRE (DEsign and Specification of Interacting REasoning components) supports the design of component-based autonomous interactive agents. Both the intra-agent functionality (i.e., the expertise required to perform the tasks for which an agent is responsible in terms of the knowledge, and reasoning and acting capabilities) and the *inter-agent functionality* (i.e., the expertise required to perform and guide co-ordination, co-operation and other forms of social interaction in terms of knowledge, and reasoning and acting capabilities) are explicitly modelled. DESIRE views the individual agents and the overall system as compositional structures – hence all functionality is designed in terms of interacting, compositionally structured components. In this paper an overview is given of the principles behind this design method. DESIRE has been used in a number of application domains, e.g. [6–13]. Section 2 briefly discusses the process of design and the role of compositionality within this process. Section 3 discusses the problem analysis and requirements elicitation process. Section 4 introduces the elements used to specify conceptual design and detailed design: process composition, knowledge composition and their relationships. Design rationale and verification is discussed in Section 5. Section 6 discusses the notion of component-based generic models that form the basis of reuse during design processes. The availability of a large variety of such generic models for agents and tasks forms an important basis of the design method. In this section a number of these models are presented. Section 7 briefly discusses the graphical software environment to support the design process. Section 8 concludes the paper with a discussion.

2. The design process and types of compositionality

The design of a multi-agent 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 analyse or verify the resulting behaviour. 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 multi-agent systems in DESIRE.

During a multi-agent system design process, DESIRE distinguishes the following descriptions (see Fig. 1):

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

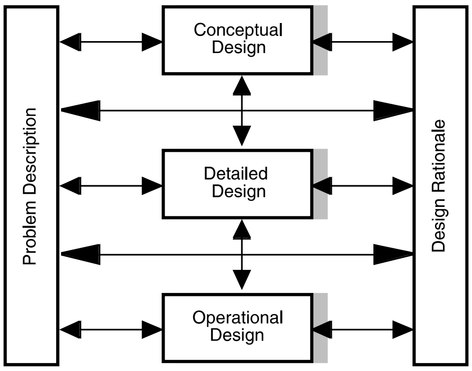


Fig. 1. Problem description, levels of design and design rationale.

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, the interaction between agents, and the interaction 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 behaviour. A detailed design provides sufficient detail 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 multi-agent 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.

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 modelled as (process) *components* composed of components at the adjacent lower level.

Processes within a multi-agent system may be viewed as the result of interaction between more specific processes. A complete multi-agent system may, for example, be seen to be 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 may 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* may be composed of a

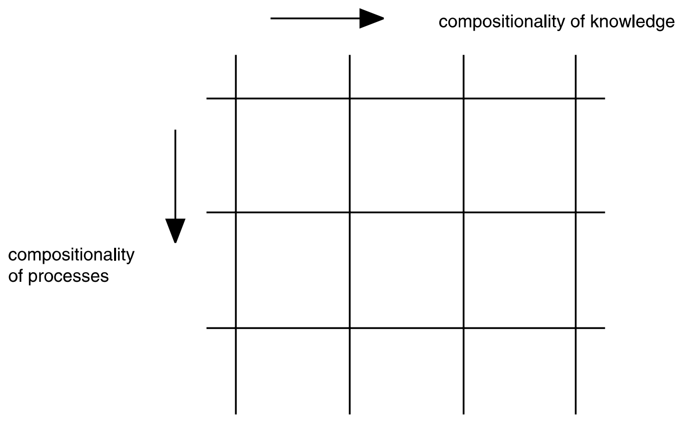


Fig. 2. Compositionality of processes and compositionality of knowledge.

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 a model: by defining processes and knowledge at different levels of abstraction, unnecessary details 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.

3. Problem description and requirements elicitation

Which techniques are used to acquire a *problem description* is not pre-defined. Techniques vary in their applicability, depending on, for example, the situation, the task, 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.

Requirements Engineering is a well-studied field of research. In recent years requirements engineering for distributed and agent systems has been studied, e.g., [18–20,23,31]. At the level of the multi-agent system, requirements are related to the dynamics of interaction and co-operation patterns. At the level of individual agents, requirements are related to agent behaviour. Due to the dynamic complexity, analysis and specification of such requirements is a difficult process.

Requirements can be expressed in an informal, semi-formal or formal manner. In the context described above, the following is an informally expressed requirement for the dynamics of the multi-agent system as a whole:

R2: Each service request must be followed by an adequate service proposal after a certain time delay.

In a structured, semi-formal manner, this requirement can be expressed as follows:

```
if at some point in time
        an agent A outputs:      a service request, to an appropriate other agent B
then at a later point in time
        agent B outputs:         a proposal for the request, to agent A
and at a still later point in time
        agent A outputs:         proposal is accepted, to agent B
```

The following temporal formalisation is made:

$\forall \mathscr{M}, \mathrm{t}, \mathrm{A}\ \exists \mathrm{B}$
[holds(state($\mathscr{M}$, t, output(A)), communication_from_to(request(r), A, B))
$\Rightarrow$ [$\exists \mathrm{t2} > \mathrm{t1} > \mathrm{t}$ holds(state($\mathscr{M}$, t1, output(B)),
communication_from_to(proposal_for(p, r), B, A))
$\wedge$ holds(state($\mathscr{M}$, t2, output(B)),
communication_from_to(accepted_proposal_for(p, r), A, B))]]

The formal language used is comparable to situation calculus (e.g., compare $\models$ to the holds-predicate), but with explicit variables for traces and time. The expression
holds(state($\mathscr{M}$, t, output(A)), communication_from_to(request(r), A, B))
means that within trace $\mathscr{M}$ at time point t a communication statement communication_from_to(request(r), A, B) is placed in the output interface of agent A. Here a trace is a sequence over time of three-valued information states of the system, including input and output information states of all of the agents, and their environment. The time frame can be discrete, or a finite variability assumption can be used. For further details on the use of this predicate logic temporal language, see [23].

Besides requirements on the dynamics of the overall multi-agent system, also requirements can be expressed on the behaviour of single agents. For example, an agent who is expected to adequately handle service requests should satisfy the following behaviour requirements:

A1: If the agent B receives a request for a service from a client A
And the necessary information regarding this client is **not** available
Then agent B issues a request for this information to that client.

Requirements on the dynamics of a multi-agent system are at a higher process abstraction level than the behaviour requirements on agents.

4. Conceptual design and detailed design

Conceptual and detailed designs consist of specifications of the following three types:

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

These three types of specifications are discussed in more detail below.

4.1. 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. Depending on the context in which a system is to be designed two different views can be taken: a task perspective, and a multi-agent perspective. The *task perspective* refers to the view in which the processes needed to perform an overall task first are identified. These processes (or sub-tasks) are then *delegated* to appropriate agents and the external world, after which these agents and the external world are designed. The *multi-agent perspective*refers to the view in which agents and an external world are first identified and then the processes within each agent and within the external world.

4.1.1. Identification of processes at different levels of abstraction

Processes can be described at different levels of abstraction; for example, the processes for the multi-agent system as a whole, processes within individual agents and the external world, processes within task-related components of individual agents.

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

Modelling process abstraction levels. The levels of process abstraction identified are modelled as *abstraction/specialisation 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, optimisation, etc.

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 sub-component relation, defining which components are a sub-component of a which other component. The distinction of different process abstraction levels results in process hiding.

4.1.2. Composition

The way in which processes at one level of abstraction in a system are composed of processes at the adjacent lower abstraction level in the same system is called *composition*. This composition of processes is described not only by the component/sub-component relations, but in addition 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. 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 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.

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 further guide processing.

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 'awake', (e.g., able to react to new input as soon as it arrives), others may need to be activated explicitly.

4.2. Knowledge composition

Knowledge composition identifies 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 an m to n correspondence between processes and knowledge structures, with $m, n > 1$.

4.2.1. 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*. These knowledge structures can be identified and described at different levels of abstraction. At the higher levels 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, or in formal textual form.

Knowledge bases. Knowledge bases use ontologies defined in information types. Relations between information types and knowledge bases define precisely which information types are used. The relationships between the concepts specified in the information types are defined by the knowledge bases during detailed design.

4.2.2. 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.

4.3. 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.

5. Design rationale and compositional verification

The *design rationale* behind a design process describes the relevant properties of a system in relation to the design requirements and the relevant assumptions. The initial requirements are stated in the initial problem description, others originate during a design process, and are added to the problem description. Important design decisions are made explicit, 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. This information is of particular importance for verification.

Requirements imposed on multi-agent systems designed to perform complex and interactive tasks are often requirements on the behaviour of the agents and the system. As in non-trivial applications the dynamics of a multi-agent system and the control thereof are of importance, it is vital to understand how system states change over time. In principle, a design specifies which changes are possible and anticipated, and which behaviour is intended. To obtain an understanding of the behaviour of a compositional multi-agent system, its dynamics can be expressed by means of the evolution of information states over time. If information states are defined at different levels of process abstraction, behaviour can be described at different levels of process abstraction as well.

The purpose of *verification* is to prove that, under a certain set of assumptions, a system adheres to a certain set of properties, for example the design requirements. A compositional multi-agent system verification method takes the process abstraction levels and the related compositional structure into account. In [6,17,28], a compositional verification method is described and applied to diagnostic reasoning, co-operative information gathering agents, and negotiating agents, respectively. The verification process is done by a mathematical proof (i.e., a proof in the form to which mathematicians are accustomed) that the specification of the system, together with the assumptions, imply the properties that a system needs to fulfil. The requirements are formulated formally in terms of temporal semantics. During the verification process the requirements of the system as a whole are derived from properties of agents (one process abstraction level lower) and these agent properties, in turn, are derived from properties of the agent components (again one abstraction level lower).

Primitive components (those components that are not composed of others) can be verified using more traditional verification methods for knowledge-based systems (if they are specified by means of a knowledge base), or other verification methods tuned to the type of specification used. Verification of a (composed) component at a given process abstraction level is done using:

- properties of the sub-components it embeds,
- a specification of the process composition relation,
- environmental properties of the component, (depending on the rest of the system, including the world).

This introduces compositionality in the verification process: given a set of environmental properties, the proof that a certain component adheres to a set of behavioural properties depends on the (assumed) properties of its sub-components, and the composition relation: properties of the interactions between those sub-components, and the manner in which they are controlled. The assumptions under which the component functions properly, are the properties to be proven for its sub-components. This implies that properties at different levels of process abstraction play their own role in the verification process.

Compositional verification has the following advantages; see also [1,24,28]:

- Reuse of verification results is supported (refining an existing verified compositional model by further decomposition, leads to verification of the refined system in which the verification structure of the original system can be reused).
- Process hiding limits the complexity of the verification per abstraction level.

A condition to apply a compositional verification method described above is the availability of an explicit specification of how the system description at an abstraction level is composed from the descriptions at the adjacent lower abstraction level.

The formalised properties and their logical relations, resulting from a compositional verification process, provide a more general insight in the relations between different forms of behaviour. For example, in [17] different properties of diagnostic reasoning and their logical relations have been formalised in this manner, and in [28] the same has been done for pro-activeness and reactiveness properties for co-operative information gathering agents. In [6] termination and successfulness properties for negotiation processes are analysed.

6. Reusability and generic models

The iterative process of modelling processes and knowledge is often resource-consuming. To limit the time and expertise required to design a system a development method should reuse as many elements as possible. Within a compositional development method, generic agent models and task models, and existing knowledge structures (ontologies and knowledge bases) may be used for this purpose. Which models are used, depends on the problem description: existing models are examined, discussed, rejected, modified, refined and/or instantiated in the context of the problem at hand. Initial abstract descriptions of agents and tasks can be used to generate a variety of more specific agent and task descriptions through refinement and composition (for which existing models can be employed as well).

Agent models and task models can be generic in two senses: with respect to the processes (abstracting from the processes at the lower levels of process abstraction), and with respect to the knowledge (abstracting from lower levels of knowledge abstraction, e.g., a specific domain of application). Often different levels of genericity of a model may be distinguished. A *refinement* of a generic model to lower process abstraction levels, resulting in a more specific model is called a *specialisation*. A refinement of a generic model to lower knowledge abstraction levels, e.g., to model a specific domain of application, is called an *instantiation*. Compositional system design focuses on both aspects of genericity, often starting with a generic agent model. This model may be modified or refined by specialisation and instantiation. The process of specialisation replaces a single 'empty' component of a generic model by a composed component (consisting of a number of sub-components). The process of instantiation takes a component of a generic model and fills it with (domain) specific information types and knowledge bases. During these refinement processes components can also be deleted or added. The compositional structure of the design is the basis for performing such operations on a design.

The applicability of a generic agent model depends on the basic characteristics of an agent in the problem description. The applicability of a generic task model for agent-specific tasks depends not only on the type of task involved, but also the way in which the task is to be approached. Since the availability of a variety of generic models is crucial for the quality of support that can be offered during a design process, in this section a number of generic models available in DESIRE are discussed.

6.1. Generic agent models

Characteristics of automated agents vary significantly depending on the purposes and tasks for which they have been designed. Agents may or may not, for example, be capable of communicating with other agents. A fully reactive agent may only be capable of reacting to incoming information from the external world. A fully cognitive and social agent, in comparison, may be capable of planning, monitoring and effectuating co-operation with other agents. Which agent models are most applicable to a given situation (possibly in combination) is determined during system design. Generic models for weak agents, co-operative agents, BDI-agents and deliberative normative agents are briefly described below.

6.1.1. Generic model for the weak agent notion: GAM

The Generic Agent Model (GAM) depicted in Fig. 3 supports the notion of a weak agent, for which *autonomy*, *pro-activeness*, *reactiveness* and *social abilities* are distinguished as characteristics; cf. [42]. This type of agent:

- reasons about its own processes (supporting *autonomy* and *pro-activeness*),
- interacts with and maintains information about other agents (supporting *social abilities*, and *reactiveness* and *pro-activeness* with respect to other agents),
- interacts with and maintains information about the external world (supporting *reactiveness* and *pro-activeness* with respect to the external world).

The six components are: Own Process Control (OPC), Maintenance of World Information (MWI), World Interaction Management (WIM), Maintenance of Agent Information (MAI), Agent Interaction Management (AIM), and Agent Specific Tasks (AST). The processes involved in controlling an agent (e.g., determining, monitoring and evaluating its own goals and plans) but also the processes of maintaining a self model are the task of the component Own Process Control. The processes involved in managing communication with other agents are the task of the component Agent Interaction Management. Maintaining knowledge of other agents' abilities and knowledge is the task of the component Maintenance of Agent Information. Comparably, the processes involved in managing interaction with the external (material) world are the task of the component

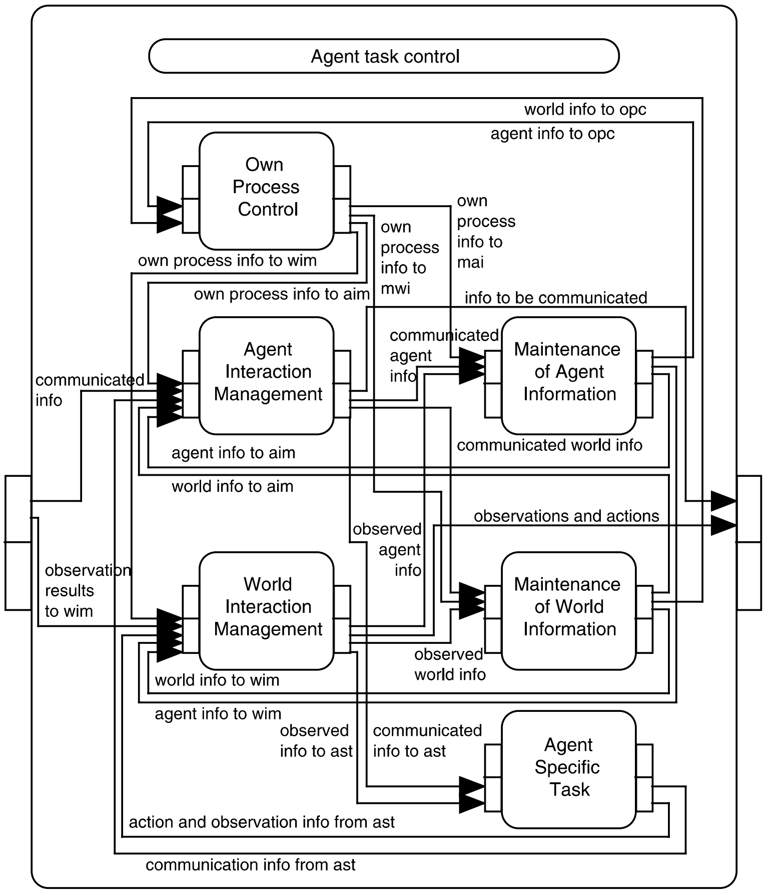


Fig. 3. Generic model for the weak agent notion.

World Interaction Management. Maintaining knowledge of the external (material) world is the task of the component Maintenance of World Information. The specific task for which an agent is designed (for example: design, diagnosis), is modelled in the component Agent Specific Task. Existing (generic) task models may be used to further specialise this component; see Section 6.2.

6.1.2. Generic co-operative agent model: GCAM

If an agent explicitly reasons about co-operation with other agents, the generic model for a weak agent depicted in Fig. 3 can be extended to include an additional component for co-operation management. This component, the Co-operation Management component includes the knowledge needed to acquire co-operation, as shown in Fig. 4.

To achieve co-operation between a number of agents requires specific plans devised specifically for this purpose. These plans are the result of reasoning by the component Generate Project. This component identifies commitments needed for all agents involved, and modifies existing plans when necessary.

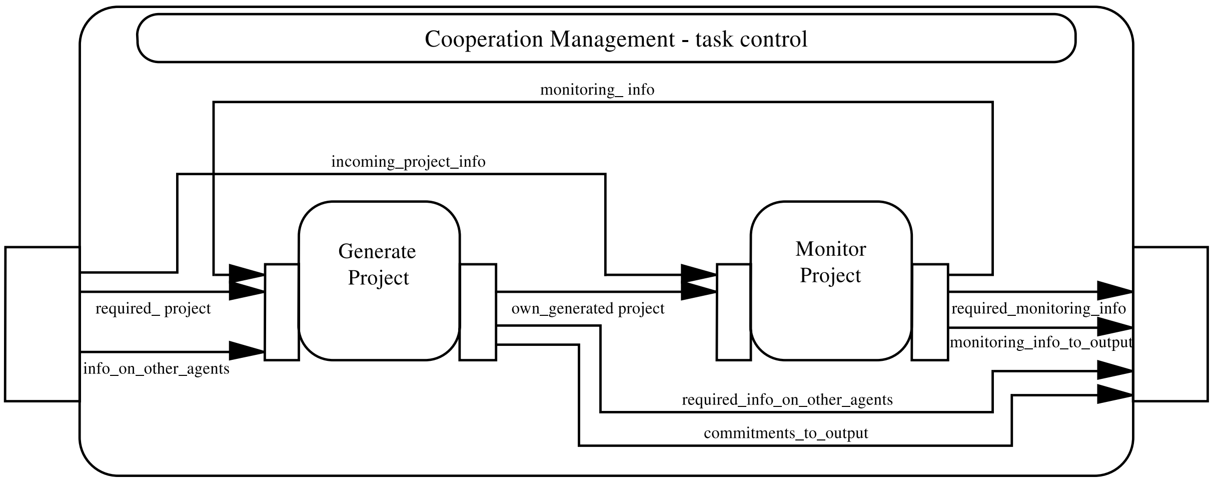


Fig. 4. Refinement of co-operation management in the generic co-operative agent model GCAM.

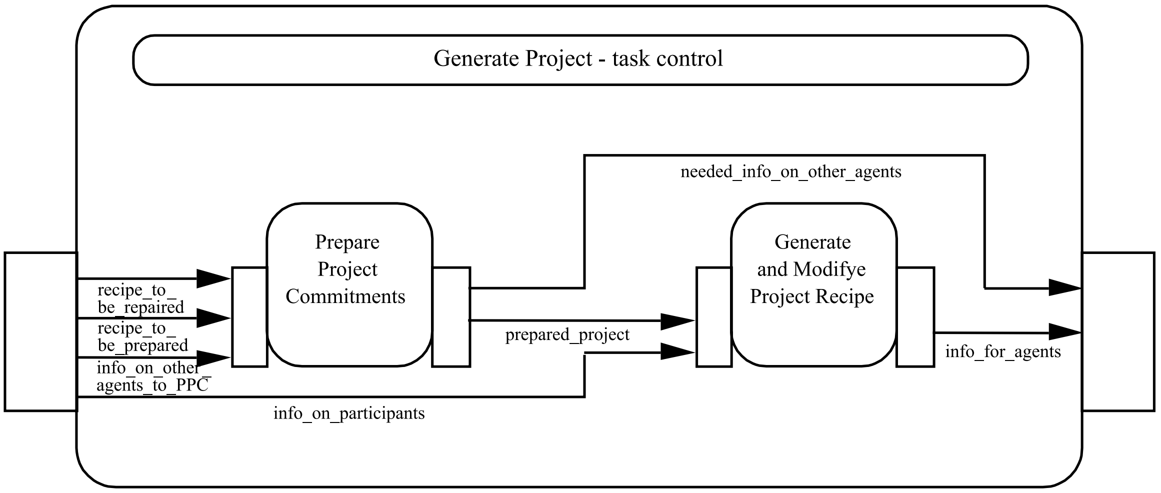

Generate Project - task control
needed_info_on_other_agents
Prepare Project Commitments
Generate and Modifye Project Recipe
recipe_to_ be_repaired
recipe_to_ be_prepared
info_on_other agents_to_PPC
info_on_participants
prepared_project
info_for_agents

Fig. 5. Composition of the component Generate Project in GCAM.

The composition of the component Generate Project in Fig. 5 includes the two components Prepare Project Commitments (for composing an initial project team) and Generate and Modify Project Recipe (to determine a detailed schedule for the project, in interaction with the project team members) for these two purposes. Execution of a plan, also part of co-operation, is monitored by each individual agent involved. This is the task of the component Monitor Project. The two sub-components of this component depicted in Fig. 6, Assess Viability (to determine the feasibility of a plan) and Determine Consequences (consequences of changes for the agents involved). The generic model of a cooperative agent is based on the approach put forward in [26]. For a more detailed explanation of the composition of processes, the knowledge involved and the interaction between components, see [9].

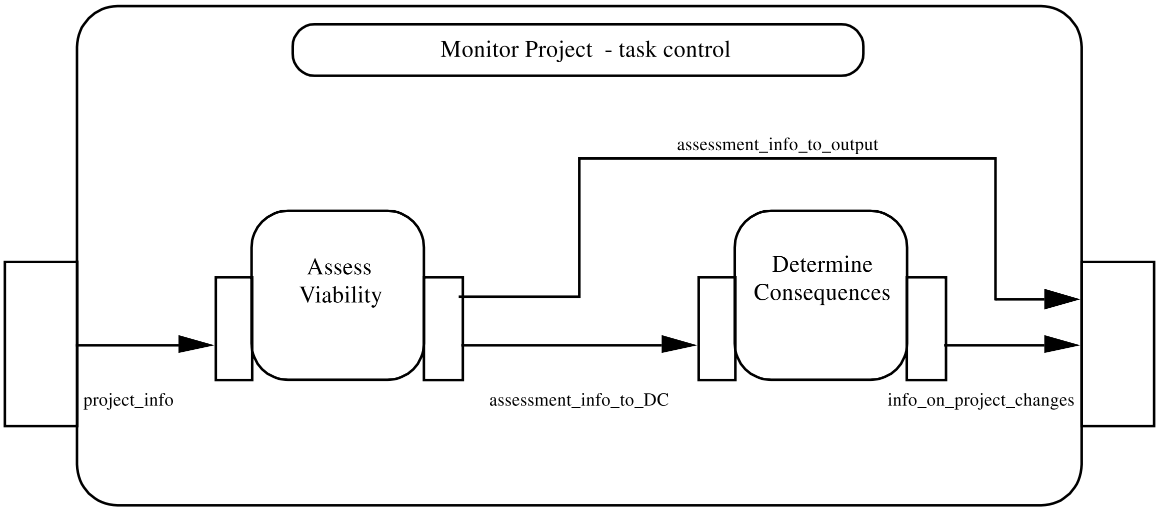


Fig. 6. Composition of the component Monitor Project in GCAM.

6.1.3. Generic model of a BDI-agent: GBDIM

An agent that bases its control of its own processes on its own beliefs, desires, commitments and intentions is called a BDI-agent. The BDI-agent model is a refinement of the model for a weak agent GAM. The refinement of own process control in the Generic Model for BDI-agents, GBDIM, is shown in Fig. 7.

Beliefs, desires, and intentions together with commitments, are determined in separate components with interaction between all three. A distinction is made between: (1) intentions and commitments with respect to goals, and (2) intentions and commitments with respect to plans. This distinction involves different types of knowledge and, as a result, is modelled by two different components as depicted in Fig. 8.

Please note that the influence of intentions and commitments with respect to goals directly influences intentions and commitments with respect to plans, and vice versa. For more detail see [8].

6.1.4. Generic model of a deliberative normative agent: GDNM

In many agent societies norms are assumed to play a role. It is claimed that not only following norms, but also the possibility of 'intelligent' norm violation are of importance. Principles for agents that are able to behave deliberatively on the basis of explicitly represented norms are identified and incorporated in a generic model for a deliberative normative agent. Using this agent model, norms can be communicated, adopted and used as meta-goals on the agent's own processes. As such they have impact on deliberation about goal generation, goal selection, plan generation and plan selection.

This generic model for an agent that uses norms in its deliberative behaviour is a refinement of the generic agent model GAM. A new component is included for society information, the component Maintenance of Society Information (MSI) at the top level and the component Own Process Control is refined as shown in Fig. 9. For more details, see [15].

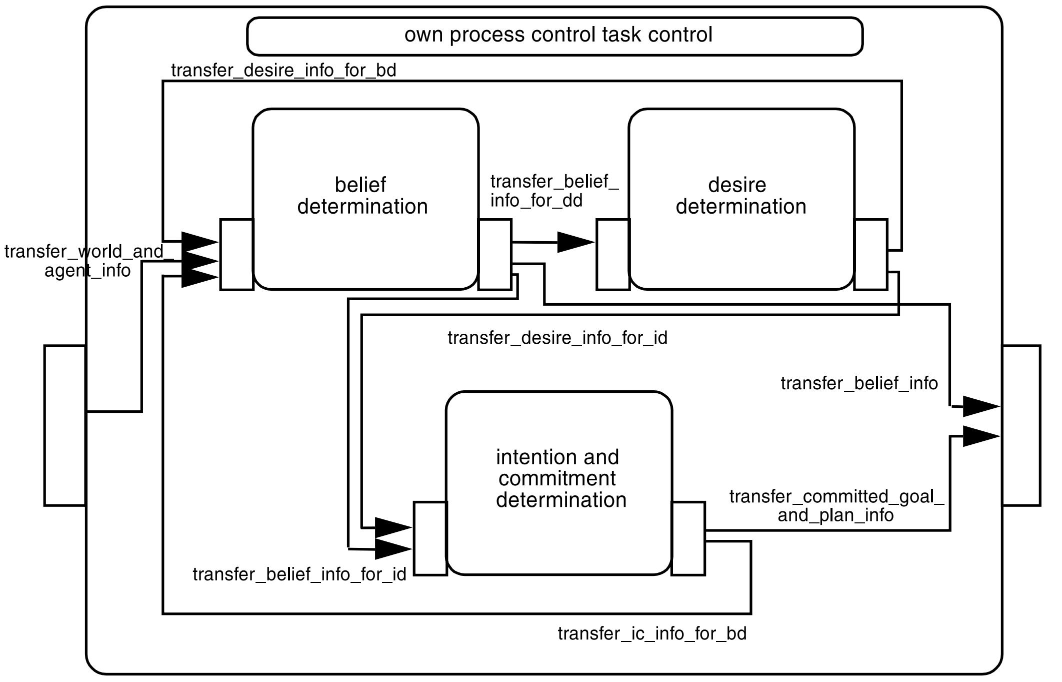

own process control task control
transfer_desire_info_for_bd
belief determination
transfer_belief_info_for_dd
desire determination
transfer_world_and_agent_info
transfer_desire_info_for_id
transfer_belief_info
intention and commitment determination
transfer_committed_goal_and_plan_info
transfer_belief_info_for_id
transfer_ic_info_for_bd

Fig. 7. Refinement of the component Own Process Control in the generic BDI-agent model GBDIM.

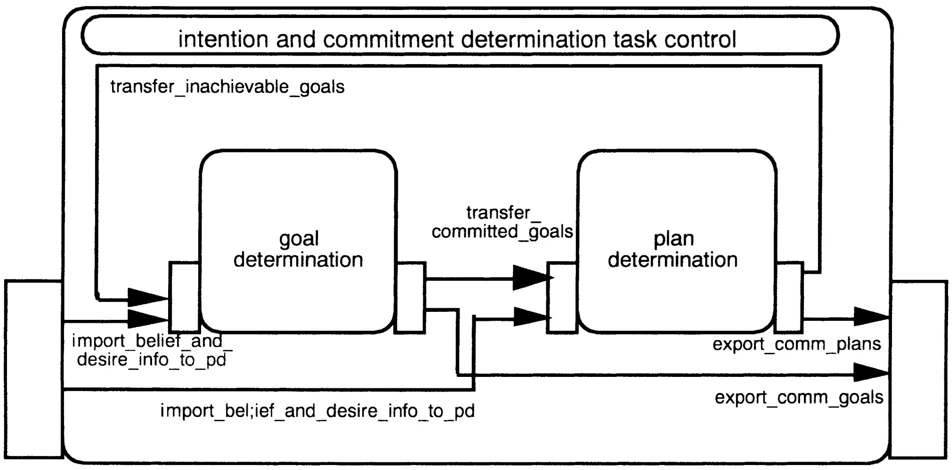

intention and commitment determination task control
transfer_inachievable_goals
goal determination
transfer_committed_goals
plan determination
import_belief_and_desire_info_to_pd
import_bel;ief_and_desire_info_to_pd
export_comm_plans
export_comm_goals

Fig. 8. Refinement of the component Intention and Commitment determination.

6.2. Generic models of problem solving methods and tasks

The specific tasks for which agents are designed vary significantly. Likewise the variety of tasks for which generic models based on specific problem solving methods have been developed is wide: diagnosis, design, process control, planning and scheduling are examples of tasks for which

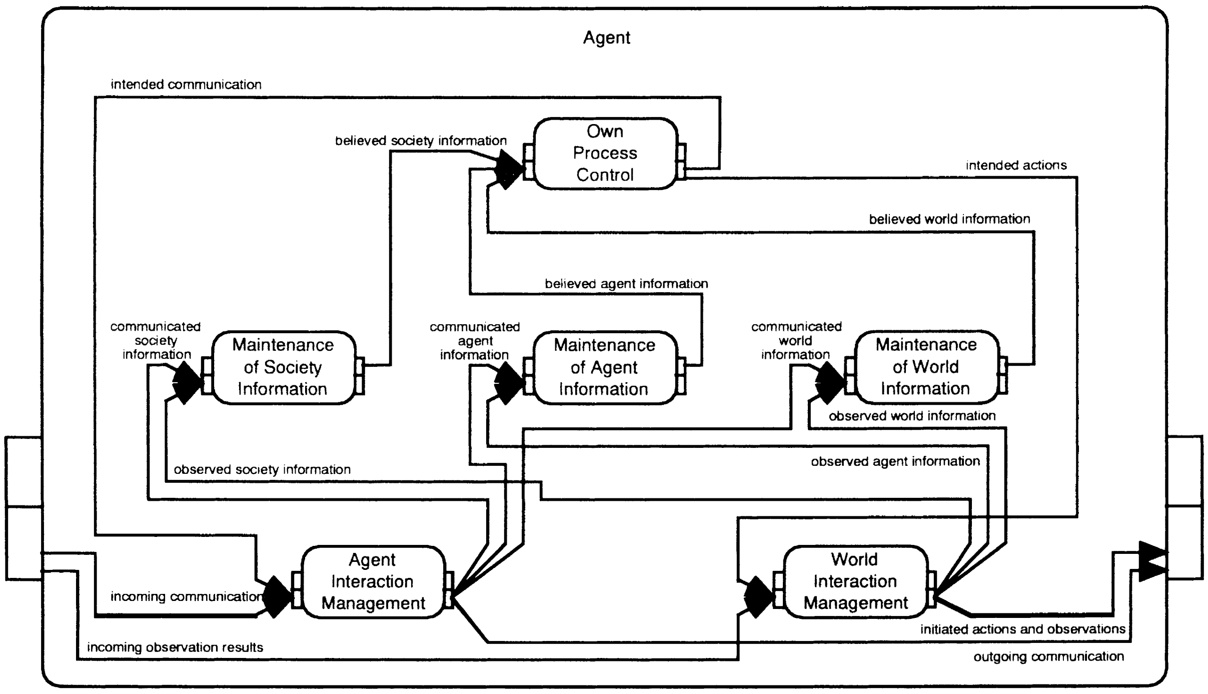

Agent
intended communication
believed society information
Own Process Control
intended actions
believed world information
believed agent information
communicated society information
Maintenance of Society Information
communicated agent information
Maintenance of Agent Information
communicated world information
Maintenance of World Information
observed world information
observed agent information
observed society information
Agent Interaction Management
World Interaction Management
incoming communication
initiated actions and observations
incoming observation results
outgoing communication

Fig. 9. A generic model for a deliberative normative agent: GDNM.

generic models are available. In this section compositional generic task models (developed in DESIRE) for the first three types of tasks are briefly described. These task models can be combined with any of the agent models described above: they can be used to specialise the agent specific task component.

6.2.1. A generic model for diagnostic tasks: GDIM

Tasks specifically related to *diagnosis* are included in the generic task model of diagnosis (for a top level composition, see Fig. 10). This generic model (the Generic DIagnosis Model GDIM) is based on determination and validation of hypotheses. It subsumes both causal and anti-causal diagnostic reasoning. Application of this generic model for both types of diagnosis is discussed in [12].

The component Hypothesis Determination is used to dynamically focus on certain hypotheses during the process. Hypothesis Validation includes determination of the observations (Observation Determination) needed to validate a hypothesis (which are transferred to the external world to be performed), and evaluation of the results of observation with respect to the hypothesis in focus (Hypothesis Evaluation) (see also Fig. 11).

6.2.2. A generic model for design tasks: GDEM

The compositional Generic DEsign Model (GDEM; see Fig. 12) [10] is based on a logical analysis of design processes and on analyses of applications, including elevator configuration and

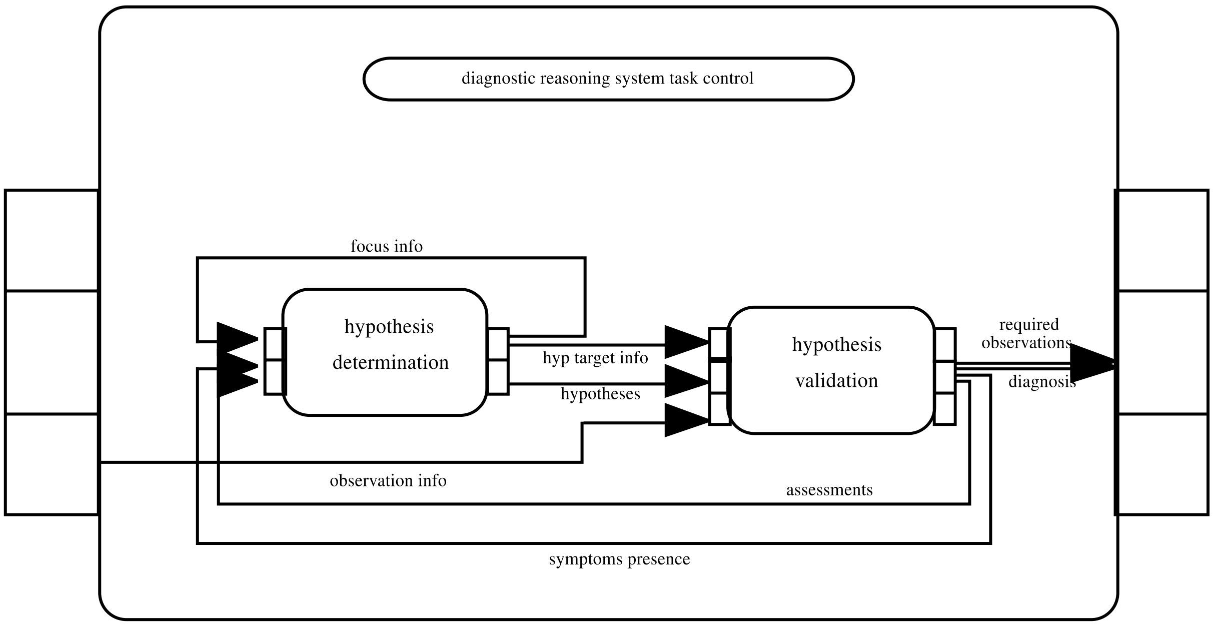


Fig. 10. Generic task model of diagnosis: GDIM.

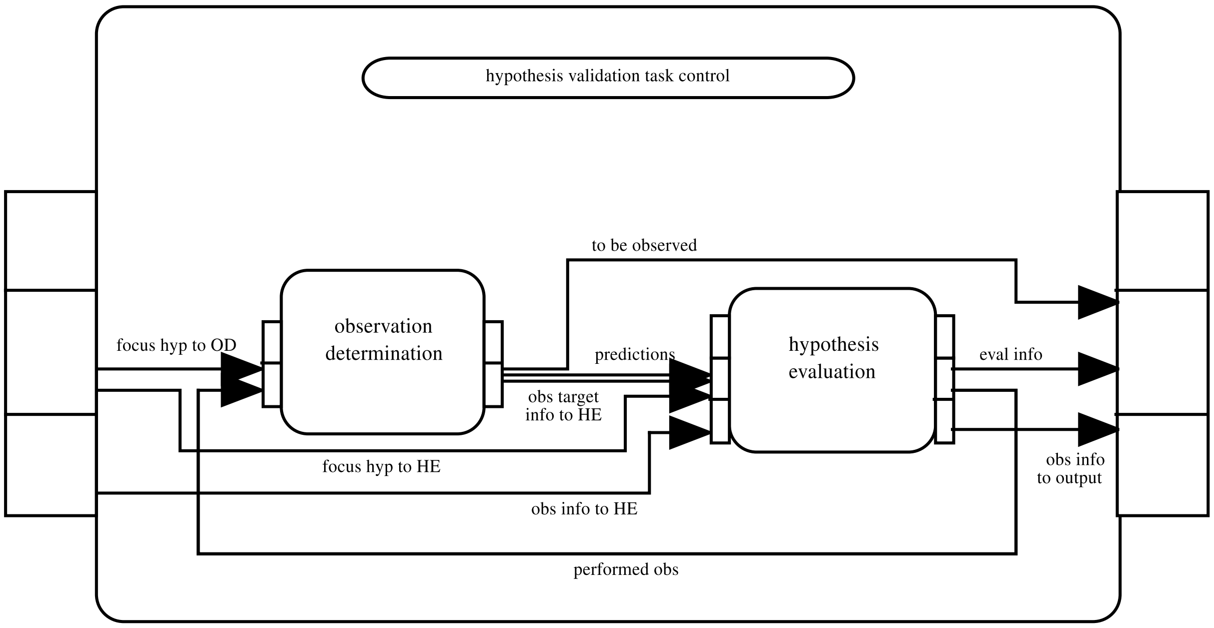


Fig. 11. Composition of the component Hypothesis Validation in GDIM.

design of environmental measures [13]. In this model Requirement Qualification Sets Manipulation (component RQS Manipulation or RQSM), Design Object Description Manipulation (component DOD Manipulation or DODM) and Design Process Co-ordination (DPC) are

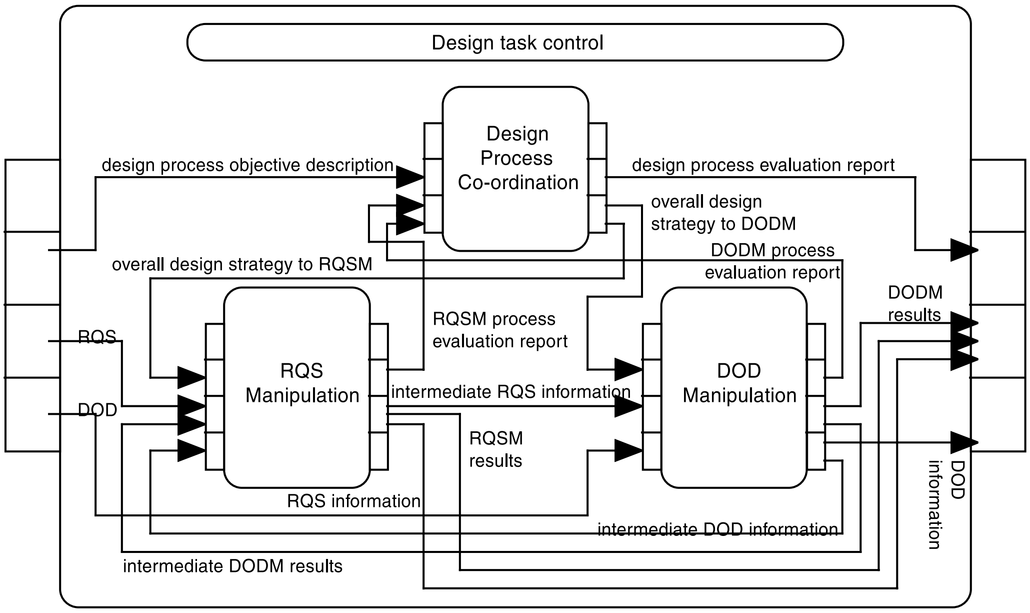

Design task control
Design Process Co-ordination
design process objective description
design process evaluation report
overall design strategy to DODM
overall design strategy to RQSM
DODM process evaluation report
DODM results
RQS
DOD
RQS Manipulation
RQSM process evaluation report
intermediate RQS information
DOD Manipulation
RQSM results
RQS information
intermediate DOD information
intermediate DODM results
DOD information

Fig. 12. Composition of the design task: GDEM.

distinguished as three separate interacting processes. The model provides a generic structure which can be refined for specific design tasks in different domains of application.

An initial design problem statement is expressed as a set of initial requirements and requirement qualifications. *Requirements* impose conditions and restrictions on the structure, functionality and behaviour of the *design object* for which a structural description is to be generated during design. *Qualifications* of requirements are qualitative expressions of the extent to which (individual or groups of) requirements are considered hard or preferred, either in isolation or in relation to other (individual or groups of) requirements. At any one point in time during design, the design process focuses on a specific subset of the set of requirements. This subset of requirements plays a central role; the design process is (temporarily) committed to the current requirement qualification set: the aim of generating a design object description is to satisfy these requirements.

During design the subsets of the set of requirements considered may change as may the requirements themselves. The same holds for design object descriptions representing the structure of the object to be designed.

The component Requirement Qualification Set Manipulation has four sub-components:

- RQS modification: the current requirement qualification set is analysed, proposals for modification are generated, compared and the most promising (according to some measure) selected,
- deductive RQS refinement: the current requirement qualification set is deductively refined by means of the theory of requirement qualification sets,
- current RQS maintenance: the current requirement qualification set is stored and maintained,
- RQSM history maintenance: the history of requirement qualification sets modification is stored and maintained.

The component Manipulation of Design Object Descriptions also has four sub-components:

- DOD modification: the current design object description is analysed in relation to the current requirement set, proposals for modification are generated, compared and the most promising (according to some measure) selected,
- deductive DOD refinement: the current design object description is deductively refined by means of the theory of design object descriptions,
- current DOD maintenance: the current design object description is stored and maintained,
- DODM history maintenance: the history of design object descriptions modification is stored and maintained.

More detail on this model can be found in [10]. In [11] the different levels of strategic reasoning in the model are described in more detail, including the component Design Process Co-ordination for the highest level of strategic reasoning.

6.2.3. A generic model for process control tasks: GPCM

Process control involves three sub-processes: process analysis, simulation of world processes and plan determination. These sub-processes are represented explicitly at the top-level of the Generic Process Control Model (GPCM) depicted in Fig. 13.

Process Analysis involves evaluation of the process as a whole and determination of the observations to be performed in the external world. This is depicted below in Fig. 14 in the composition of the component Process Analysis.

Note that two types of observations can be performed: *incidental observations* that return an observation result for only the current point in time, and *continuous observations* that continuously return all updated observation results as soon as changes in the world occur.

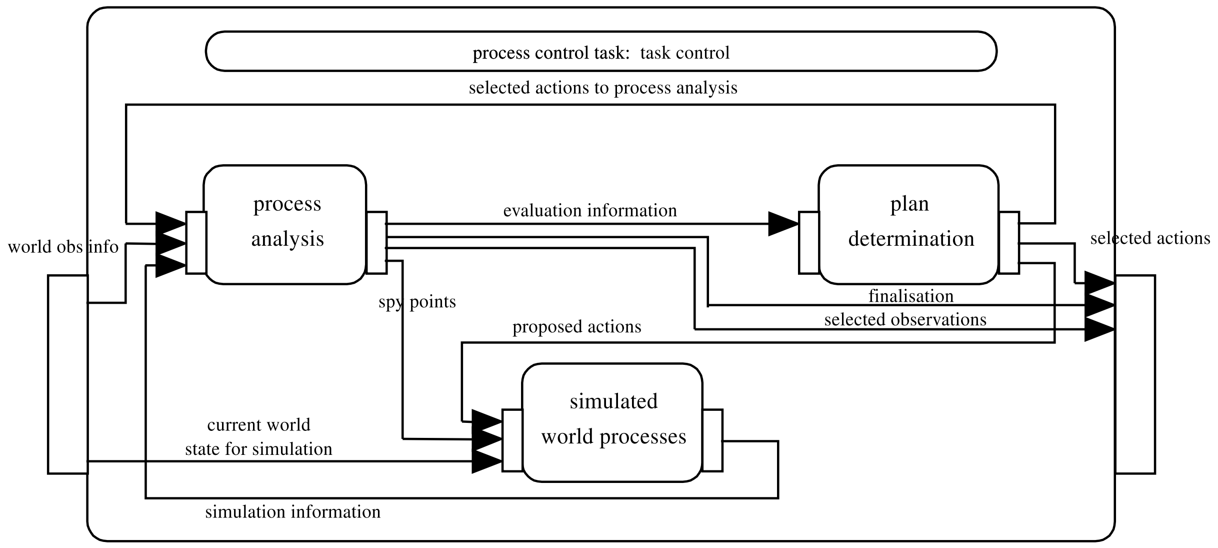


Fig. 13. Process composition of process control: GPCM.

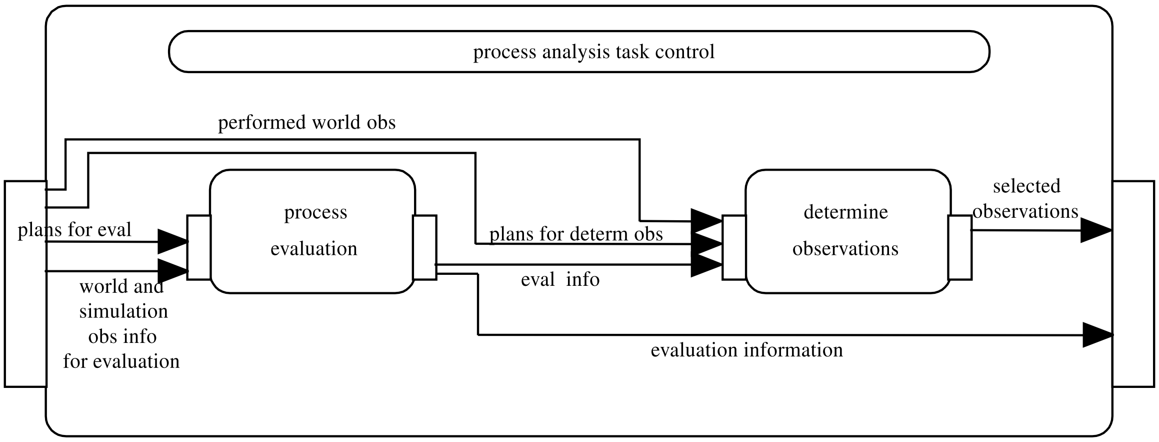


Fig. 14. Process composition of process analysis: information links.

6.3. Generic models of reasoning patterns

An example of a generic model for a specific reasoning pattern, is a model for reasoning patterns in which assumptions are dynamically added and retracted (sometimes called hypothetical reasoning), is discussed. 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). For example, if an assumption is 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),
- assuming this set of assumptions for a while (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),

and so on.

As an example, if an assumption 'a is true' is chosen, and the facts derived from this assumption contradict information that is obtained from a different source, the assumption 'a is

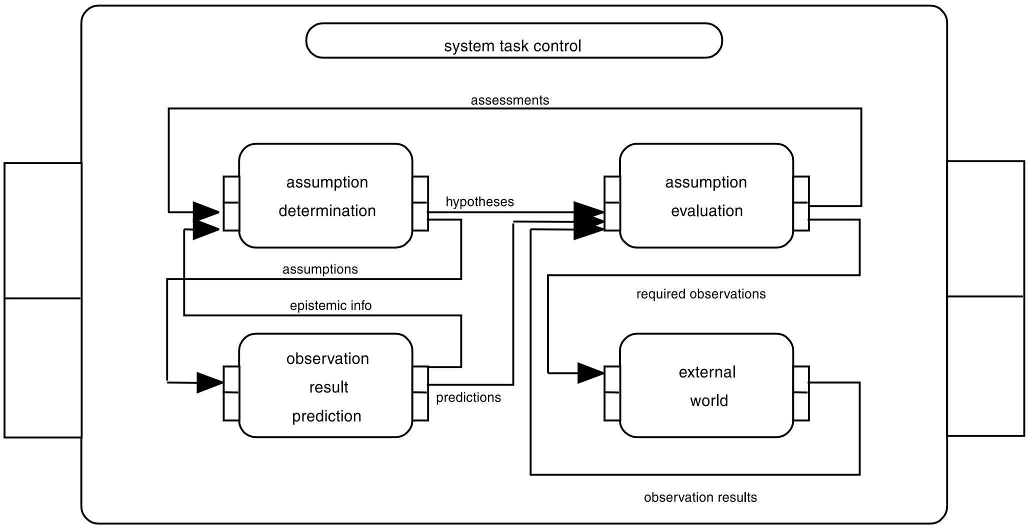


Fig. 15. A generic model for reasoning with and about assumptions: GARM.

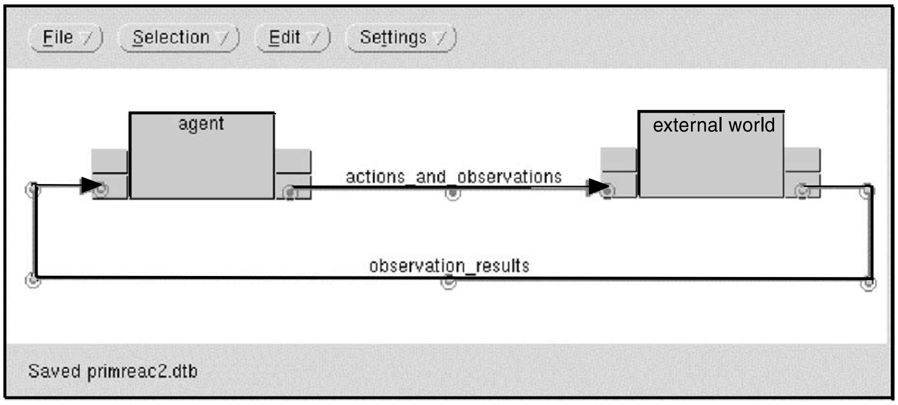
true' may be rejected and the converse 'a is false' may be assumed. This reasoning pattern also occurs in diagnostic reasoning based on causal knowledge.

The generic model for reasoning with and about assumptions consists of four primitive components: External world, Observation Results Prediction, Assumption Determination, Assumption Evaluation (see Fig. 15). 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. Note that this generic reasoning model is applied, among others, in the generic model for diagnosis GDIM presented in Section 6.2.1. However, the model has other types of application as well. For example, on the basis of this generic reasoning model, more specialised models have been designed for:

- a generic model for default reasoning with explicit strategic knowledge on resolution of conflicting defaults (GDRM),
- a generic model for reasoning on the basis of a Closed World Assumption (GCWARM), with possibilities for context-sensitive informed and scoped variants of the Closed World Assumption.

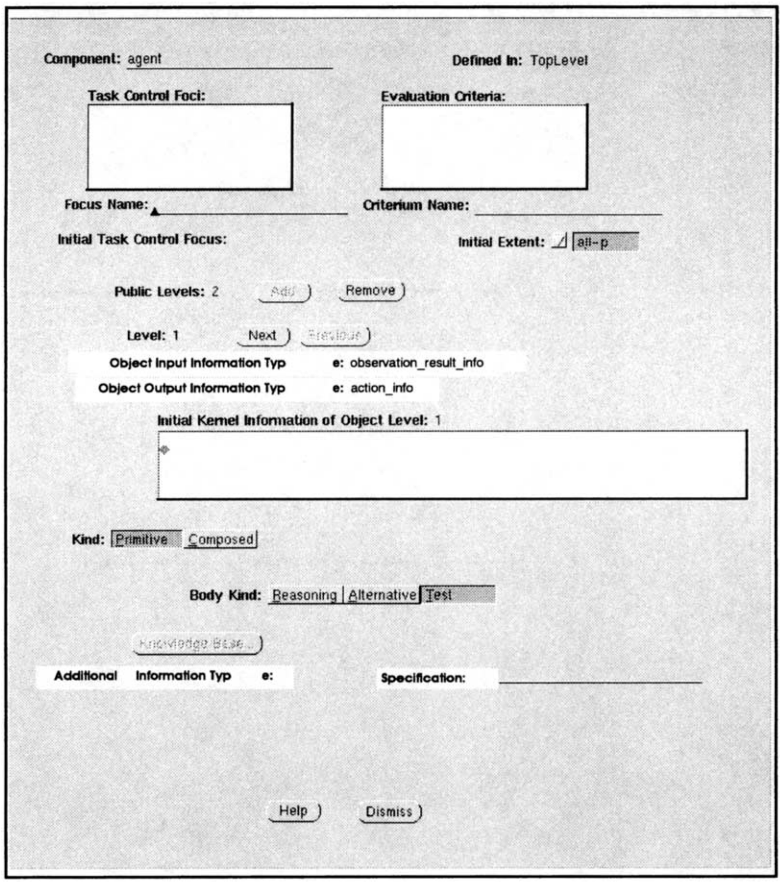
7. Supporting software environment

The compositional design method DESIRE is supported by a software environment. The DESIRE software environment includes a number of facilities. Graphical design tools support specification of conceptual and detailed design of processes and knowledge at different abstraction levels. A detailed design in DESIRE provides enough detail to be able to develop an operational implementation automatically in any desired environment. An implementation generator supports prototype generation of both partially and fully specified models. The code generated by the



Saved primreac2.dtb

Fig. 16. Graphical design tool for process composition.



Component: agent

Defined In: TopLevel

Task Control Foci:

Evaluation Criteria:

Focus Name:

Criterium Name:

Initial Task Control Focus:

Initial Extent: all-p

Public Levels: 2 Add Remove

Level: 1 Next Previous

Object Input Information Type: observation_result_info

Object Output Information Type: action_info

Initial Kernel Information of Object Level: 1

Kind: [✓ Primitive] Composed

Body Kind: Reasoning | Alternative | [✓ Test]

Knowledge Base...

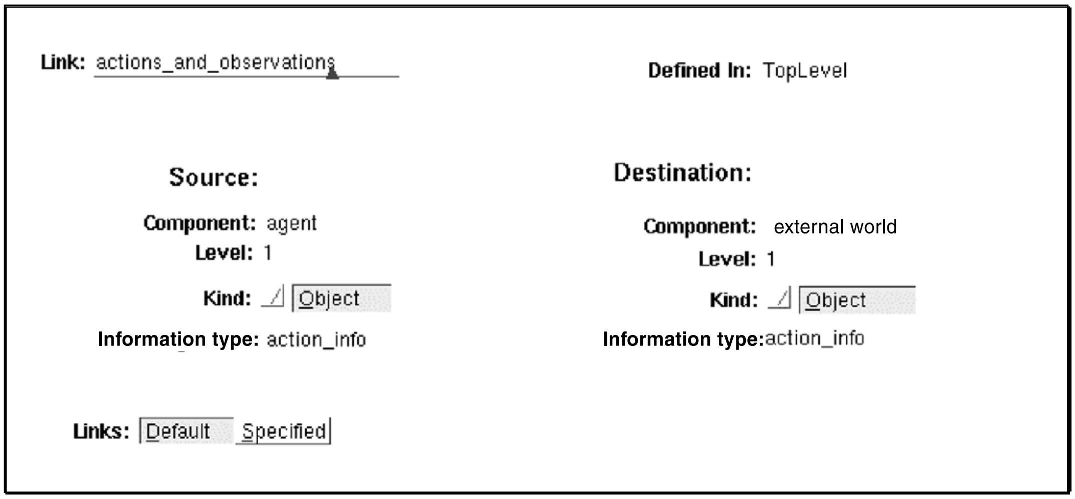
Additional Information Type:

Specification:

Help Dismiss

Fig. 17. Component editing window for a component.

implementation generator can be executed in an execution environment. Screenshots of interaction with the tools illustrate the support the tools provide. Fig. 16 shows the result of the creation (by a mouse click, and then filling the names) of two components Agent and the External World and two links between the components. The precise specifications of these components and links are created in interaction with the graphical editors to make the drawing, as shown in Figs. 17 and 18. Moreover, if within one of the components a compositional structure using subcomponents is


Link: actions_and_observations
Defined In: TopLevel

Source:
Component: agent
Level: 1
Kind: Object
Information type: action_info

Destination:
Component: external world
Level: 1
Kind: Object
Information type: action_info

Links: Default Specified

Fig. 18. Editor for information links.

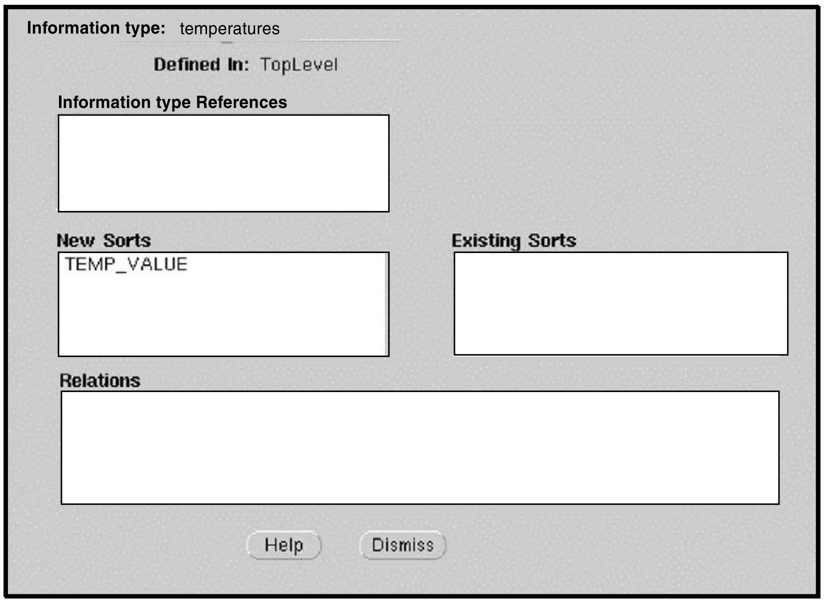

Information type: temperatures
Defined In: TopLevel
Information type References
New Sorts
TEMP_VALUE
Existing Sorts
Relations

Fig. 19. Editor for information types.

required, by a mouse click on this component a new drawing area can be opened, where again components can be introduced (zoom in).

Fig. 17 depicts the initial specification of the Agent component in which, for example, the input and output information types are defined. Fig. 18 shows the specification of an information link between the External World and the Agent. For example, the type of information to be exchanged, namely action_info, is specified in this window. Fig. 19 shows how information types are defined. The example information type temperatures requires a new sort TEMP_VALUE.

8. Discussion

The basic principles behind compositional multi-agent 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 operationalisation of these principles within a compositional development method for multi-agent systems is, however, a distinguishing element. Such a method can be supported by a (graphical) software environment in which all three levels of design are supported: from conceptual design to implementation. Libraries of both generic models and instantiated components, of which a few have been highlighted 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.

A number of approaches to conceptual-level specification of multi-agent systems have been recently proposed. On the one hand, general-purpose formal specification languages stemming from Software Engineering are applied to the specification of multi-agent systems (e.g., [33,38] for approaches using Z, resp. Z and CSP). A compositional development method such as DESIRE is committed to well-structured compositional designs that can be specified at a higher level of conceptualisation than in Z or VDM and, in particular, allows for specification in terms of knowledge bases, which especially for applications in information-intensive domains is an advantage. Moreover, designs can be implemented automatically using automated prototype generators. In [32] an approach to the composition of reactive system components is described. Specification of components is done on the basis of temporal logic. Two differences with our approach are the following. First, their approach is limited to reactive components. In our approach components are allowed to be non-reactive as well. Another difference is that in their case specification of the type of the composition of components is limited. In our case the task control specification forms the part of the composition specification where the dynamics of the composition is defined in a tailored manner, using temporal task control rules. This enables to specify, for each composition, precisely the type of composition that is required. This is also a difference with [33,38].

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, [30] describe a language on the one hand based on the BDI agent architecture [37], and on the other hand based on object-oriented design methods.

In [40] an agent is constructed from components using a central message board within the agent which manages the interaction between the agent's components and integrates the activity within the agent. Our approach is more general in the sense that a component-based architecture of an agent (e.g., the model GAM) need not commit to such a central message-board; if desired, it is one of the architectural possibilities. Moreover, components within DESIRE are more self-contained in the sense that they include knowledge bases and relate to specific inference procedures and settings. In contrast, in [40] components are quite heterogeneous; for example, a component can be just a knowledge base, which only gets its dynamic semantics if it is processed by another component. Another difference is that in [40] components are specified as a type of logic programs. It is not clear how declarative and/or procedural semantics of these programs are defined. For example, they allow component replacement as one of the steps in dynamics. This suggests dynamic semantics that are on the programming level; how to define such semantics on a conceptual level is far from trivial. In our approach semantics is defined on a conceptual design level based on traces of compositional states.

The Concurrent MetateM framework [21] is another modelling framework for multi-agent systems. A comparison is discussed for the structure of agents, inter-agent communication and meta-level reasoning (for a more extensive comparison, see [35]).

For the *structure of agents*, in DESIRE, the knowledge structures that are used in the knowledge bases and for the input and output interfaces of components are defined in terms of information types, in which sort hierarchies can be defined. Signatures define sets of ground atoms. An assignment of truth values *true*, *false* or *unknown* to atoms is called an information state. Every primitive component has an internal information state, and all input and output interfaces have information states. Information states evolve over time. Atoms are persistent in the sense that an atom in a certain information state is assigned to the same truth value as in the previous information state, unless its truth value has changed because of updating an information link.

Concurrent MetateM does not have information types, there is no predefined set of atoms and there are no sorts. The input and output interfaces of an object consist only of the names of predicates. Two valued logic is used with a closed world assumption, thus an information state is defined by the set of atoms that are true.

In a DESIRE specification of a multi-agent system, the agents are (usually) subcomponents of the top-level component that represents the whole (multi-agent) system, together with one or more components that represent the rest of the environment. A component that represents an agent can be a composed component: an agent task hierarchy is mapped into a hierarchy of components. All (sub-)components (and information links) have their own timescale.

In a Concurrent MetateM model, agents are modelled as objects that have no further structure: all its tasks are modelled with one set of rules. Every object has its own timescale.

The *communication* between agents in DESIRE is defined by the information links between them: communication is based on point-to-point or broadcast message passing. Communication between agents in Concurrent MetateM is done by broadcast message passing. When an object sends a message, it can be received by all other objects. On top of this, both multi-cast and point-to-point message passing can be defined.

In DESIRE, *meta-reasoning* is modelled by using separate components for the object and the meta-level. For example, one component can reason about the reasoning process and information state of another component. Two types of interaction between object- and meta-level are distinguished: upward reflection (from object- to meta-level) and downward reflection (from meta- to object-level). The knowledge structures used for meta-level reasoning are defined in terms of information types, standard meta-information type can automatically be generated.

For meta-reasoning in Concurrent MetateM, the logic MML has been developed. In MML, the domain over which terms range has been extended to incorporate the names of object-level formulae. Execution of temporal formulae can be controlled by executing them by a meta-interpreter. These meta-facilities have not been implemented yet.

The compositional approach to agent design in this paper has some aspects in common with object oriented design methods; e.g., [5,16,39]. However, there are differences as well. Examples of approaches to object-oriented agent specifications can be found in [4,29]. 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, [27] 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 behaviour, in addition to state. An object does not encapsulate *behaviour*: it has no control over the execution of methods – if an object x invokes a method m on an object y, then y has no control over whether m is executed or not – it just *is*. In this sense, object 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’), and that can initiate action (objects that can say ‘go’).

A difference between the compositional 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 composition relation is defined in a more specific manner: the static aspects by information links, and the dynamic aspects by (temporal) task control knowledge, according to a pre-specified format. A similarity is the (re)use of generic structures: generic models in DESIRE, and patterns (cf. [3,22]) in object-oriented design methods, although their functionality and compositionality are specified in different manners, as discussed above.

References

[1] M. Abadi, L. Lamport, Composing specifications, ACM Transactions on Programming Languages and Systems 15 (1) (1993) 73–132.

[2] O. Akihik, N. Yasuo, H. Yutaka, H. Masaonri, H. Shinichi, Plangent: an approach to making mobile agents intelligent, IEEE Internet Computing 1 (2) (1997).

[3] C. Alexander, A Pattern Language, Oxford University Press, Oxford, 1977.

[4] Y. Aridor, D.B. Lange, Agent design patterns: elements of agent application design, in: Proceedings of the Second Annual Conference on Autonomous Agents, Agents'98, ACM Press, New York, 1998, pp. 108–115.

[5] G. Booch, Object-Oriented Analysis and Design, second ed., Addison-Wesley, Reading, MA, 1994.

[6] F.M.T. Brazier, F. Cornelissen, R. Gustavsson, C.M. Jonker, O. Lindeberg, B. Polak, J. Treur, Compositional Design and Verification of a Multi-Agent System for One-to-Many Negotiation, in: Proceedings of the Third International Conference on Multi-Agent Systems, ICMAS'98, IEEE Computer Society Press, Silver Spring, MD, 1998, pp. 49–56.

[7] F.M.T. Brazier, B. Dunin-Keplicz, N.R. Jennings, J. Treur, V. Lesser, Formal specification of multi-agent systems: a real-world case, in: Proceedings of the First International Conference on Multi-Agent Systems, ICMAS'95, MIT Press, Cambridge, MA, 1995, pp. 25–32;
Extended version in: M. Huhns, M. Singh (Eds.), Formal Methods in Cooperative Information Systems: Multi-Agent Systems, International Journal of Cooperative Information Systems 6 (1997) 67–94 (special issue).

[8] F.M.T. Brazier, B. Dunin-Keplicz, J. Treur, L.C. Verbrugge, Modelling the internal behaviour of BDI-agents, in: J.-J.Ch. Meyer, P.Y. Schobbes (Eds.), Formal Models of Agents (Selected papers from final ModelAge Workshop), Lecture Notes in AI, vol. 1760, Springer, Berlin, 1999, pp. 36–56.

[9] F.M.T. Brazier, C.M. Jonker, J. Treur, Formalisation of a cooperation model based on joint intentions, in: J.P. Müller, M.J. Wooldridge, N.R. Jennings (Eds.), Intelligent Agents III (Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages, ATAL'96), Lecture Notes in AI, vol. 1193, Springer, Berlin, 1997, pp. 141–155;
Extended version in:, International Journal of Cooperative Information Systems 9 (2000) 171–207.

[10] F.M.T. Brazier, P.H.G. van Langen, Zs. Ruttkay, J. Treur, On formal specification of design tasks, in: J.S. Gero, F. Sudweeks (Eds.), Artificial Intelligence in Design '94, Kluwer Academic Publishers, Dordrecht, 1994, pp. 535–552.

[11] F.M.T. Brazier, P.H.G. van Langen, J. Treur, Strategic knowledge in compositional design models, in: J.S. Gero, F. Sudweeks (Eds.), Proceedings of the Fifth International Conference on Artificial Intelligence in Design, AID'98, Kluwer Academic Publishers, Dordrecht, 1998, pp. 129–147.

[12] F.M.T. Brazier, J. Treur, N.J.E. Wijngaards, The acquisition of a shared task model, in: N. Shadbolt, K. O'Hara, G. Schreiber (Eds.), Advances in Knowledge Acquisition, Proceedings of the 9th European Knowledge Acquisition Workshop, EKAW'96, Lecture Notes in AI, vol. 1076, Springer, Berlin, 1996, pp. 278–289.

[13] F.M.T. Brazier, J. Treur, N.J.E. Wijngaards, Modelling interaction with experts: the role of a shared task model, in: W. Wahlster (Ed.), Proceedings of the 12th European Conference on AI, ECAI'96, Wiley, Chichester, 1996, pp. 241–245.

[14] A.W. Brown (Ed.), Component-Based Software Engineering, IEEE Computer Society Press, Silver Spring, MD, 1996.

[15] C. Castelfranchi, F. Dignum, C.M. Jonker, J. Treur, Deliberative normative agents: principles and architecture, in: N.R. Jennings, Y. Lesperance (Eds.), Intelligent Agents VI. Proceedings of the Sixth International Workshop on Agent Theories, Architectures and Languages, ATAL'99, Lecture Notes in AI, vol. 1757, Springer, Berlin, 2000, pp. 364–378.

[16] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, P. Jeremaes, Object-Oriented Development: The FUSION method, Prentice-Hall International, Hempel Hempstead, England, 1994.

[17] F. Cornelissen, C.M. Jonker, J. Treur, Compositional verification of knowledge-based systems: a case study for diagnostic reasoning, in: E. Plaza, R. Benjamins (Eds.), Knowledge Acquisition, Modelling and Management, Proceedings of the 10th EKAW, Lecture Notes in AI, vol.1319, Springer, Berlin, 1997, pp. 65–80.

[18] A. Dardenne, A. van Lamsweerde, S. Fickas, Goal-directed requirements acquisition, Science in Computer Programming 20 (1993) 3–50.

[19] R. Darimont, A. van Lamsweerde, Formal refinement patterns for goal-driven requirements elaboration, Proceedings of the Fourth ACM Symposium on the Foundation of Software Engineering (FSE4) (1996) 179–190.

[20] E. Dubois, P. Du Bois, J.M. Zeippen, A formal requirements engineering method for real-time, concurrent, and distributed systems, in: Proceedings of the Real-Time Systems Conference, RTS'95, 1995.

[21] M. Fisher, Representing and executing agent-based systems, in: M. Wooldridge, N. Jennings (Eds.), Intelligent Agents – Proceedings of the International Workshop on Agent Theories, Architectures, and Languages, ATAL'94, Lecture Notes in Artificial Intelligence, vol. 890, Springer, Berlin, 1995.

[22] E.E. Gamma, R. Helm, R. Johnson, J. Vlissides, Desing Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley/Longman, Reading, MA/New York, 1994.
[23] D.E. Herlea, C.M. Jonker, J. Treur, N.J.E. Wijngaards, Specification of behavioural requirements within compositional multi-agent system design, in: F.J. Garijo, M. Boman (Eds.), Multi-Agent System Engineering, Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'99, Lecture Notes in AI, vol. 1647, Springer, Berlin, 1999, pp. 8–27.
[24] J. Hooman, Compositional verification of a distributed real-time arbitration protocol, Real-Time Systems 6 (1994) 173–206.
[25] J. Hopkins, Component primer, Communications of the ACM 43 (1) (2000) 27–30.
[26] N.R. Jennings, Controlling cooperative problem solving in industrial multi-agent systems using joint intentions, Artificial Intelligence Journal 74 (2) (1995).
[27] N.R. Jennings, M. Wooldridge, Applications of intelligent agents, in: N.R. Jennings, M. Wooldridge (Eds.), Agent Technology: Foundations, Applications, and Markets, Springer, Berlin, 1998, pp. 3–28.
[28] C.M. Jonker, J. Treur, Compositional verification of multi-agent systems: a formal analysis of pro-activeness and reactiveness, in: W.P. de Roever, H. Langmaack, A. Pnueli (Eds.), Proceedings of the International Workshop on Compositionality, COMPOS'97, Lecture Notes in Computer Science, vol. 1536, Springer, Berlin, 1998, pp. 350–380, Extended version to appear in: International Journal of Cooperative Information Systems, 2002.
[29] E.A. Kendall, P.V. Murali Krisna, C.V. Pathak, C.B. Suresh, in: Proceedings of the Second Annual Conference on Autonomous Agents, Agents'98, ACM press, New York, 1998.
[30] D. Kinny, M.P. Georgeff, A.S. Rao, A methodology and technique for systems of BDI agents, in: W. van der Velde, J.W. Perram (Eds.), Agents Breaking Away, Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'96, Lecture Notes in AI, vol. 1038, Springer, Berlin, 1996, pp. 56–71.
[31] G. Kontonya, I. Sommerville, Requirements Engineering: Processes and Techniques, John Wiley and Sons, New York, 1998.
[32] K. Lano, J. Bicarregui, T. Maibaum, J. Fiadeiro, Composition of reactive system components, in: Proceedings of the Workshop on Foundations of Component-Based Systems (FoCBS '97), Zurich, 1997.
[33] M. Luck, M. d'Inverno, A formal framework for agency and autonomy, in: V. Lesser (Ed.), Proceedings of the First International Conference on Multi-Agent Systems, ICMAS'95, AAAI Press, 1995, pp. 254–260.
[34] D.L. Martin, A.J. Cheyer, D.B. Moran, The open agent architecture: a framework for building distributed software systems, Applied Artificial Intelligence 13 (1999) 91–128.
[35] M. Mulder, J. Treur, M. Fisher, Agent modelling in MetateM and DESIRE, in: M.P. Singh, A.S. Rao, M.J. Wooldridge (Eds.), Intelligent Agents IV, Proceedings of the Fourth International Workshop on Agent Theories, Architectures and Languages, ATAL'97, Lecture Notes in AI, vol. 1365, Springer, Berlin, 1998, pp. 193–207.
[36] H.S. Nwana, D.T. Ndumu, L.C. Lee, J. Collis, ZEUS: a tool-kit and approach for building distributed multi-agent systems, in: Proceedings of the Third International Conference on Autonomous Agents '99, 1999, pp. 360–361.
[37] A.S. Rao, M.P. Georgeff, Modeling rational agents within a BDI architecture, in: R. Fikes, E. Sandewall (Eds.), Proceedings of the Second Conference on Knowledge Representation and Reasoning, Morgan Kaufman, Los Altos, CA, 1991, pp. 473–484.
[38] M.D. Rice, S.B. Seidman, Architectural issues in component-based software engineering, in: Proceedings of the Workshop on Foundations of Component-Based Systems (FoCBS '97), Zurich, 1997.
[39] J. Rumbaugh, M. Blaha, W. Pelerlani, F. Eddy, W. Lorensen, Object-Oriented Modelling and Design, Prentice-Hall, Eaglewoods Clifs, NJ, 1991.
[40] N. Skarmeas, K. Clark, Component-based agent construction, Department of Computer Science, Imperial College, University of London, 1999.
[41] M. Sparling, Lessons learned through six years of component-based development, Communications of the ACM 43 (10) (2000) 47–53.
[42] M. Wooldridge, N.R. Jennings, Agent theories, architectures, and languages: a survey, in: M. Wooldridge, N.R. Jennings (Eds.), Intelligent Agents, Proceedings of the First International Workshop on Agent Theories, Architectures and Languages, ATAL'94, Lecture Notes in AI, vol. 890, Springer, Berlin, 1995, pp. 1–39.