# Principles of Compositional Multi-Agent System Development

Frances M.T. Brazier, Catholijn M. Jonker, Jan Treur[1]

**(in: Proc. of the IFIP'98 Conference IT&KNOWS'98, J. Cuena (ed.), Chapman and Hall, 1998)**

***Abstract***
*A dedicated development method for multi-agent systems requires adequate means to describe the characteristics of agents and multi-agent systems. Compositional multi-agent system development is based on the principles process and knowledge abstraction, compositionality, reuse, specification and verification. Although the paper addreses these principles of compositional multi-agent system development from a generic perspective, some of the examples used to illustrate the notions discussed are taken from the compositional development method DESIRE.*

## 1. Introduction

Recent technological developments have considerably increased the amount of information exchanged between systems across the world. New developments in hardware and related protocols (parallel systems, the digital information superhighway), distributed operating systems and distributed databases have provided the means for industry to develop distributed, multi-agent industrial applications. It is important that the basic principles and lessons of software and knowledge engineering are applied to the development and deployment of such multi-agent systems. At present, the majority of existing agent applications are developed in an ad hoc fashion - following little or no rigorous design methodology and with limited a priori specification of the agents or of the system as a whole.

In many areas development methods have been developed in which a conceptual design of complex systems is specified before systems are implemented. Such specifications focus on the semantics of systems abstracting from implementation details, providing a basis for verification and validation of the functionality of the systems. A dedicated development method for multi-agent systems, requires adequate means to describe the characteristics of multi-agent systems, in particular, the control of the dynamics of (concurrent) agent reasoning behaviour and acting behaviour. To obtain genericity and flexibility, a commitment to one type of agent model is not sufficient. Generic models for a number of different types of agents should be available. One of the agent notions often used in the literature is the weak agent notion [16]. In Section 4.1 a generic compositional model for a weak agent is shown. Another well known agent model is the BDI-model [14]. A generic compositional model for this type of agent is shown in Section 4.3.

[1] Vrije Universiteit Amsterdam, Department of Mathematics and Computer Science, Artificial Intelligence Group, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands.
URL: http://www.cs.vu.nl/~{frances,jonker,treur}. Email: {frances,jonker,treur}@cs.vu.nl

A compositional multi-agent system development method based on the principles described in this paper can provide support to multi-agent system designers during the entire design process. In subsequent sections these principles are discussed: *process and knowledge abstraction*, *compositionality*, *reuse*, and *verification*. One specific compositional multi-agent development method is DESIRE (DEsign and Specification of Interacting REasoning components); cf. [3]. Although the paper addresses the principles of compositional multi-agent system development from a generic perspective, some of the examples used to illustrate the notions discussed are taken from this compositional development method.

## 2. Compositional Design of Multi-Agent Systems

In this section some general software and knowledge engineering principles behind compositional design are discussed.

### 2.1. The Design Process

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 involved, 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 is called *evolutionary development* of systems.

During a multi-agent system development process, the following descriptions can be distinguished (see *Figure* 1): problem description, conceptual design, detailed design, operational design, design rationale.

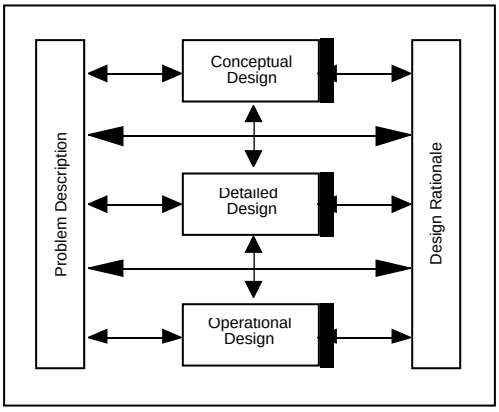


**Figure 1 Problem description, levels of design and design rationale**

The *problem description* includes the *requirements* imposed on the design: requirements often evolve during a design process. The *design rationale* specifies the choices made during design at each of the levels, and assumptions with respect to its use. The *conceptual design* includes conceptual models for individual agents, 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 (in an implementation independent manner) all aspects of a system's knowledge and behaviour. A detailed design is an adequate basis for *operational design*: prototype implementations can be generated automatically from the detailed design (see also Section 7).

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, required properties of models can be documented with the related assumptions (see Section 7 for more details), defining the limiting conditions under which the model has specific behaviour.

### 2.2. Compositionality of Processes and Knowledge

Compositionality is a general principle to structure a design, applicable both to processes and knowledge. Compositionality is a means to acquire *information and process hiding* within 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 (discussed in more detail in Sections 4 and 5).

*Processes* within a multi-agent system may be viewed as being 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 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. Thus 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.

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 often be combined from a number of more specific knowledge structures which, in turn, may again be composed of other even more specific knowledge structures. This entails different *levels of knowledge abstraction*.

These two notions of compositionality (of processes and of knowledge) are discussed in more detail in Section 3.

# 3. Conceptual and Detailed Design

In this section the different aspects of conceptual and detailed design are discussed in more detail: process composition in Section 3.1, knowledge composition in Section 3.2, and the relation between process composition and knowledge composition in Section 3.3.

## 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

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. 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 are distinguished. These processes (or sub-tasks) are then *delegated* to appropriate agents and the external world. The *multi-agent 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 controlling its own processes.

*Specification of a process*

The identified processes are modelled as *components*. For each process the *types of information* required as input, and resulting as output, are identified as well. This is modelled as *input and output interfaces* of the components.

*Specification of abstraction levels*

The identified levels of process abstraction 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, et cetera.

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 result in process hiding.

### 3.1.2. Composition Relation for Processes

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*

Knowledge of information exchange defines which types of information can be transferred between components and the *information links* by which this can be achieved.

*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 awake). Evaluation criteria, expressed in terms of the evaluation of the results (success or failure), provide a means to guide further processing. Task control knowledge can be specified to constrain the number of possible process traces that can be generated. Depending on the application, task control knowledge can be specified in different ways, varying from rather open approaches that entail almost no constraints on the behaviour (e.g., when all components and links are made awake), to a strictly prescribed sequence of activations of components and links. Task control is specified separately for each process abstraction level. The degree to which behaviour is constrained by task control can differ for these abstraction levels, and can differ between components within one abstraction level. For example, at the top level of a system, agents and the links between agents may be not constrained in their behaviour (which realizes their autonomy), and within an agent at a lower process abstraction level, task control may specify a fixed sequence of activation of components and links.

## 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 *Figure* 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.

*Knowledge bases*

A knowledge base defines a part of the knowledge that is used in one or more of the processes.

Knowledge bases use ontologies defined in information types. Which information types are used in a knowledge base defines a relation between information types and knowledge bases.

#### 3.2.2. Composition Relation for 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 that are distinguished, and results in information and knowledge hiding.

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

As shown in *Figure* 2 *compositionality of processes* and *compositionality of knowledge* are two different dimensions. Each process in a process composition uses knowledge structures. Which knowledge structures are used for which processes is defined by the relation between process composition and knowledge composition. The compositional knowledge structures can be related to one or more compositional process structures, where needed; the cells within the matrix depicted in *Figure* 2 define these relations. Note that not all cells need to be filled in this matrix. For example, in a special case where knowledge composition is completely dependent on the process composition the matrix in *Figure* 2 shows only a diagonal of filled cells.

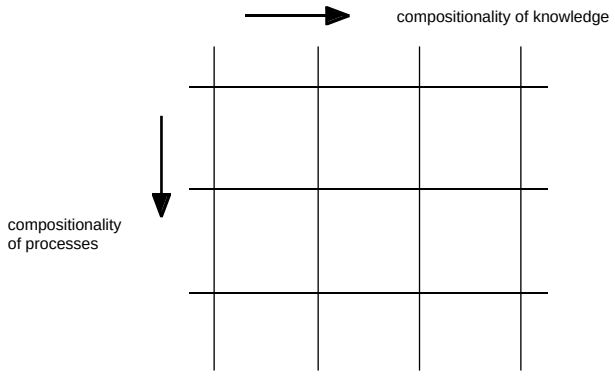


**Figure 2 Compositionality of processes and compositionality of knowledge**

# 4. 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). Generic 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 development 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 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. In this section a number of these types of generic models (available in DESIRE) are discussed.

## 4.1. Generic Agent Models

The 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. A BDI-agent shows behaviour on the basis of its beliefs, desires and intentions; cf. [14]. Which agent models are most applicable to a situation (possibly in combination) is determined during system design. Generic models for cognitive agents, co-operative agents and BDI-agents are briefly described below (these are some of the agent models available in DESIRE).

### 4.1.1. Generic Model for the Weak Agent Notion

The composition within an agent capable of reasoning, acting and communicating not only on the basis of incoming information from the external world, but also on the basis of information communicated by other agents, is shown in *Figure* 4.

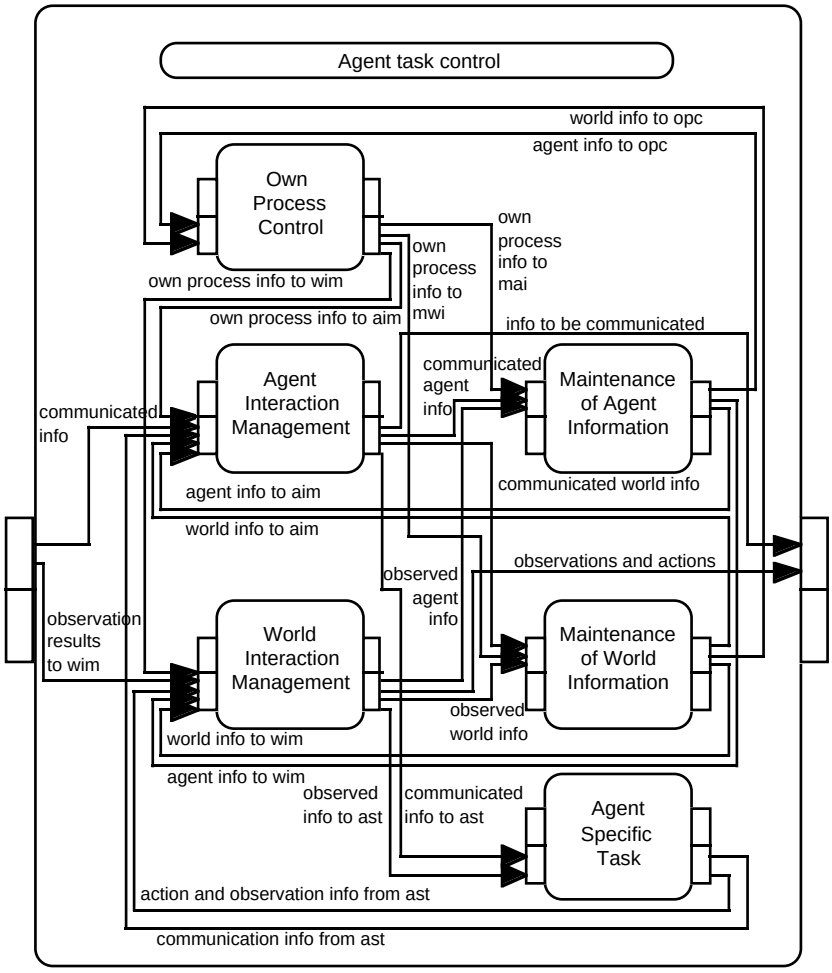


**Figure 4 Generic model for the weak agent notion**

This agent model supports the notion of weak agent, for which *autonomy*, *pro-activeness*, *reactiveness* and *social abilities* are distinguished as characteristics; cf. [16]. This type of agent model:

- 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). Note that the specific tasks for which an agent is designed are not explicitly specified in this generic model, nor is the interaction with maintain history.

### 4.1.2. Generic Model of a Co-operative Agent

If an agent explicitly reasons about co-operation with other agents, the model depicted in *Figure* 5 may be useful; cf. [10]. As a refinement of a Co-operation Management component, which is added as a seventh component to the model depicted in *Figure* 4, a co-operative agent has explicit knowledge on the basis of which co-operation with other agents is approached. For further levels of specialisation, see [5].

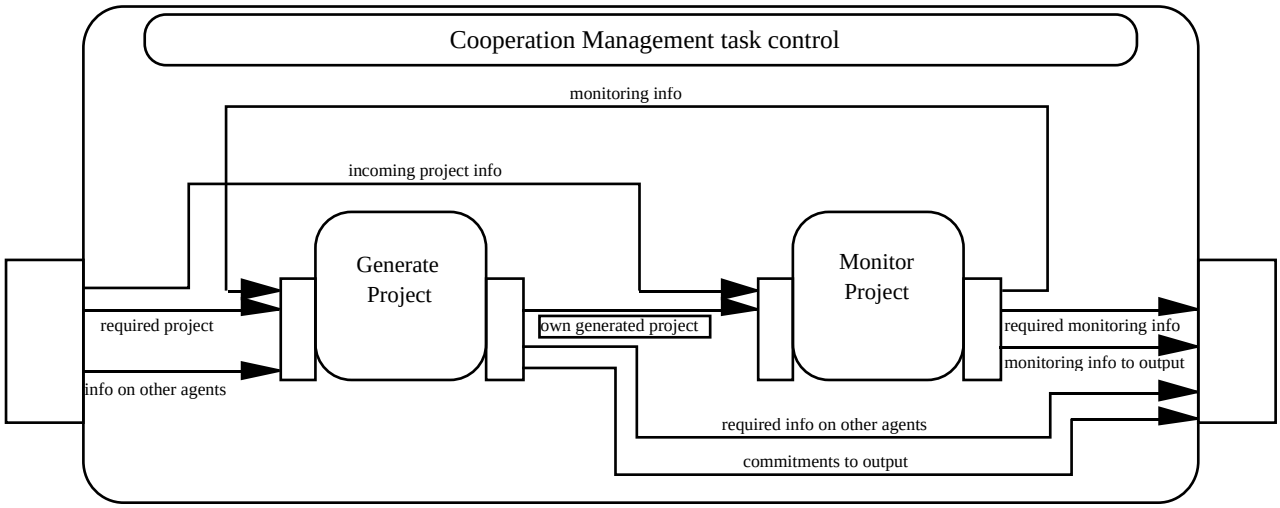


**Figure 5 Refinement of cooperation management in a generic cooperative agent model**

### 4.1.3. Generic Model of a BDI-Agent

An agent that bases its reasoning on its own beliefs, desires, commitments and intentions, a BDI-agent, is, in fact, a special type of cognitive agent. A BDI agent's own process control is based on its reasoning with and about beliefs, desires, commitments and intentions. The refinement of own process control is presented in *Figure* 6. For further levels of specialisation, see [4].

## 4.2. Generic Models of Tasks

The specific tasks for which agents are designed vary significantly. Likewise the variety of tasks for which generic task models have been developed is wide: diagnosis, design, process control, planning and scheduling are examples of tasks for which generic models are available. In this section compositional generic task models (developed in DESIRE) for the first two 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.

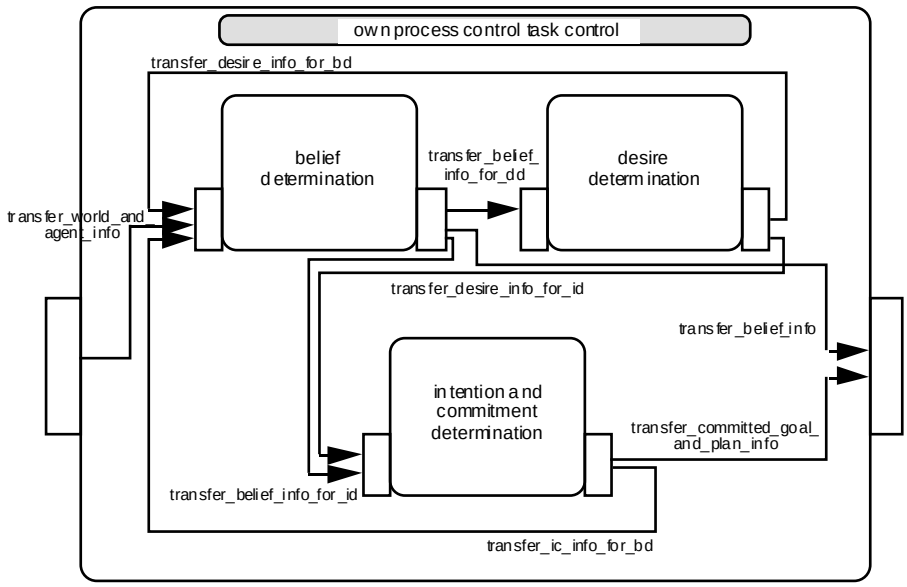


**Figure 6 Refinement of own process control in a generic BDI-agent model**

The tasks specifically related to *diagnosis* are included in the generic task model of diagnosis (for a top level composition, see *Figure* 7). This generic model 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 [7].

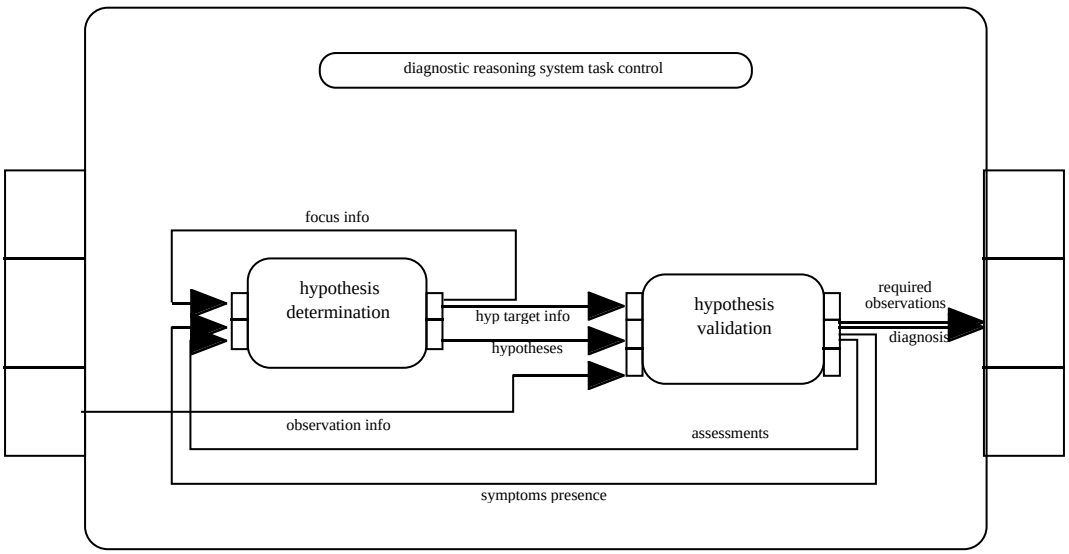


**Figure 7 Generic task model of diagnosis: top level**

A generic task model of *design* in which reasoning about three aspects of design are clearly distinguished is presented in [6]. The model distinguishes reasoning about requirements and preferences, from reasoning about the design object description, and reasoning about (co-ordination of) the overall design process

## 5. Verification

During the design process, the multi-agent system to be designed is often expressed in terms of graphical and textual language elements. For practical use of these language elements it is crucial that the designer has a conceptually clear picture of their meaning, both for her/his own understanding and for communication with others. Such a clear picture of the meaning of language elements can be obtained by describing *semantics* in an *informal*, semi-formal or formal manner. Defining and explaining *formal semantics* in addition is useful not only for the study of theoretical foundations and for developers of tools to support modelling, but also for verification and validation.

The requirements imposed on multi-agent systems designed to perform complex and interactive tasks sometimes are requirements on a final outcome, but more often are 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 important, it is vital to understand how system states change over time. In principle, the 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 is 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 will adhere 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. An example of a compositional verification method is described and applied to diagnostic reasoning, co-operative information gathering agents, and negotiating agents in [8], [11], and [2], 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, implies the properties that it 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 can be derived from properties of agents (one process abstraction level lower) and these agent properties, in turn, can be 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 the 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], [9], [15]:

- 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 [8] different properties of diagnostic reasoning and their logical relations have been formalised in this manner, and in [11] the same has been done for pro-activeness and reactiveness properties for co-operative information gathering agents. In [2] termination and successfulness properties for negotiation processes are analysed. In the references mentioned more details of the compositional verification method and its application can be found.

## 6. Supporting Software Environment

A compositional development method can be supported by a software environment. Such an environment includes tools to support system development during all phases of design. Graphical design tools, for example, can support specification of conceptual and detailed design of processes and knowledge at different abstraction levels. A detailed design is a solid basis to develop an operational implementation in any desired environment. An implementation generator can support prototype generation of both partially and fully specified models. The code generated by an implementation generator can be executed in an execution environment. The compositional development method DESIRE has such a supporting software environment.

## 7. Discussion

The basic principles behind compositional multi-agent system development 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., [13] for an approach using Z). 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 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, [12] describe a language on the one hand based on the BDI agent architecture [14], 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.

## References


[1] ABADI, M., LAMPORT, L., Composing Specifications, ACM Transactions on Programming Languages and Systems, Vol. 15, No. 1, 1993, pp. 73-132.

[2] BRAZIER, F.M.T., CORNELISSEN, F., GUSTAVSSON, R., JONKER, C.M., LINDEBERG, O., POLAK, B., TREUR, J., 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, 1998.

[3] BRAZIER, F.M.T., DUNIN-KEPLICZ, B., JENNINGS, N.R., TREUR, J. , Formal specification of Multi-Agent Systems: a real-world case. In: V. Lesser (ed.), Proc. of the First International Conference on Multi-Agent Systems, ICMAS'95, MIT Press, Cambridge, MA, 1995, pp. 25-32. Extended version in: International Journal of Cooperative Information Systems, M. Huhns, M. Singh (eds.), special issue on Formal Methods in Cooperative Information Systems: Multi-Agent Systems, vol. 6, 1997, pp. 67-94.

[4] BRAZIER, F.M.T., DUNIN-KEPLICZ, B., TREUR, J., and VERBRUGGE, L.C., Modelling the internal behaviour of BDI-agents. In: P.Y. Schobbens, A. Cesta (eds.), Proc. Third International Workshop on Formal Models of Agents, MODELAGE'97, Lecture Notes in AI, Springer Verlag. In press, 1998.

[5] BRAZIER, F.M.T., JONKER, C.M., TREUR, J., Formalisation of a cooperation model based on joint intentions. In: J.P. Müller, M.J. Wooldridge, N.R. Jennings (eds.), Intelligent Agents III (Proc. of the Third International Workshop on Agent Theories, Architectures and Languages, ATAL'96), Lecture Notes in AI, volume 1193, Springer Verlag, 1997, pp. 141-155.

[6] BRAZIER, F.M.T., LANGEN, P.H.G. van, RUTTKAY, Zs., TREUR, J., On formal specification of design tasks. In J.S. Gero & F. Sudweeks (eds.), Artificial Intelligence in Design '94, Dordrecht: Kluwer Academic Publishers , 1994, pp. 535–552.

[7] BRAZIER, F.M.T., TREUR, J., WIJNGAARDS, N.J.E., The Acquisition of a Shared Task Model. In: N. Shadbolt, K. O'Hara, G. Schreiber (eds.), Advances in Knowledge Acquisition, Proc. 9th European Knowledge Acquisition Workshop, EKAW'96, Lecture Notes in AI, vol. 1076, Springer Verlag, 1996, pp. 278-289.

[8] CORNELISSEN, F., JONKER, C.M., TREUR, J., Compositional Verification of Knowledge-based Systems: a Case Study for Diagnostic Reasoning. In: E. Plaza, R. Benjamins (eds.), Knowledge Acquisition, Modelling and Management, Proc. of the 10th EKAW, Lecture Notes in AI, vol. 1319, Springer Verlag, 1997, pp. 65-80.

[9] HOOMAN, J., Compositional Verification of a Distributed Real-Time Arbitration Protocol. Real-Time Systems, vol. 6, 1994, pp. 173-206.

[10] JENNINGS, N.R., Controlling Cooperative Problem Solving in Industrial Multi-Agent Systems using Joint Intentions, Artificial Intelligence Journal 74 (2), 1995.

[11] JONKER, C.M., TREUR, J., Compositional verification of multi-agent systems: a formal analysis of pro-activeness and reactiveness. In: [15], 1998.

[12] KINNY, D., GEORGEFF, M.P., RAO, A.S. , A Methodology and Technique for Systems of BDI Agents. In: W. van der Velde, J.W. Perram (eds.), Agents Breaking Away, Proc. 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'96, Lecture Notes in AI, vol. 1038, Springer Verlag, 1996, pp. 56-71.

[13] LUCK, M., D'INVERNO, M.. A formal framework for agency and autonomy. In: V. Lesser (ed.) Proc. of the first International Conference on Multi-Agent Systems, ICMAS'95, pp. 254-260, AAAI Press, 1995.

[14] RAO, A.S., GEORGEFF, M.P., Modeling rational agents within a BDI architecture. In: R. Fikes and E. Sandewall (eds.), Proceedings of the Second Conference on Knowledge Representation and Reasoning, Morgan Kaufman, 1991, pp. 473-484.

[15] ROEVER, W.P. de, LANGMAACK, H., PNUELI, A. (eds.), Proc. of the Int. Workshop on Compositionality, COMPOS'97, Springer Verlag. In press, 1998.

[16] WOOLDRIDGE, M., JENNINGS, N.R., Agent theories, architectures, and languages: a survey. In: [17], pp. 1-39.

[17] WOOLDRIDGE, M., JENNINGS, N.R. (eds.), Intelligent Agents, Proc. of the First International Workshop on Agent Theories, Architectures and Languages, ATAL'94, Lecture Notes in AI, vol. 890, Springer Verlag, 1995.