

An agent (re-)configuration model using templates for agent design

D.G.A. Mobach

in co-operation with H.M. Boonstra

16th January 2001

Abstract

In today's networked society, agents are becoming more and more prevalent. Automatic design of agents could help in supplying the right agents at the right time. This thesis presents a model for configuration-based (re-)design of agents, in the context of a central agent creation facility: the *Multi-Agent Factory*. The model is described using the compositional development framework DESIRE. Retrievable, partial agent descriptions named *Templates* are used in the multi-agent factory design-centre to configure agents. A prototype agent is configured by combining a number of templates. The creation process is examined by implementing the prototype agent using the Java object-oriented programming language.

Master's thesis of D.G.A. Mobach

Student number: 0630365

Email: dgmobach@cs.vu.nl

Intelligent Interactive Distributed Systems
Department of Artificial Intelligence
Division of Mathematics and Computer Science
Faculty of Sciences
Vrije Universiteit Amsterdam
The Netherlands

Supervisors:

Prof. Dr. F.M.T. Brazier

Dr. N.J.E. Wijngaards

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 1 |
| 2 | Relevant Literature | 1 |
| 2.1 | Multi-Agent Systems | 2 |
| 2.1.1 | Weak agents | 2 |
| 2.1.2 | Strong agents | 2 |
| 2.1.3 | Multi-agent system characteristics | 2 |
| 2.1.4 | Task delegation | 3 |
| 2.2 | Design Patterns and Frameworks | 3 |
| 2.3 | Case Based Reasoning | 4 |
| 3 | Multi-Agent Factory | 5 |
| 3.1 | Introduction to the multi-agent factory | 5 |
| 3.2 | Factory Overview | 6 |
| 3.3 | Use of a Multi-Agent Factory | 6 |
| 4 | Overview of relevant modeling paradigms | 7 |
| 4.1 | The DESIRE framework | 7 |
| 4.2 | The Java object-oriented programming language | 8 |
| 5 | Design Models | 8 |
| 5.1 | The process of design | 9 |
| 5.2 | Design Strategies | 10 |
| 5.3 | Generic Design Model | 11 |
| 5.3.1 | Subprocesses of design | 11 |
| 5.3.2 | Subprocesses of RQSM | 13 |
| 5.3.3 | Subprocesses of DODM | 14 |
| 5.4 | Model for Re-design of Compositional Systems | 14 |
| 5.4.1 | Refinements within RQS Modification | 15 |
| 5.4.2 | Refinements within RQSM History Maintenance | 16 |
| 5.4.3 | Refinements within DOD Modification | 16 |
| 5.4.4 | Refinements within DODM History Maintenance | 16 |
| 6 | The multi-agent factory design process | 17 |
| 6.1 | Templates & Properties | 17 |
| 6.2 | Template Retrieval | 17 |
| 6.3 | The multi-agent design-centre | 17 |
| 6.3.1 | DPC in the multi-agent factory | 18 |
| 6.3.2 | RQSM in the multi-agent factory | 18 |
| 6.3.3 | DODM in the Multi-Agent Factory | 22 |
| 6.4 | Example knowledge in the multi-agent design-centre | 25 |
| 6.5 | Assembly process | 31 |
| 7 | Agent configuration example | 32 |
| 7.1 | Trace | 32 |
| 7.2 | The resulting agent | 37 |
| 8 | Implementation | 37 |
| 9 | Conclusions and future work | 38 |
| 9.1 | Conclusions | 38 |
| 9.2 | Future work | 39 |
| | Appendices | 41 |
| A | Java class descriptions | 42 |
| B | Example trace | 52 |

1 Introduction

Computer and communication infrastructures are becoming more complex. The amount of information available within computer systems is becoming increasingly difficult to manage. Agents, capable of performing specific tasks with reasonable autonomy, can be of assistance by taking some of the work out of human hands. These agents, however, are complex, knowledge intensive, software entities. Designing agents for specific tasks is a difficult process. The task an agent has to perform has to be examined and the processes and knowledge used when performing the task have to be identified. To facilitate the job of designing an agent, generic agent models such as those in [Brazier et al., 2000c] and [Brazier et al., 2000b] have been developed which can be refined for specific agents with specific tasks in specific domains.

As the demand for agents with various abilities grows, a system capable of constructing agents on demand for specific tasks and domains is very useful. Agents only have to be constructed when they are needed, and may be destroyed when their job is done. Agents are tailored specifically for their tasks and domains, and when demands change or new tasks emerge, the existing agents are replaced by more capable ones, or the existing agents are remodeled to fit the new situation.

The goal of this thesis is to model the automatic generation of agents as a configuration-based design process, by using a central agent-creation facility: a multi-agent factory. This facility can configure agents from existing templates and modules on request.

The *Multi-Agent Factory* described in this thesis presents a means to design and construct agents for specific tasks and domains. The agents are designed using a configuration-based design process. Agent-templates are available through an intelligent *Template Retrieval* system. The template retrieval process used by the multi-agent factory is described in [Boonstra, 2000]. A template contains a conceptual design description using the DESIRE compositional specification method, properties describing the available functionality and how to apply the template in the design, and programming code which is used during the actual agent construction. Within a network environment, several multi-agent factories are present, as well as multiple template storage facilities. When constructing agents, multi-agent factories could communicate with each other and template storage facilities to exchange template-, design- and construction-knowledge.

The template-based design process within the multi-agent factory is the main topic of this thesis. The use of template information during an agent creation process is illustrated for an example Internet agent in a Java prototype.

The structure of this thesis is as follows: Section 2 discusses relevant literature on the main subjects addressed in this thesis: multi-agent systems and design knowledge re-use. Section 3 presents an overview of the multi-agent factory. The two modeling paradigms used in this thesis are described in Section 4: declarative modeling (DESIRE), and object-oriented modeling (Java). Section 5 discusses the models on which the design process of the multi-agent factory is based. The multi-agent factory design process is described in Section 6, including a detailed trace describing part of an example agent-design process within the multi-agent factory. Section 7 presents a full trace of the example agent-design process at a less detailed level. Section 8 briefly discusses the example agent implementation. Conclusions and a discussion of future work are presented in Section 9. Appendix A contains Java class descriptions of the classes used by the prototype agent implementation. Appendix B contains an example trace of an activation cycle of the prototype agent. Appendix C contains the complete Java implementation of the prototype agent.

2 Relevant Literature

Re-usable, generic structures are of great help when designing complex objects, as these pre-fabricated elements provide previously developed and tested parts that can be incorporated into a new design. The use of these structures helps designers in recognizing the relevant concepts in the problem at hand, and also provides possible solutions which can be adapted by the designer to a solution for the specific problem at hand. Reusable structures are also useful to convey design expertise by designers to other designers, who can adapt and extend the knowledge by applying their experiences to it. Ideally, an artefact design could eventually

be designed by composing a number of generic reusable elements in a particular way, only leaving small, domain-specific elements open, which can then be filled in by a designer.

In this section, first an introduction is given on the topic of Multi-Agent Systems. Subsections 2.2 and 2.3 discuss two different forms of design by reuse: design patterns and case based reasoning.

2.1 Multi-Agent Systems

The term *Agent* is widely used to describe all sorts of applications. This leads to problems when trying to define exactly what an agent is. Two notions of agents have been introduced to classify agents: a *weak* notion and a *strong* notion of agency.

2.1.1 Weak agents

For an agent to be classified as a weak agent, it must exhibit four types of behaviour [Wooldridge and Jennings, 1995]:

- Autonomous behaviour: An agent has some form of control over its own processes and is not directed in any way by outside parties.
- Social behaviour: An agent is able to communicate with other agents (including humans).
- Responsive behaviour: An agent is able to react to changes in its environment.
- Pro-active behaviour: An agent is able to set goals and can take the initiative to pursue these goals.

2.1.2 Strong agents

The stronger notion of agency [Dennett, 1987] demands that agents possess more human-like characteristics, such as beliefs, intentions and desires. Although these seem very high-level characteristics, they can be used effectively for designing agents. Models for such *Informational and Motivational* agents are described in [Brazier et al., 2000d].

2.1.3 Multi-agent system characteristics

An advantage of using agents as a basis of a distributed system is that each agent can independently function within the system, and use its knowledge to solve problems, possibly in co-operation with other agents within the system. The agent population within a multi-agent system is heterogeneous: each agent has its own view of the system, and each agent has different capabilities. The fact that each agent has control of its own processes means that there is no need for extensive, centralistic global system control. The use of independent components also contributes to the overall robustness of the multi-agent system. Problematic agents are easily replaced by others, and the system is easily extended with new functionality by adding new agents. Other characteristics include that data within the system is decentralized and computation is asynchronous.

Some problems encountered when designing multi-agent systems include:

- How to determine a useful division of problems within a system and assign agents to them.
- How to enable useful communication between the agents within a system.
- How to ensure that the independent agents work together in a useful manner and contribute to the overall success of the system.

Recently, agent based systems have begun to appear in a wide range of applications, ranging from Air Traffic Control systems to Entertainment systems. The article [Jennings et al., 1998] gives a good overview of the diversity of application domains.

2.1.4 Task delegation

To assign tasks to agents within a multi-agent system, the main task of the system can be decomposed into simpler subtasks, resulting in a task hierarchy. Each agent is then assigned a subset of these simpler subtasks. In the example displayed in figure 1, task **T** is divided into three subtasks, **sT1**, **sT2** and **sT3**. Due to the complexity of these subtasks, subtask **sT1** is assigned to agent **B**, **sT2** is assigned to agent **C** and **sT3** is assigned to agent **A**. In order for task **T** to be completed, all agents must succeed in their assigned tasks, possibly in co-operation with each other. This assignment of (sub)tasks to agents within a multi-agent system is called task delegation.

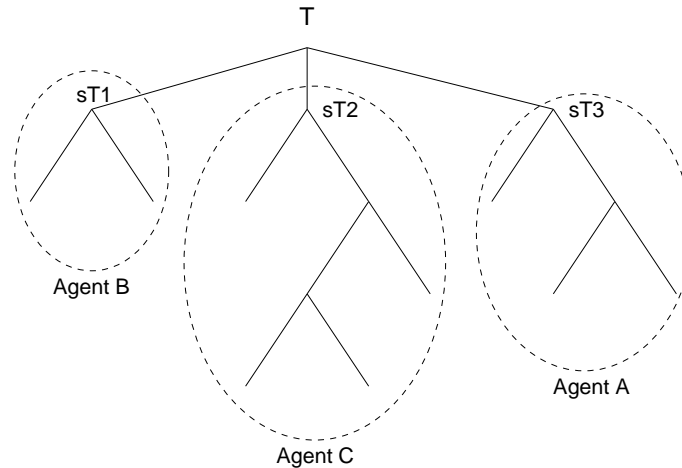


Figure 1. Assignment of subtasks to agents: task delegation.

2.2 Design Patterns and Frameworks

This section discusses design patterns in the domain of software development. In the 1960s, the architect Christopher Alexander introduced the term 'patterns' to indicate recurring themes in architecture [Alexander, 1964]. Over time, patterns have been recognized to be useful by the software engineering community. The role of patterns in software development is to provide reusable documentation on solutions to recurring problems encountered in software design. The term 'software pattern' can refer to both the solution itself or a description of how to apply the solution. [Gabriel, 1996] presents the following definition for a pattern, which can also be found on [pat, 2000]:

Each pattern is a three-part rule, which expresses a relation between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain software configuration which allows these forces to resolve themselves.

The *design patterns* introduced by the 'Gang of Four' in [Gamma et al., 1995] are most popular, although patterns have been used to describe problems occurring in all aspects of the software design process, such as *development organization*, *project planning* and *software processes*. The design patterns described in [Gamma et al., 1995] are often further decomposed into *architectural patterns*, *design patterns* and *implementation patterns*:

- **Architectural patterns:** Contain descriptions of characteristics of software systems as a whole, such as system properties and overall configuration.
- **Design patterns:** Contain descriptions of the components within the software system and their relationships.
- **Implementation patterns:** Contain descriptions of system components at the programming-level.

Within the pattern community it is generally agreed upon that a pattern should contain at least the following elements:

- **Name:** Needed to be able to refer to the pattern.

- **Problem:** The problem to which the pattern applies, in terms of goals and objectives of a pattern.
- **Context:** A description of the situation in which a problem occurs.
- **Forces, or tradeoffs:** The details of a problem are carefully examined revealing the possible interactions and/or conflicts between the elements of the solution and also with the goals of the pattern.
- **Solution:** An elaborate description of the structure and behaviour of the solution. The elements of the solution and interactions between these elements are examined, as well as possible problems that could be encountered when implementing the solution.
- **Examples:** Sample problems to which a pattern is applied. Using these examples it can be shown how a pattern can be used in specific contexts.
- **Resulting Context:** A description of the consequences of applying the pattern. An analysis is made of the way the context is altered by the pattern, and what problems may appear in the context after pattern application.
- **Rationale:** A description of the motivations behind the pattern itself. The choices and the reasons behind the choices made when constructing the pattern are examined.
- **Related Patterns:** Other patterns which are related to the pattern or the context the pattern is applicable to, are described, such as patterns which could be useful after application of this pattern, or possible alternative patterns.
- **Known Uses:** Real world applications of the pattern, to be able to verify the usefulness of the pattern in specific contexts.

The elements described here make up the *canonical form* [Buschmann et al., 1996] of patterns. Other, but similar pattern formats, such as the *GoF format* [Gamma et al., 1995], contain the same or similar elements.

Related patterns are grouped into so called *pattern languages*. A pattern language reveals the connections between the described patterns to present a vocabulary to better understand large problems and the patterns that can be used to solve those problem.

While design patterns contain mostly informal descriptions on how to construct software solutions, frameworks are actual partial solutions built by *using* patterns. From this point of view, design patterns can be said to be more generic than frameworks, as frameworks are defined for a particular application domain, while design patterns describe, at a higher-level, ways to solve recurring design problems. Frameworks are *partial* solutions, they contain *plug-points* enabling the framework to be extended and reused in different circumstances, and to be composed with other frameworks. The templates used in the multi-agent factory can be compared to frameworks, in the sense that templates are also partial solutions. Templates however, contain both conceptual descriptions *and* specific implementations of solutions.

2.3 Case Based Reasoning

This sections discusses Case-Based Reasoning (CBR) as described in [Kolodner, 1993] and in [Watson and Marir, 1994]. CBR is believed to have made its entry in 1977 in the work of Schank and Abelson [Schank and Abelson, 1977] about the notion of scripts. Scripts allow the recording of general knowledge about expected/required behaviour in specific situations. Kolodner defines a case as:

'A contextualized piece of knowledge representing an experience that teaches a lesson fundamental to achieving the goals of the reasoner.'

A case consists of a *problem description*, a *problem solution* and an *outcome*. The problem description describes the context of the experience. The problem solution describe the experience and the outcome describes the result of the experience (the lesson).

To allow use of the cases they must be indexed and stored in order to allow retrieval. Indexing is important because retrieval uses the index to asses similarity between a given situation and stored cases. An index should be predictive, it should address the purposes for which the case will be used, it should be abstract enough in order to be useful, and concrete enough to be recognizable.

Because two situations are never the same (unless it is the same situation) cases often have to be adapted in order to apply them to the current situation. The more specific the case the less adaption is needed to apply the case. Case based reasoning thus not relies on generic knowledge but on a combination between (generic) norms and (specific) deviations from these norms.

Newly taught lessons can be made into cases by adding context information. Adding new cases to the case based reasoner extends the strength of the reasoner because more specific knowledge becomes available. The strength of a specific case based reasoner depends on:

1. the experiences it has had,
2. its ability to understand new situations in terms of those old experiences,
3. its ability to adapt,
4. its adeptness at evaluation and repair,
5. its ability to integrate new experiences into its memory appropriately.

The following similarities exist between processes of CBR and processes of a Multi-Agent Factory:

- Both CBR and the MAF use proven methods in solving problems.
- Both CBR and the MAF rely on the assumption that situations recur.
- When new information becomes available during reasoning both CBR and the MAF can use this information to retrieve new (and hopefully better) cases / templates.
- Learning can in both CBR and the MAF be implemented by adding new cases / templates.
- Both CBR and the MAF can only function properly if they have a representative set of cases / templates for a specific domain.
- Both CBR and the MAF rely strongly on their retrieval capabilities.

3 Multi-Agent Factory

This section contains an introduction to the multi-agent factory concept and a global overview of the multi-agent factory, describing the five elements distinguished within the multi-agent factory.

3.1 Introduction to the multi-agent factory

The goal of this thesis is to describe a model for a facility capable of creating agents on demand. The multi-agent factory is such a facility and provides functionality by which agents can be designed and built according to given specifications. The agents are designed using a template based configuration approach. In this approach previously designed parts of agents are used and connected together, to create an agent design. The created design is subsequently used to build the operational description of the agent, using the code-objects located in the used templates. The template information employed in the agent creation process consists of conceptual descriptions which are used during the design process and code objects related to the templates which are used during the construction process.

A useful multi-agent factory capable of automatically generating agents should adhere to the following desiderata:

- The multi-agent factory is able to create an agent according to the specifications of its desired functionality, even if the specifications are vague or incomplete. The design process should use a minimalistic approach when designing agents, meaning that the functionality of the final agent design should have as little unrequested, additional functionality as possible.
- The multi-agent factory should be able to give an evaluation of the degree of success of the agent creation process.

- The multi-agent factory should be able to interact with other multi-agent factories and template repositories, possibly to exchange design knowledge and template information.
- Successful agent designs should be stored in some way, to prevent unnecessary 're-inventing' of agent designs.
- The conceptual design descriptions of agents produced by the design-centre should be usable for the assembly process within the multi-agent factory to assemble an operational agent description corresponding to the conceptual description.

3.2 Factory Overview

Five processes can be distinguished within the multi-agent factory:

- **Factory Management:**
Analogous with a 'real-world' factory, the factory management is responsible for guiding the overall factory process. For example, decisions have to be made concerning the way in which available resources are spent to satisfy clients. Also, to make sure the factory continues to be useful in the future, decisions have to be made concerning the acquisition of new knowledge.
- **Account Management:**
This process is responsible for managing factory-client interaction. This process has to ensure that the information given by the client is translated to information which can be used to design and construct the agents. The account management process is also responsible for making sure that the clients are really who they say they are, and that the transaction between the factory and the client cannot be falsely interpreted.
- **(Multi-)Agent Design-Centre:**
The design process is responsible for creating the conceptual design description of the agents that need to be constructed. Within the multi-agent factory, agents are designed using templates. These templates contain (partial) conceptual descriptions of agents.
- **Assembly:**
Agent design descriptions are used to create the actual operational agents by the assembly process. This process also uses the design description delivered by the design process to retrieve the (Java) code from the templates used within the design.
- **Template Retrieval:**
The template retrieval process is responsible for searching template repositories for suitable templates. Other processes can issue queries to the template retrieval process, indicating the type/functionality they require from templates.

3.3 Use of a Multi-Agent Factory

The five processes described above cooperate to achieve the main task of a multi-agent factory: creating agents on demand. The use of a multi-agent factory reduces an agent creation task to a task for which a user only needs to specify the required functionality of the agent(s), and set some variables which influence the multi-agent factory process (such as a time limit for the agent creation process, or a size limit for the agents that are to be created). The automatic agent creation process allows for novice users to create and use powerful agents, without having to acquire extensive knowledge on how to design and build agents, or without turning to third party agent developers. More advanced users could also benefit from an agent creation facility, especially when the processes within the multi-agent factory could be directed in a flexible manner.

Examples of multi-agent factory use:

- Example 1: new agent creation.
A user expresses the need for an agent which is capable of translating **SQL** into **QUEL**. Because no translating agents are currently present, the multi-agent factory decides to design a new translating agent.

- Example 2: agent modification.
An Internet agent which is currently only capable of **HTTP** communication expresses the need for **Gopher** communication, to be able to access other regions of the Internet. The multi-agent factory modifies the agent by adding the elements necessary for Gopher communication.
- Example 3: agent cloning.
An agent notices that it is not capable of performing a certain task by itself. The agent issues a request to a multi-agent factory indicating that it needs several clones of itself.

4 Overview of relevant modeling paradigms

This section describes the two modeling paradigms used in this thesis. The declarative modeling paradigm (DESIRE) is used to describe the multi-agent factory design process and is also used to describe the example agent. The object-oriented paradigm (Java) is used to implement the example agent.

4.1 The DESIRE framework

The modeling paradigm used to create the models used in this thesis is the compositional development method DESIRE (DESIGN and SPECIFICATION of INTERACTING REASONING components) [Brazier et al., 1997]. DESIRE allows for the independent modeling of processes within a task and knowledge used by the task. Using a component/subcomponent representation, tasks and subtasks can be identified and placed within a task hierarchy. Figure 2a displays the task hierarchy for a diagnostic reasoning task. The different levels within this structure define the different abstraction levels in the design.

The modeling of the task structure also includes modeling of task control. In DESIRE, each component contains some form of task control knowledge. Primitive (non-composed) components contain *task control* knowledge in the form of goals to achieve and the extent in which to pursue these goals. In addition to goals and extents, composed components also contain *task control* knowledge concerning activation of their subcomponents, and about the information flow between subcomponents and between subcomponents and the component itself.

Figure 2b displays an example of process composition for a diagnostic reasoning task. Within the diagnostic reasoning task two subtasks can be recognized: Determining hypotheses about the state of the object being diagnosed and validating these hypotheses to see if they are correct. The task DIAGNOSTIC REASONING is modeled as a composed component containing two subcomponents: HYPOTHESIS DETERMINATION and HYPOTHESIS VALIDATION. The subcomponents could in turn be composed components, containing subcomponents that represent subprocesses of HYPOTHESIS DETERMINATION and HYPOTHESIS VALIDATION. The figure also shows examples of information links: the information link *hypotheses* is used to transfer information on generated hypotheses from the output interface of HYPOTHESIS DETERMINATION to the input interface of HYPOTHESIS VALIDATION. The information link *validation results* is used to transfer information on validations of the generated hypotheses from the output interface of HYPOTHESIS VALIDATION to the input interface of HYPOTHESIS DETERMINATION.

Information exchange is explicitly modeled using information links. Components explicitly state which information is present on their input and output interfaces, and information links can be specifically defined to transfer specific information.

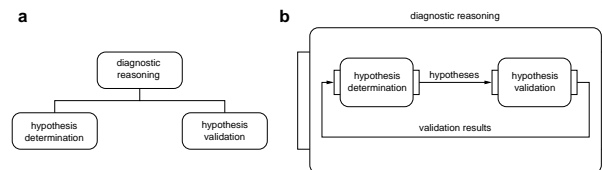


Figure 2. Task hierarchy / process composition.

The modeling of the knowledge used within a task is achieved using knowledge structures: *information types* and *knowledge bases*. The language used to specify these structures is order-sorted predicate logic.

Information types allow for structured specification of the information used by different processes within the task. Information types can contain more specific information types. Within these information types objects or terms and relations or functions on these objects or terms can be described. Information types can also contain meta-descriptions of other information types, allowing for the specification of knowledge at different abstraction levels. Knowledge bases define the knowledge used by processes. Knowledge bases can contain knowledge elements: *facts* and *rules*. Information types define the ontologies that can be used by knowledge bases.

4.2 The Java object-oriented programming language

The Java object-oriented programming language [Flanagan, 1999], introduced in 1995, has become a widely used programming language. This section describes the most important features that are responsible for its popularity and usefulness.

First, a distinction has to be made between three elements that make up Java: the programming language itself, the Java Virtual Machine and the Java Platform.

The Java language is an object-oriented language with a C-like syntax. One of the goals the designers had in mind was to make the language powerful, but usable. Compared to for instance C++, the Java language is more accessible and easier to use.

The Java Virtual Machine gives the Java language some of its power. The JVM is responsible for interpreting the compiled java code and executing it. JVMs allow for portability (usable on various system architectures) of the Java code, as holds for any platform for which a JVM is written. Although the first JVMs were very slow, the performance since then has improved much and new techniques such as *just-in-time-compilation* also contribute to the increase in speed and efficiency of code execution.

The Java Platform is the set of pre-defined class packages that provide implementations for much used functions such as networking, security, data structures, graphics and input/output. These packages also attribute to the portability of the language, as developers no longer have to rely on platform specific implementations of Application Program Interfaces (APIs).

Java is an object-oriented language, therefore the most important concept within the Java language is the *class*. A class consists of *fields* and *methods*, named *members* of the class. Instantiated classes are named objects, and objects are used in Java programs. When an object is created using a class, in general, the methods and fields of the class become available in the object, and can be used to manipulate/retrieve the state of the object in the program.

The major advantages of using Java when writing applications include:

- **Portability:** a program written on a specific platform is not limited to that platform. Keeping in mind the number of different platforms that are currently used and the increasing amount of connectivity between them, writing a program once, and being able to run it on all other platforms can be very useful indeed.
- **Security:** Java was developed with security as a major requirement. The ability to execute a program securely ('sandboxing'), is highly desirable in today's networked society.
- **Network oriented:** the importance of computer networks was also recognized by the Java designers, and consequently the use of network resources has been made very easy.

The advantages Java has to offer over other languages make it very suitable when programming Internet applications. The notion of portability gives applications the potential of moving across networks and performing functions at various locations. The example agent prototype implementation described in this thesis was written in Java for these reasons.

5 Design Models

This section presents an introduction on the process of design and the use of design strategies, followed by a description of the two design models on which the multi-agent factory design process is based: the *Generic Design Model* and the *model for Re-design of Compositional Systems*.

5.1 The process of design

The Merriam-Webster online dictionary [mer, 2000] defines **designing** (the verb) as: *to create, fashion, execute, or construct according to plan*. A **design** (the noun) of an artefact is defined as: *'a plan or protocol for carrying out or accomplishing something'*. This description indicates that in order for a design of an artefact to be made, the desired function(s) have to be known that are required of the artefact that is to be designed. The activity of design is a very knowledge intensive process, because a great deal of reasoning is involved to turn requirements into an artefact design that satisfies them. For example, during a design process, a number of the initially specified requirements may have to be dropped. Designing an artefact is also a very domain-specific task. A person or system performing a design task should essentially be an expert on the domain in which the artefact is to be designed.

A person designing a table for example could encounter the problem that the required material the table is to be made of is not available in the quantities that are necessary to satisfy the requirement indicating the size of the table. The designer then has to choose to either drop one of the requirements and design a smaller table or choose another material. It can also occur that during a design process the requirements posed to the design process change. For example, in the table-design example it could occur that a new shipment of the desired building material arrives. The previously determined conflict between size and material then no longer holds, and it is possible to reinstate the previously dropped requirements. Consequently, the design description then has to be changed to satisfy the new requirements.

The example above illustrates that extensive knowledge is needed to create a design description. A *system* that is to be able to design needs to be provided with this knowledge if it is to succeed in its task. Viewed as a black box, the inputs and outputs of such a design system can be examined. Figure 3 shows three distinctive types of information flowing *to* and *from* the design system.



Figure 3. A design system and its inputs and outputs.

The information needed as input by the design system can be divided into three types:

- **process objectives:** this type of information contains the higher-level goals set for the design process as a whole. These goals are used by the design system to guide the process in a direction indicated by the objectives. For example, the information may contain statements about the maximum amount of time and money the design process may take to find a useful design. Other information may include objectives indicating the desired complexity of the design.
- **initial qualified requirements:** this type of information contains requirements indicating the desired functionality of the final design object, and qualifications on these requirements. If a design system is to be able to make intelligent decisions about dropping or maintaining certain requirements, qualifications over these requirements are necessary which describe which requirements are more important than others.
- **initial artefact description:** this type of information contains a possible initial description of the object to be (re-)designed. For example, when designing a table, an initial (partial) design could already be presented indicating that the table has four legs.

The information presented by the design system after the design process is completed is also divided into three types:

- **process evaluation:** this type of information contains evaluations of the design objectives. This includes information on which process objectives were met to which extent.
- **final qualified requirements:** this type of information contains the final set of qualified requirements and assessments of the final qualified requirements with respect to the initial qualified requirements.
- **final artefact description:** this type of information contains the final design description and assessments of the design object description with respect to the final qualified requirements.

5.2 Design Strategies

The design process can be viewed as a search process. From this point of view, designing an object requires a search in two search spaces. The goal of the design process is to *find* a design object description that satisfies a certain set of qualified requirements related to the requirements initially stated. One of the search spaces that has to be traversed in this process is the space containing all possible object design descriptions. The other search space is the space containing all possible qualified requirement sets.

A design problem can be said to have been solved if in the requirement qualification set space a satisfactory set of qualified requirements is found, and a design object description is available in the design object space that satisfies this requirement qualification set. Figure 4, taken from [Brazier et al., 1994] shows the two search spaces with the dashed lines indicating which requirement qualification sets and design object descriptions have been examined together. The solid arrows indicate a transition to either a new requirement qualification set or a new design object description.

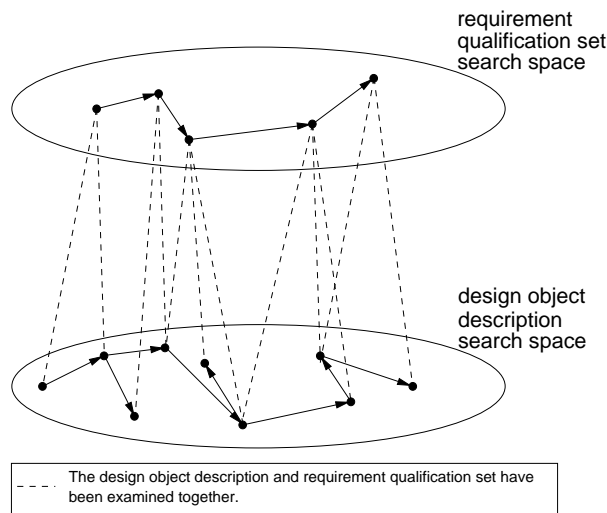


Figure 4. The design search spaces.

In the table-design example, the *design object description search space* consists of a search space containing possible table designs. Searching this space means examining different table designs to see if a matching and acceptable requirement qualification set can be found. The *requirement qualification set search space* consists of a search space containing possible sets of requirements that can be used to describe the desired properties of a table. Searching this space means examining these different requirement sets.

When designing a complex object, both the requirement qualification set space and the design object description space can be extremely large. In such a situation, simple search algorithms such as depth-first or breadth-first search will not be powerful enough to find a solution within reasonable time. In order for a design problem to be solved quicker and with good results, design *strategies* are needed. The use of strategies allows the design system to give direction to the search process. [Brazier et al., 1998b] discusses the different forms of strategic knowledge required when designing an artefact.

The choices made during the design process, such as: “*which modification must be made to the current design description ?*” or “*which requirements set must be used next ?*” are based on the design strategies used. The strategic knowledge used in the design process can be decomposed into three kinds of strategic knowledge:

- strategic knowledge used for determining which modification has to be made to the current *requirement qualification set*.
- strategic knowledge used for determining which modification has to be made to the current *design object description*.
- strategic knowledge used to guide the design process in general. This includes knowledge about which search space is to be traversed next and when to stop the design process.

This strategic knowledge is used in three levels of reasoning: Reasoning at the level of the requirements and their qualifications and relations, reasoning about design object descriptions, and reasoning about the design process as a whole. Each level uses the above mentioned strategies: strategies concerning the way in which requirements and their qualifications are focussed upon, strategies concerning the way in which design object descriptions are generated and modified, and strategies concerning the entire design process.

5.3 Generic Design Model

The generic model for design described in this section is a model created using the DESIRE framework briefly introduced in Section 4.1. A detailed description of the model can be found in [Brazier et al., 1998a]. A number of generic models have been designed using this framework. Two examples of generic agent models can be found in [Brazier et al., 2000c] and [Brazier et al., 2000b]. In short, a generic model is realized by determining the desired functionality, abstracting the generic elements from the more domain-specific elements and translating the generic elements to a task model [Brazier et al., 2000d]. These models can be used as starting points when designing agents. A generic model helps in identifying the types of knowledge and processes involved and presents a basis for further instantiation of the model for specific tasks. Once a generic model is defined, the model can be made suitable for a specific task in a specific domain by refinement. Refining the generic model with respect to adding domain knowledge is called instantiation. Modifying it with respect to defining additional task structures is called specialisation.

5.3.1 Subprocesses of design

This section describes those parts of the generic design model which are of importance when describing the multi-agent factory design-centre model.

The generic model for design distinguishes three direct subprocesses of the design process: DESIGN PROCESS CO-ORDINATION (DPC), DESIGN OBJECT DESCRIPTION MANIPULATION (DODM) and REQUIREMENT QUALIFICATION SET MANIPULATION (RQSM). These subprocesses and their information exchange are described in this section.

The contribution of each of the three subprocesses to the design process can be made clear by looking at figure 5. The figure distinguishes the three parts of the design process when viewed as a design process: one process which is responsible for searching the RQS solution space, another process which is responsible for searching the DOD search space, and a separate process responsible for deciding in what space the next search step should occur. In the generic model for design, RQSM is responsible for searching the RQS space, DODM for searching the DOD space and DPC is responsible for determining the following step in the overall design process.

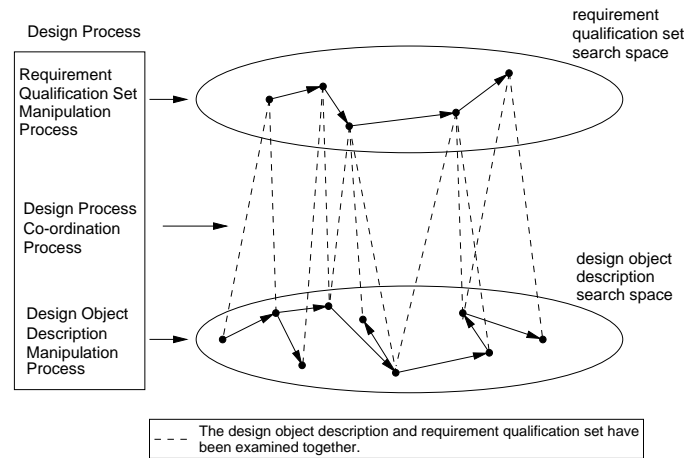


Figure 5. The role of the processes in the design process.

Each of the three processes reason about the design process on a different level of abstraction. DODM reasons on the lowest level of abstraction about the artefact to be designed, RQSM

reasons at a higher abstraction level about the requirements which the artefact must satisfy and DPC reasons at the highest level of abstraction about the design process itself. Figure 6 displays the top-level of the design process using the DESIRE compositional view. Processes are displayed as components with input and output interfaces. Information transferred between components is represented by information links. The names of the information links correspond with the names of the transferred information.

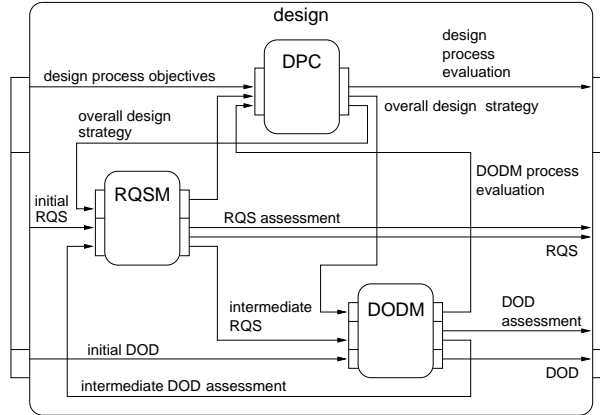


Figure 6. Top-level view of the design task process decomposition.

DESIGN PROCESS CO-ORDINATION

DPC represents the task that determines the overall design strategy. DPC receives three different types of information on its input:

- **design process objectives:** contains information regarding the design as a whole, such as time constraints. This information is provided by the agent using the design process.
- **rqsm process evaluation:** contains evaluations of the RQSM process, such as if the current overall design strategy has been fulfilled or not.
- **dodm process evaluation:** contains information similar to **rqsm process evaluation**.

DPC uses this information to determine the next step in the design process. DPC monitors the information coming from RQSM and DODM to determine if the overall design strategy is still successful. It also monitors if the design process is still satisfying the overall design objectives. When DPC has determined what the next course of action should be, it communicates this information to RQSM and DODM by providing them the current overall design strategy it has determined. The following types of information are on the output interface of the DPC process:

- **design process evaluation:** Contains evaluations of the design process as a whole, such as whether the design process has failed or if it is still running, and evaluations of the initial design process objectives.
- **overall design strategy to DODM:** Contains the information on the overall design strategy, such as a strategy indicating that the next modification step should be performed by RQSM.
- **overall design strategy to RQSM:** Contains information similar to **overall design strategy to DODM**

REQUIREMENT QUALIFICATION SET MANIPULATION

RQSM represents the task that is responsible for searching the space of requirement qualification sets to find the most suitable, acceptable set. RQSM bases its decisions on the overall design strategy received from DPC and the evaluation information it receives from DODM. RQSM has the following types of information on its input interface:

- **overall design strategy to rqsm:** contains overall design process strategy information.
- **initial RQS:** contains the initial requirement qualification set.
- **intermediate dod assessment:** contains evaluations of the current design object description with regard to the current qualified requirements set.

The initial RQS is the qualified requirement set that the design process as a whole must try to satisfy as well as possible. Within the limits indicated by the current overall design strategy, the initial RQS may be modified in order to find a set for which a design object description can be found. The following types of information are available on the output interface of RQSM:

- **RQSM process evaluation:** contains evaluations on the progress of the RQSM process, such as information indicating if RQSM was able to find a new RQS modification.
- **intermediate RQS:** contains the RQS currently being examined in the design process.
- **RQS assessment:** contains evaluations of the requirement qualification sets.
- **RQS:** contains the requirement qualification set itself.

DESIGN OBJECT DESCRIPTION MANIPULATION

DODM represents the task that is responsible for modifying the DOD. the main goal of DODM is to find a DOD that is valid with respect to the requirement qualification set currently under examination. The following information types can be distinguished on the input interface of the process:

- **overall design strategy to DODM:** contains overall design process strategy information.
- **intermediate RQS:** contains the requirement qualification set for which a valid DOD has to be constructed.
- **initial DOD:** contains an initial design object description, for example when re-designing an agent.

DODM produces four types of information on its output interface :

- **DODM process evaluation:** contains evaluations on the progress of the DODM process, such as information about the success of the modification process.
- **DOD assessment:** contains information on the satisfaction of qualified requirements by a DOD.
- **intermediate DOD assessment:** contains information similar to DOD ASSESSMENT, only this information is used as feedback for RQSM
- **DOD:** contains a design object description itself.

An activation cycle within the design process has the following form: Once the design task is activated, DPC is activated, determines the best course of action and communicates this to RQSM and DODM by means of an overall design strategy. According to the strategy either DODM or RQSM is activated, after which the other manipulation task is activated or control is returned to DPC. DPC then has determine what the next course of action should be.

Within RQSM and DODM, additional subprocesses are distinguished. The following subsections describe these processes and their role in the design process.

5.3.2 Subprocesses of RQSM

The generic model for design defines four subprocesses of the REQUIREMENT QUALIFICATION SET MANIPULATION process: RQS MODIFICATION, RQSM HISTORY MAINTENANCE, DEDUCTIVE RQS REFINEMENT and CURRENT RQS MAINTENANCE. The four subprocesses are shown in figure 7.

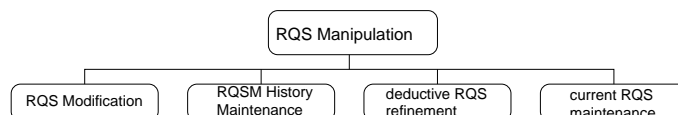


Figure 7. Subprocesses distinguished in RQSM according to the generic design model, shown as a process hierarchy.

RQS Modification is responsible for the modification of requirement qualification sets. The modifications are based on the qualified requirements currently under examination, the overall design strategy received from DPC, history information and evaluation information from DODM.

RQSM History Maintenance is responsible for storing and retrieving information about the rqsm process and the requirement qualification sets considered during design.

deductive RQS refinement is responsible for deducing properties of the requirements, such as properties indicating that conflicting requirements are present.

current RQS maintenance is responsible for storing the contents of the requirement qualification set currently under examination by the RQSM process.

5.3.3 Subprocesses of DODM

Within DODM, four subprocesses have been distinguished (see figure 8). These processes are similar to the four processes in RQSM:

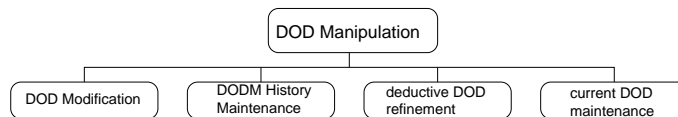


Figure 8. Subprocesses distinguished in DODM according to the generic design model.

DOD Modification is responsible for the modification of design object descriptions. The modifications are based on the overall design strategy, history information, the design object description currently under examination, and the current qualified requirement set under examination.

DODM History Maintenance is responsible for storing and retrieving information about the DODM process and the design object description considered during design.

deductive DOD refinement is responsible for deducing properties of design object descriptions.

current DOD maintenance is responsible for storing the contents of the design object description currently under examination by the DODM process.

5.4 Model for Re-design of Compositional Systems

The design centre of the multi-agent factory uses a number of elements from the model for re-design of compositional systems. This section describes those parts of the the model that are of importance for understanding the multi-agent factory design centre. A detailed description of the model can be found in [Wijngaards, 1999] and [Brazier et al., 2000a]. Elements from this model are useful for the multi-agent factory as it presents a model of a design process using compositional structures. The templates used in the multi-agent factory are based on the same compositional structures as are used for the design process in the model for re-design of compositional systems.

The model for re-design of compositional systems is based on the generic design model described in the previous section. The generic model has been refined for use in the domain of re-design of compositional systems. The design object descriptions within this model are representations of (multi-)agent systems. The requirement qualification sets in this model are sets containing desired properties of (multi-)agent systems. Examples of a (partial) DOD and a partial RQS are displayed in figure 9: the RQS in the example consists of four requirements stating desired capabilities of a diagnostic reasoning system. The DOD example shows elements used in a DOD to describe a design object.

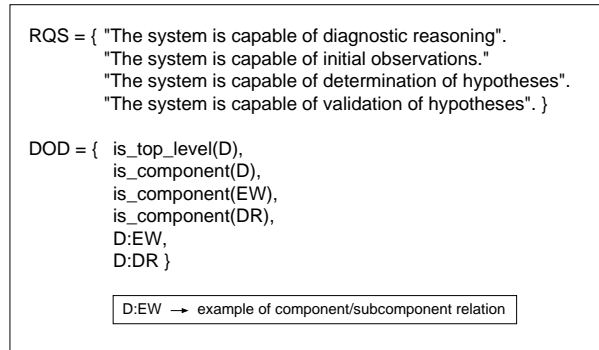


Figure 9. Examples of a design object description and a requirement qualification set from the model for re-design of compositional systems.

First, the processes in the generic design model are further *specialized*. After this, the knowledge structures present in the generic design model are further *instantiated*. Using figure 10, based on similar figures in [Wijngaards, 1999] and [Brazier et al., 2000a], the situation can be explained further: the generic design model has a certain level of refinement. The model for re-design of compositional system is based on the generic design model, but the process and knowledge structures have been further refined. The model for re-design of compositional systems is therefore less generic than the generic design model. It has more domain specific processes and structures. Note that the grid used in the figure does not imply a specifically defined distance between the models.

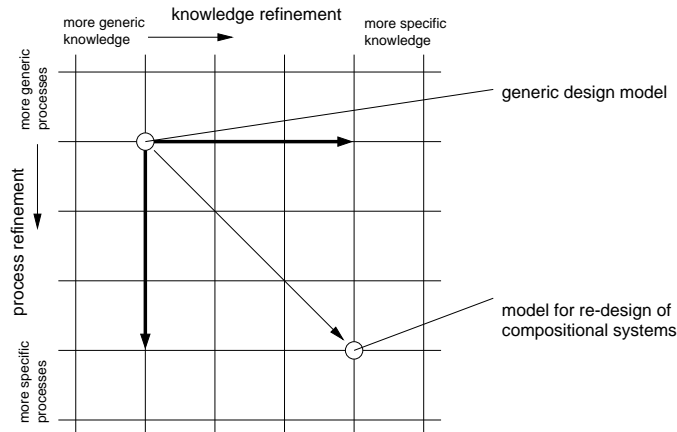


Figure 10. Refinement of Generic Design Model to a Model for Re-design of Compositional Systems.

5.4.1 Refinements within RQS Modification

This section describes the most important refinements of RQSM distinguished by the model for re-design of compositional systems. Within the RQSM MODIFICATION process, four subprocesses have been identified (see figure 11): RQS MODIFICATION PROCESS CO-ORDINATION, RQS VALIDATION, RQS MODIFICATION FOCUS IDENTIFICATION and RQS MODIFICATION DETERMINATION.

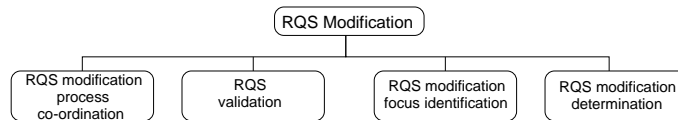


Figure 11. The four subprocesses of RQS Modification.

- RQS MODIFICATION PROCESS CO-ORDINATION is responsible for co-ordinating the RQS modification process. The decisions it makes are based on overall design strategy information, history information from RQSM HISTORY MAINTENANCE (see figure 7), and information on the progress of the modification process. The overall design strategy the co-ordination process receives is translated into a local modification strategy.

- RQS VALIDATION is responsible for validating the current RQS.
- RQS MODIFICATION FOCUS IDENTIFICATION is responsible for selecting the qualified requirements which need to be modified from the current RQS. This selection is based on the local strategy information issued by RQS MODIFICATION PROCESS CO-ORDINATION.
- RQS MODIFICATION DETERMINATION is responsible for the actual modifications that are made to the current RQS. The modifications made are based on the local strategy and on the current RQS focus.

5.4.2 Refinements within RQSM History Maintenance

Within RQSM HISTORY MAINTENANCE two subprocesses are identified (see figure 12): RQS HISTORY MAINTENANCE and RQS MODIFICATION STATE HISTORY MAINTENANCE

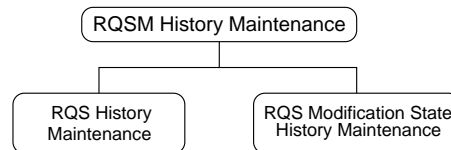


Figure 12. The two subprocesses of RQSM History Maintenance.

- RQS HISTORY MAINTENANCE is responsible for the managing design requirements history knowledge.
- RQS MODIFICATION STATE HISTORY MAINTENANCE is responsible for managing history knowledge regarding the modification process itself.

5.4.3 Refinements within DOD Modification

Within DODM, a similar distinction between processes is made. DOD MODIFICATION has four subprocesses: DOD MODIFICATION PROCESS CO-ORDINATION, DOD VALIDATION, DOD MODIFICATION FOCUS IDENTIFICATION and DOD MODIFICATION DETERMINATION (see figure 13).

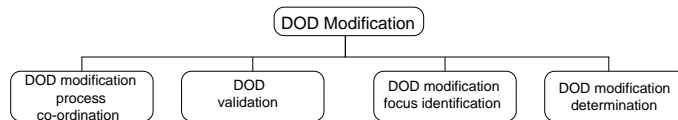


Figure 13. The four subprocesses of DOD Modification.

- DOD MODIFICATION PROCESS CO-ORDINATION is responsible for co-ordinating the DOD modification process, based on the overall design strategy (translated to a local strategy), history information from DODM HISTORY MAINTENANCE (see figure 8), and information on the progress of the modification process.
- DOD VALIDATION is responsible for validating the current DOD.
- DOD MODIFICATION FOCUS IDENTIFICATION is responsible for selecting the parts of the design object description which need to be modified. This selection is based on the local strategy information issued by DOD MODIFICATION PROCESS CO-ORDINATION.
- DOD MODIFICATION DETERMINATION is responsible for the actual modifications that are made to the current design object description. The modifications made are based on the local strategy and on the current DOD focus.

5.4.4 Refinements within DODM History Maintenance

DODM HISTORY MAINTENANCE is composed of three subprocesses (see figure 14): DOD HISTORY MAINTENANCE, DOD ASSESSMENTS & RQS HISTORY MAINTENANCE and DOD MODIFICATION STATE HISTORY MAINTENANCE.

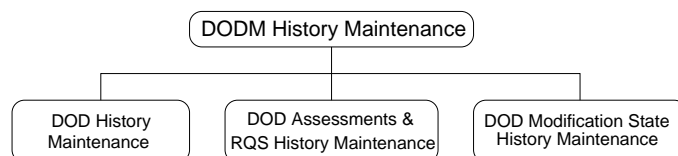


Figure 14. The three subprocesses of DODM History Maintenance.

- DOD HISTORY MAINTENANCE is responsible for the managing design object description history knowledge.
- DOD MODIFICATION STATE HISTORY MAINTENANCE is responsible for managing history knowledge regarding the modification process itself.
- DOD ASSESSMENTS & RQS HISTORY MAINTENANCE is responsible for managing DOD assessments and sets of qualified requirements.

6 The multi-agent factory design process

This section presents the model for the multi-agent factory design-centre. In Section 6.1 the elements used by the configuration-based design process to create agents are discussed: *Templates* and *Properties*. Section 6.2 presents the *Template Retrieval* process. The three main processes of the configuration-based design process are discussed in Section 6.3, with the emphasis on the use of templates and the template retrieval process within these three processes. A detailed example of a configuration cycle within the design process is presented in Section 6.4. Section 6.5 discusses the multi-agent factory assembly process, as this process uses the conceptual design object description created by the design process to create a detailed operational agent specification.

6.1 Templates & Properties

A *template* is a partial solution for a design problem. A template describes those parts of the solution that are relevant at a certain level in the design process without making commitments to details. These details can be choices of algorithm or detailed knowledge needed on a specific domain. This allows the re-design process to use gradual refinement and to explore new combinations of components. A template contains two descriptions: a conceptual description and an operational description. These descriptions may have open slots which allow the combination of templates. Combinations of templates, made by configuration based re-design, allow straightforward implementation of a completed design because a mapping can be made between the configuration of templates at a conceptual level and object oriented implementation at the operational level. Aside from template modules exist. Modules are complete solution without open slots and responsible for providing specific functionality.

In the template model, *properties* are used to state qualities or traits that are distinctive for the object described in the template, and for the context in which the template can be used. The properties of a template object are called *template object properties*. The properties of a context are called *pre-conditions*. Within the template object properties, *open slot properties* are defined: prescriptive properties that describe what the parts placed in the open slots should be able to do in order for the combination to work. The open slot properties provide a (re-)design process with knowledge about what it has to complete.

6.2 Template Retrieval

The template retrieval mechanism collects, stores and retrieves templates. Template retrieval entails matching queries with templates. Queries may specify aspects of the template (e.g. functionality, knowledge or compositional structure). The retrieval mechanism is domain independent and can be used within different design environments (e.g. MAF, architectural design). The template retrieval process may involve reformulating the query into sub-queries by applying design knowledge.

The design-centre within the multi-agent factory can communicate with a template retrieval process using two types of information: a Qualified Property Set used to communicate desired properties and their qualifications, and Retrieval Objectives used to influence the overall template retrieval process.

6.3 The multi-agent design-centre

The structure of the multi-agent design centre is based on the models presented in Chapter 5. This section describes the combination of parts of the models with a template retrieval

process to form a model for the *design-centre* of the multi-agent factory. An attempt is made to describe the interaction between the existing processes within the design model and the template retrieval process. Although the most important elements of the design-centre and key knowledge are described, a full description of process structures and knowledge structures is not provided. Instead, the elements are described at a higher abstraction level with the emphasis on the use of knowledge and processes. This section describes implications of the template-based design method on the three subprocesses of design: DPC, DODM and RQSM.

6.3.1 DPC in the multi-agent factory

The use of templates to modify design object descriptions and qualified requirement sets implies a need for DPC to be able to reason about a template-based design process. DPC has to determine an overall template-based design strategy and monitor the template-based design process.

Because templates are the primary building blocks used in the design process, DPC must be able to express strategies on how to use templates. For example, DPC can decide on the basis of overall design objectives which it has received, that only a limited number of templates may be used in the design, or perhaps, that a pre-defined skeleton agent template must be used for all agents.

Also, DPC must be able to process incoming process information from both DODM and RQSM regarding the template-based design process. For example, DODM may communicate to DPC that it is unable to construct an agent using the overall strategy that indicates that only three templates may be used. DPC must then be able to reason about this and decide that issuing a strategy indicating that the design process must be restarted and that *four* templates may be used, could lead to success.

6.3.2 RQSM in the multi-agent factory

This section describes the use of the template retrieval process within the RQSM process. First the location of template retrieval within the RQSM is discussed. Then the use of qualified property sets within RQSM is described. The third topic addressed is the RQS modification process, emphasizing the use of information from TR. The last topic describes the role of the co-ordination process with respect to the use of the template retrieval system.

RQSM is responsible in the design-centre for modifying the requirement qualification set. The process can make use of the Qualified Property Set information returned by the template retrieval process to refine qualified requirements. The assumption is made that the qualified property set contains the most important properties of the retrieved templates with respect to the query. This means that the conceptual object properties within the template set information will not be used by the RQSM process, as the most useful properties are already present in the qualified property set. The qualified requirements used in the template based design process are similar to the ones used in the model for re-design of compositional systems (see section 5.4), but additional knowledge is added indicating the relations between qualified requirements and templates.

Figure 15 displays an example RQS: within the two qualified requirements the '*hard*' argument represents the qualification of the requirement, and the '*user*' argument represents the agent that communicated the requirement to the multi-agent factory. The **related_to** entry indicates that qualified requirements '*rq1*' and '*rq2*' were added on the basis of knowledge returned by the template retrieval process, which indicated that template '*t1*' possesses the properties present in the qualified requirements.

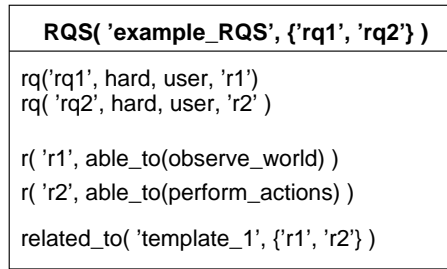


Figure 15. Example of a requirement qualification set, as used in the design-centre.

A typical RQS modification cycle using template retrieval involves the following steps:

- The RQSM process focusses on the qualified requirements to be modified.
- A qualified property set is created using the qualified requirements in focus.
- The RQSM co-ordination process is informed about the modification decision.
- Retrieval objectives are established by the RQSM co-ordination process.
- The QPS and the retrieval objectives are communicated to the template retrieval process.
- The template retrieval process returns a possibly modified QPS to the modification process and evaluation information of the retrieval process to the co-ordination process.
- Assessment of information returned by the template retrieval process.
- The modification process uses the QPS information to change the RQS if necessary.

Template Retrieval in RQSM

Because of the analogy between TEMPLATE RETRIEVAL and history maintenance, TEMPLATE RETRIEVAL is placed within the RQSM HISTORY MAINTENANCE process. Figure 16 displays the new situation at the process level.

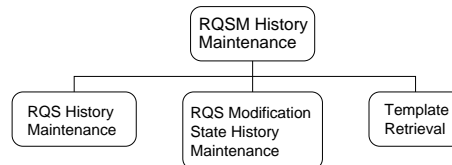


Figure 16. The template retrieval process within RQSM History Maintenance.

With the addition of TR in the RQS MANIPULATION process, a new method of rqs modification of the qualified requirement set becomes available to RQS MODIFICATION. RQS MODIFICATION can choose to construct a qualified property set from the focussed requirements in order to find a matching template set for it. At first this might not seem like a modification, but RQS MODIFICATION can use the information returned by TR to modify the current RQS. If TR can find a matching template set, the QPS which it has received is returned including information indicating which template(s) have been found. If TR cannot find a match for the QPS, it will try to find a QPS for which it can find one or more templates, according to the retrieval objectives it has received in its input interface. If TR succeeds in this task, the modified QPS is returned to RQS MODIFICATION. This task uses the QPS to modify the RQS. The flow of information is shown in figure 17. The dark line indicates the path the information must follow. As this path passes through a number of processes, these processes must also be aware of the type of information that it receives, in order to transfer it in the correct manner to the correct location. The processes in figure 17 that need to be aware of the type of information transferred are RQS MODIFICATION and RQSM HISTORY MAINTENANCE.

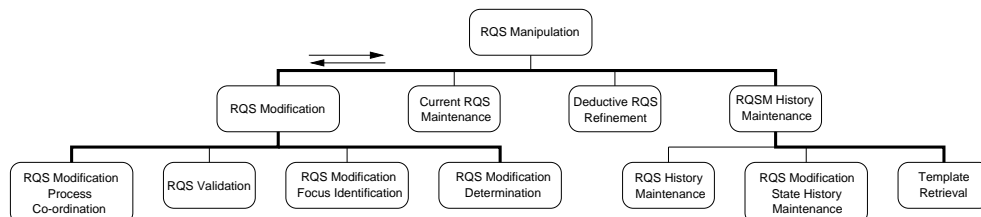


Figure 17. Communication to and from the template retrieval process within RQSM.

The retrieval objectives for TR are determined by RQS MODIFICATION PROCESS CO-ORDINATION. The objectives are based on information from RQS MODIFICATION and overall design strategy information. The qualified property set constructed by the modification process is used to communicate desired template properties to template retrieval and is constructed by RQS MODIFICATION DETERMINATION.

Qualified Property Set usage within RQSM

A qualified property set has to be created when template retrieval has been selected as the modification method for the currently focussed subset of the RQS. Figure 18 shows the use of qualified property sets within RQSM. A QPS has to be created by the RQSM process in order to be able to communicate with the template retrieval process. Also, RQSM has to be able to receive QPS information *from* the template retrieval process. The received information can then be used to modify the current RQS. RQS MODIFICATION bases its method selection on strategies received from RQS MODIFICATION PROCESS CO-ORDINATION and on information about the current focus from RQS MODIFICATION FOCUS IDENTIFICATION.

A qualified property set consists of one or more qualified properties. A qualified property is created using information from one of the focussed qualified requirements: the required property is extracted from the qualified requirement and inserted into a new qualified property information structure.

The RQS MODIFICATION process evaluates the incoming QPS to see if it is good enough to modify the RQS. RQS MODIFICATION may encounter two problems when assessing the incoming QPS: The QPS has *more* properties in it than were issued in the query to TR, or the QPS has *fewer* properties.

If more properties than issued are present, RQS MODIFICATION has to make the decision if the additional properties introduced can be added to the RQS or if they introduce new problems. The additional properties could conflict with the requirements already present in the set. On the other hand, the additional properties could also refine some of the requirements already present in the set.

If fewer properties are present than were issued, it must be decided if a sufficient number of the required properties are refined to continue with the found template, or if a new template retrieval cycle must be initiated.

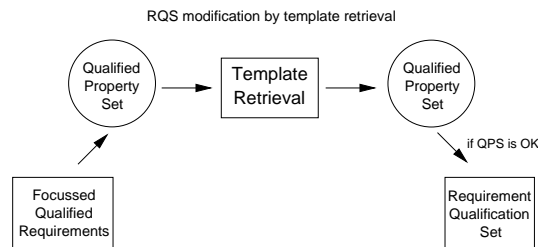


Figure 18. Use of the template retrieval process and qualified property sets in RQSM.

RQS Modification using template retrieval

If it has been determined that the returned QPS from TR is good enough, the properties and their qualifications are added to the RQS. The information within the QPS consists of property information indicating which properties are satisfied by which returned templates. The qualified properties returned could be either the same as those issued to TR (an ideal case), or they are not, in which case TR has been unable to find matching templates and has modified the QPS to some degree. If there are *fewer* properties than originally issued, the focussed qualified requirements which have not been satisfied by the returned template must be removed from the current RQS. If there are *more* properties than originally proposed, the new properties have to be inserted into the current RQS as qualified requirements.

For example (see figure 19), RQS MODIFICATION sends a QPS containing one qualified property to TR, and TR returns a QPS containing two qualified properties, of which one has the same property as the one sent by RQS MODIFICATION. The new qualified property '*tr-pq2*' is added to the RQS. The qualified property containing the same property as the qualified property in the QPS *sent* to TR, '*tr-pq1*', is not added to the RQS, as it is already present in the qualified requirement used to query TR, '*rq1*'. However, its relation with the

found template(s), that is also indicated within the returned QPS, is applied to the qualified requirement used.

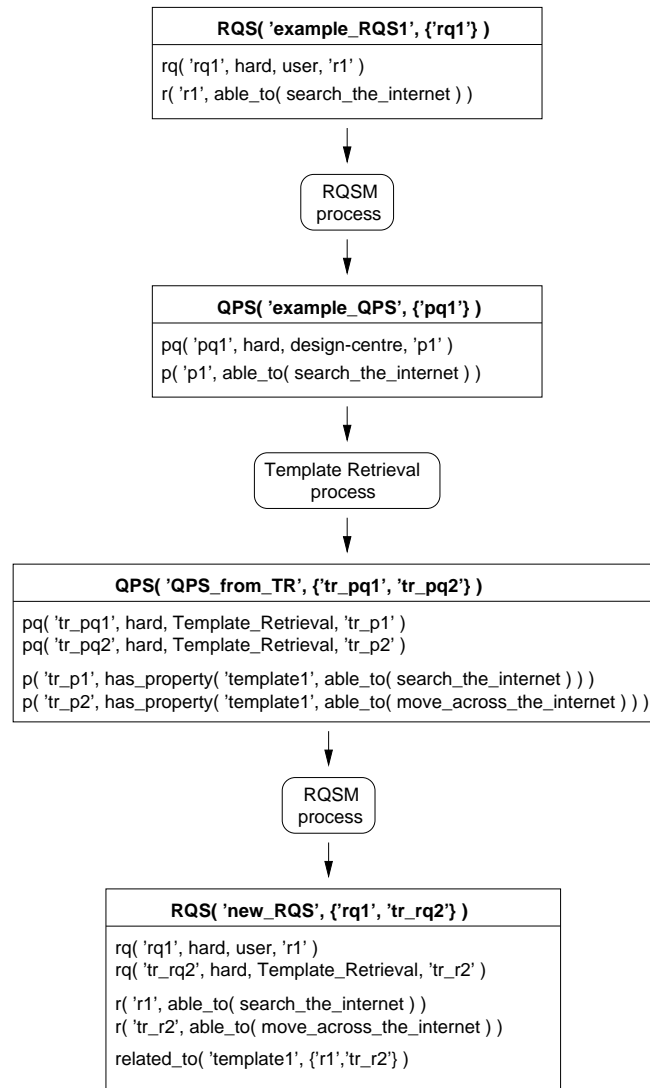


Figure 19. adding qualified properties to a requirement qualification set.

Process co-ordination within RQSM

Within RQSM, RQS MODIFICATION PROCESS CO-ORDINATION is responsible for determining the retrieval objectives once the template retrieval method has been determined to be the modification method. Using this information, the structure of the template information can be influenced, and constraints on the retrieval process itself can be imposed. It should also be possible for the co-ordination process to indicate the importance of an objective. This allows for the indication of preferences without risking a failure of template retrieval. For example, Co-ordination could decide to issue an objective indicating that the retrieval time is subject to a time constraint, but that the indicated time is to be used as a guideline, not a deadline: **retrieval_objective(minimize_retrieval_time(2min), soft)**

Retrieval objectives indicating a template structure:

- **single.template**: Indication that a single template is desired by RQS MODIFICATION.
- **alternative_templates**: Indication that multiple alternatives for the same QPS are desired by RQS MODIFICATION.
- **basic.template**: Indication that a template is desired that in addition to the properties desired also provides an agent basis. This objective could for example be issued at the beginning of the design process.

- `interpret_query`: Indication that the QPS sent by RQS MODIFICATION may be modified by TEMPLATE RETRIEVAL.

Retrieval objectives regarding the template retrieval process:

- `minimize_retrieval_time`: Indication of the desired time TEMPLATE RETRIEVAL has at its disposal.
- `template_code_object_language`: Indication of the programming language the code-objects in the templates should have.

RQS MODIFICATION PROCESS CO-ORDINATION also receives the evaluations of the retrieval objectives from TEMPLATE RETRIEVAL. This information is used in the reasoning process within the co-ordination process to determine the local strategy within RQS MODIFICATION. If the objectives are not sufficiently met, a decision could be made to stop the design process or to re-activate TR with new retrieval objectives.

6.3.3 DODM in the Multi-Agent Factory

This section discusses four topics concerning the use of the template retrieval process within DODM. First the location of TR within DODM is discussed, after which the use of qualified property sets within DODM is discussed. Then the DOD modification process is described, with the emphasis on the use of information from TR. The fourth and last topic describes the role of the co-ordination process with respect to the use of the template retrieval system.

DODM is the process which is responsible in the design-centre for combining the templates into an agent description. The Requirement Qualification Set communicated by RQSM is used to validate the Design Object Description to determine if (further) modification of the DOD is necessary. If modification is necessary, DODM needs to formulate a query to retrieve useful templates from the template retrieval process. A typical DOD modification cycle using template retrieval involves the following steps:

- The design object description is analyzed, and a focus is made on possible open slots.
- A QPS is created when DODM has chosen for modification without notification of RQSM.
- The DODM co-ordination process is informed about the modification decision.
- The DODM co-ordination process establishes retrieval objectives, possibly using specific template indications.
- The QPS and retrieval objectives are sent to the template retrieval process.
- The template retrieval process returns a template set.
- Assessment of the information returned by the template retrieval process.
- The modification process uses the template information to modify the design object description.

The design object description used to represent the design in DODM within the multi-agent factory, consists of three parts:

1. A description of the elements within the agent, as applied from the used templates. This information consists of descriptions of agent components and their relations, information types and knowledge bases used within the agent and the information flow between the components in the agent.

```

is_component( example_component )
has_input_information_type( example_component, example_input_type )
has_subcomponent( example_component, example_subcomponent )
....

```

Figure 20. Example of a partial design object description, as used in the design-centre.

2. Information on how the used templates are applied in the agent description. When a template is applied, the descriptions within the template are used to create an instance of the template to insert into the DOD. To make it possible to construct the operational agent using the code objects within the used templates, the agent-assembly process also

needs to receive indications of which templates were used where in the design description. Also, the modification process itself could benefit from information indicating how the templates have been used in the design description. For example, when a modification entails replacement of a previously inserted template by a newer version, the modification process could locate the part of the design description which has to be replaced by examining the information indicating how the template was inserted.

- Information on the properties of the design object description. This information can be used to determine properties of the design object description as a whole, based on the properties of the applied template information. For example, it could occur that adding a second template to a design introduces new properties, which in conjunction with the properties from the first template produce additional new properties for the design object description as a whole. The information also contains properties of the open slots within the design.

To be able to derive these new properties, the DODM process should either possess the same semantic property networks TR uses in its retrieval process, or should be able to request this information from the TR process.

The process of constructing a design object description within the multi-agent factory consists of repeatedly applying templates, finding the open slots within the new design object description, and retrieving and applying templates to fill in the open slots. The component responsible for examining the open slots within the design is DOD OPEN SLOT ANALYSIS.

Template Retrieval in DODM

Within DODM, the interface to the template retrieval process is located in the DODM HISTORY MAINTENANCE PROCESS. Figure 21 shows a process-tree containing the TR process. The TR process plays an important role within DODM, as it is the process which delivers the primary building blocks used in the artefact design process.

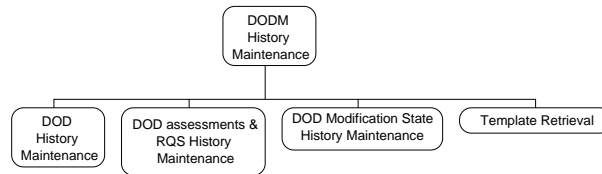


Figure 21. Location of Template Retrieval in DODM History Maintenance.

Within DOD MODIFICATION, two processes are able to communicate with TR: DOD MODIFICATION DETERMINATION and DOD MODIFICATION PROCESS CO-ORDINATION. DOD MODIFICATION DETERMINATION can communicate queries *to*, and receive template information *from* TR. DOD MODIFICATION PROCESS CO-ORDINATION can communicate retrieval objectives *to*, and receive evaluations of retrieval objectives from TR. Figure 22 displays the flow of information between the relevant processes.

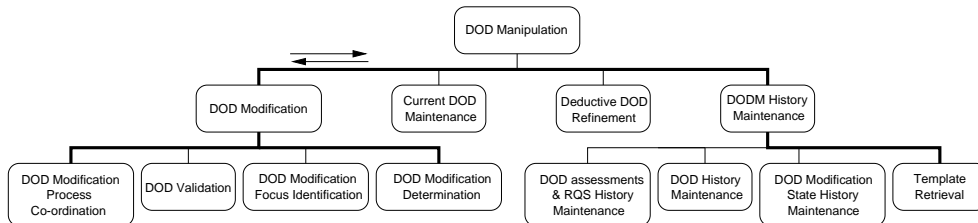


Figure 22. Communication to and from the template retrieval process within DODM.

Qualified Property Set usage within DODM

The DODM process can use qualified property set information to communicate required properties to template retrieval. These properties may include properties expressing the desired structure of the template. For example, a qualified property set can be constructed containing, among other properties, an indication that the returned (composed) template must contain

two components.

DOD Modification using templates

Modifying a design object description entails using conceptual object information contained in a template to add information to a design object description. Information describing the (partial) design objects in the template as well as information describing the properties of the (partial) design objects are used. Within the information describing the properties, information about the properties of *open slots* is also included. The handling of open slots by the DOD modification process can be subdivided into three actions:

- Detecting the open slots in the design. This includes recognizing possible relations between open slots.
- Deciding if the open slots encountered require notification of RQSM, or if the open slots can be filled by directly querying TR.
- Applying templates into the design, by inserting them into the open slots of the current design object description.

The process capable of detecting the open slots in design object descriptions is added as an additional process within DOD MODIFICATION. Figure 23 displays the new situation. DOD OPEN SLOTS ANALYSIS bases its analysis on incoming DOD information and an incoming local strategy from the modification process co-ordination process.

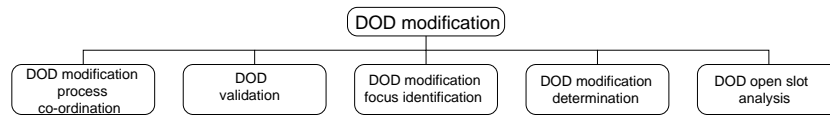


Figure 23. Location of DOD Open Slot Analysis in DOD Modification.

DOD OPEN SLOT ANALYSIS communicates the analysis of the open slots to co-ordination. Included within this information is a recommendation indicating if the open slots are to be used as requirements in the design process, or that DODM itself can issue a query to TR. Based on this recommendation and on strategy information, co-ordination decides which method of open slot handling is currently the best.

When DOD OPEN SLOT ANALYSIS examines the DOD, its main task is to find an open slot that is most useful for finding a template. For example, if the current DOD contains three open slots, DOD OPEN SLOT ANALYSIS is used to establish if there are open slots which are 'sub-slots' of other open slots in the design. If these are found, the top-most open slot is selected as the open slot which properties are to be used to retrieve a fitting template. The process also determines whether filling the open slot that is selected for modification requires RQSM to query for a new template, or if DODM is capable of querying for a new template by itself. If the design co-ordination process has indicated that a template indication has been communicated by RQSM, a request for that template can be issued directly. If no template was indicated, but analysis indicates that it is not necessary to involve RQSM in this modification (for example because the modification involves a small open slot not related to any qualified requirements), the properties of the open slot can be used by DOD MODIFICATION DETERMINATION to assemble a QPS, which can then be sent to the template retrieval process.

Using a template to modify a design object description entails determining which template to apply, and also to which open slot the template is to be applied. Within a design object description, four types of open slots can be present:

- **Component open slot:** An open slot component is a component present in the (partial) design object description. Information links can be connected to it, and the input and output information types can be defined, but the component itself has no inner 'workings'. It is a composed component without any subcomponents, information link definitions and task control knowledge. The component open slot is filled using a component template.

A template that defines a filling for a component open slot is applied by instantiating the template component *within* the open slot component already present in the DOD. The component is inserted as a *subcomponent* in the open slot, and automatically generated information links are generated to carry the necessary information from the outside of

the open slot component to the component instantiated from the template. Also, simple task control knowledge is added to the open slot component indicating that if the open slot component is activated, first the information link transferring the information to the added component is activated, then the added component itself is activated, and finally the outgoing information link is activated.

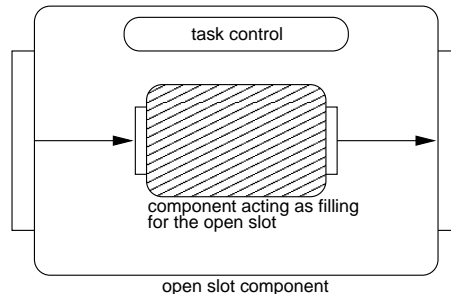


Figure 24. Filling an open slot component.

- **Knowledge Base open slot:** Within a (partial) design object description, components may exist which contain a knowledge base open slot. Properties of the open slot indicate the properties that are desired from the knowledge base that is to be used to fill in the open slot. The knowledge base defined within the template presents itself as a single knowledge base, but within this single knowledge base, possibly multiple knowledge bases could be defined.
- **Information Type open slot:** This type of open slot defines an empty information type. Using a template to fill the open slot entails adding the new information type as a *subtype* of the open slot information type. Within the information used to fill in the open slot, possibly multiple information types may be defined.
- **Information Link open slot:** An information link open slot is an empty information link. This means that no information types are specified, nor is there any information filtering present within the link. An information link template is used to fill in the open slot information link.

Templates also contain conceptual pre-conditions indicating what the template expects as its context in a design object description. For example, a knowledge base template could demand that the DOD provides certain knowledge structures. These pre-conditions can be used to retrieve new templates if the information that is demanded is not present in the current DOD.

Process co-ordination within DODM

The co-ordination process within DOD MODIFICATION is responsible for determining the retrieval objectives to be communicated to TR. Next to the retrieval objectives already stated in the section describing the co-ordination process within RQS MODIFICATION (see section 6.3.2), the co-ordination process within DOD MODIFICATION is also capable of asking for a specific template.

The retrieval objectives are based on information from DOD MODIFICATION DETERMINATION, DOD OPEN SLOT ANALYSIS and on the current modification strategy. When DOD MODIFICATION DETERMINATION determines that a modification is to be made using template information, the process examines the current RQS to determine if RQSM has issued a requirement indicating which template to use. If such a requirement is present, information containing the template indication is communicated to the co-ordination process.

6.4 Example knowledge in the multi-agent design-centre

This section presents a detailed trace of a design process cycle within the multi-agent factory design-centre. First, the RQSM process modifies the incoming qualified requirement set using the template retrieval process. The second part of the trace shows the modification of the design object description using template information within DODM. The DESIGN PROCESS CO-ORDINATION process guides the overall design process. The trace shows for both the manipulation processes the most important processes with respect to the use of the template

retrieval process. Other processes, such as the maintenance of history information, are not addressed in this trace.

DESIGN PROCESS CO-ORDINATION

The overall design co-ordination process determines when a new design cycle has to be started. A relevant overall design strategy is sent to the other components, indicating that a new initial requirement qualification set is present and no initial design object description. The initial received RQS is shown in figure 25.

| RQS('RQS', irq1, irq2, irq3) |
|--|
| rq('irq1', hard, PA, ir1) |
| rq('irq2', hard, PA, ir2) |
| rq('irq3', hard, PA, ir3) |
| r('ir1', able_to(perform(agent_functions))) |
| r('ir2', able_to(search)) |
| r('ir3', able_to(visit_www_stores)) |
| r('ir4', able_to(analyze_computer_store_info)) |

Figure 25. The initial requirement qualification set.

REQUIREMENT QUALIFICATION SET MANIPULATION

Within the RQSM process,

- Examination of the incoming RQS.
- Retrieval of the QPS to modify the RQS.
- Application of the QPS to the RQS.
- Preparation of the RQS for communication to DODM.

When RQSM is activated, two new types of information are present on its input: A new strategy issued by DESIGN PROCESS CO-ORDINATION and an initial requirement qualification set issued by another agent in the system. Within RQSM, RQS MODIFICATION PROCESS CO-ORDINATION, a subprocess of RQS MODIFICATION, receives in its input interface: the overall strategy and no history information as the design process has just started.

RQS Validation

The incoming RQS is analyzed and the first qualified requirement is considered to be useful for template retrieval. Knowledge used in this step may be (partial) knowledge about the semantic property network information used in the TR process. Using this knowledge, the validation process could determine that the property within the requirement is available within the knowledge of TR. This is a good indication that TR will be able to find a template for it. TR sends information regarding the analysis to RQS MODIFICATION PROCESS CO-ORDINATION. **evaluation(irq1, 'fit_for_template')**

RQS Modification Process Co-ordination

The overall design strategy received indicates that a standard design strategy has to be followed. The information also indicates that an initial requirement qualification set was communicated to the design process. This is an indication for the co-ordination process that a new design cycle has started. The co-ordination process has also received information from RQS VALIDATION indicating that one of the qualified requirements is useful for an attempt to retrieve a template. The co-ordination process decides that the local strategy will be to retrieve a template.

The local strategy information is communicated to the other components, relevant information is sent to RQSM HISTORY MAINTENANCE, and information on the progress of the RQSM process is sent to DESIGN PROCESS CO-ORDINATION.

RQS Modification Focus Identification

The process responsible for determining a focus on a subset of the RQS receives the incoming strategy, the validation information from RQS VALIDATION, and also the initial RQS itself. The strategy indicates that a template should be retrieved. The validation information contains information indicating that *irq1* is suitable for template retrieval. The focus is placed on *irq1*, as this seems the most high-level requirement to start with.

The focus information is communicated to the RQS MODIFICATION DETERMINATION PROCESS. Information on the progress of the focus determination process is communicated to RQS MODIFICATION PROCESS CO-ORDINATION.

RQS Modification Process Co-ordination

The co-ordination process is informed by the processes of their progress. The focus determination process has succeeded in determining a focus. There is no need to alter the local strategy or intervene in the design process. The relevant information is sent to RQSM HISTORY MAINTENANCE, and DESIGN PROCESS CONTROL is informed on the progress of the process.

RQS Modification Determination

The process responsible for determining a modification to the RQS receives a strategy from the co-ordination process, information on the current foci within the RQS, and the RQS itself. The strategy indicates that a template needs to be retrieved. The focus information is used to create a qualified property set from the currently focussed initial requirement.

In this example, the qualification is taken directly from the qualified requirement. Another possibility is that the retrieval of templates is always based on properties qualified as hard.

| QPS('QPS1', {'pq1'}) |
|--|
| pp('pq1', hard, personal_assistant, 'p1') |
| p('p1', able_to(perform(agent_functions))) |

Figure 26. The qualified property set that is to be sent to TR.

The set of qualified properties, in this case only one, is communicated to TR. Information about the progress of the modification process is sent to RQS MODIFICATION PROCESS CO-ORDINATION.

RQS Modification Process Co-ordination

New information is available from the modification determination process, indicating that a qualified property set has been created successfully. The co-ordination process must now issue retrieval objectives to TEMPLATE RETRIEVAL. As no special overall design strategies have been issued, and the design process has recently started, a retrieval objective is sent indicating that the qualified property set may be interpreted by TR.

Relevant information is communicated to RQSM HISTORY MAINTENANCE and the retrieval objective is communicated to TR.

Template Retrieval

Information is received from RQS MODIFICATION PROCESS CO-ORDINATION and RQS MODIFICATION DETERMINATION. The retrieval process returns evaluations of the retrieval objectives indicating that a template has been found. The qualified property set returned is the following:

| QPS('QPS2', {'pq1', 'pq2', 'pq3'}) |
|---|
| based_on(QPS1) |
| pp('pq1', hard, TR, 'p1') |
| pp('pq2', hard, TR, 'p2') |
| pp('pq3', hard, TR, 'p3') |
| p('p1', has_property('t1', able_to(perform(agent_functions)))) |
| p('p2', has_property('t1', able_to(interact_with(material_world)))) |
| p('p3', has_property('t1', able_to(interact_with(agent)))) |

Figure 27. The qualified property set returned by TR.

Within the returned QPS, each property indicates that template 't1' is the template which is to be used to add the property to the design object description. The QPS is communicated to RQS MODIFICATION DETERMINATION. The evaluations are communicated to RQS MODIFICATION PROCESS CO-ORDINATION.

RQS Modification Process Co-ordination

The evaluations of the retrieval objectives received from TR indicate that a template has been retrieved. A new local strategy is issued indicating that a template application cycle now has to take place. The strategy is communicated to the other components and relevant information is communicated to RQSM HISTORY MAINTENANCE.

RQS Modification Determination

The QPS returned by TR is used to modify the current RQS. The qualified properties within the set consist of more detailed property information than originally sent to TR. Two additional properties were inserted into the QPS, as these properties are also fulfilled by the template. Within each property, information is contained about which template this property is related to. This is useful in case a composed template is returned, containing several properties and templates.

The actual property of a template is extracted from the *has_property* information, and this property is used to create a new requirement. The template indication from the *has_property* information is also added to the RQS, as DODM can use this information to retrieve relevant templates when necessary. Also, the relation between the original requirement that was sent to TR and the returned property containing the same property information is recognized and the redundant property is not added to the RQS. Figure 28 displays the qualified requirement set created using the QPS information.

The addition of the new requirements leads to the following RQS:

| RQS('RQS1', {'irq1', 'irq2', 'irq3', 'rq2', 'rq3'}) |
|--|
| <pre> rq('irq1', hard, PA, ir1) rq('irq2', hard, PA, ir2) rq('irq3', hard, PA, ir3) rq('rq2', hard, TR, 'r2') rq('rq3', hard, TR, 'r3') r('ir1', able_to(perform(agent_functions))) r('ir2', able_to(search)) r('ir3', able_to(visit_www_stores)) r('ir4', able_to(analyze_computer_store_info)) r('r2', able_to(interact_with(material_world))) r('r3', able_to(interact_with(agent))) related_to('t1', {'ir1', 'r2', 'r3'}) </pre> |

Figure 28. The requirement qualification set after the template modification method.

The co-ordination process is notified of the current progress.

RQS Validation

The validation process examines the new RQS and finds that some qualified requirements are related to a template, but not all. RQS VALIDATION reasons that the qualified requirements with no template relation are not valid for use in the current RQS.

RQS Modification Process Co-ordination

The information from the validation process and the modification determination process indicates that the a new RQS has been created, but not found to be valid. A local strategy is sent to RQS MODIFICATION PROCESS CO-ORDINATION that the RQS must be modified according to the information communicated by RQS VALIDATION.

The strategy is communicated to the other components and relevant information is communicated to RQSM HISTORY MAINTENANCE.

RQS Modification Determination

The RQS is modified by removing the qualified requirements indicated by the validation process. The new RQS is shown in Figure 29.

| RQS('RQS2', {'irq1', 'rq2', 'rq3'}) |
|---|
| rq('irq1', hard, PA, 'ir1') rq('rq2', hard, TR, 'r2') rq('rq3', hard, TR, 'r3') |
| r('ir1', able_to(perform(agent_functions))) r('r2', able_to(interact_with(material_world))) r('r3', able_to(interact_with(agent))) related_to('t1', {'ir1', 'r2', 'r3'}) |

Figure 29. The requirement qualification set created using the template information.

RQS Validation

The validation process finds no problems within the new current RQS. The co-ordination process is notified of this progress.

RQS Modification Process Co-ordination

The information communicated by the other components indicates that the RQS is now ready to be sent to the DODM process. The components are informed of this decision and information is communicated to DESIGN PROCESS CO-ORDINATION about the progress. Also, relevant information is communicated to the history process.

DESIGN PROCESS CO-ORDINATION

The information received from the RQSM process indicates that a new RQS has been created and that DODM is to use this RQS to create a new design object description. The relevant overall strategy information is sent to the other components.

DESIGN OBJECT DESCRIPTION MANIPULATION

Within the detailed trace of the DODM process, four major steps can be distinguished:

- Examining the incoming RQS information.
- Retrieval of a relevant template.
- Application of a template.
- Analysis of the new design object description, to reveal if it satisfies the current RQS, and if there are open slots in the design which have to be filled in.

DOD Modification Process Co-ordination

The co-ordination process receives the overall design strategy and uses this information to determine a local strategy. The local strategy is communicated to the other components in DOD MODIFICATION.

DOD Validation

The incoming RQS from RQSM is used to determine if the design object description is satisfactory. As the current DOD is empty, all qualified requirements from the RQS are unsupported. Information about the validation is sent to the co-ordination process.

DOD Modification Process Co-ordination

Standard activation, no important changes.

DOD Modification Focus Identification

The focus determination process cannot determine a focus as the DOD is currently empty. Information about the empty focus is communicated to DOD MODIFICATION PROCESS CO-ORDINATION and DOD MODIFICATION DETERMINATION.

DOD Modification Process Co-ordination

Standard activation, no important changes.

DOD Modification Determination

A new modification step has to be made. The information in the RQS indicates that a template has been found to use in the modification process. The information containing the template indication is communicated to the co-ordination process.

DOD Modification Process Co-ordination

The co-ordination process receives this information and uses it to formulate a new retrieval objective. The objective is communicated to TR.

Template Retrieval

Information is received from DOD MODIFICATION PROCESS CO-ORDINATION. The template indicated in the objective is located and the template set information is communicated to DOD MODIFICATION DETERMINATION for use. The retrieval process returns evaluations of the retrieval objectives to DOD MODIFICATION PROCESS CO-ORDINATION.

DOD Modification Process Co-ordination

Standard activation, no important changes.

DOD Modification Determination

The template information is used to create the first design object description. Within the DOD, three language elements can be distinguished:

- The language elements describing the artefact elements: The individual parts of an agent: Information describing these elements can be seen in figure 20. This information is taken from the conceptual description within the template. The names of elements used in the template are substituted with 'real world' names currently present in the design object description.

```
is_component( component_name )
is_composed( component_name )
has_input_information_type( component_name, information_type_name )
has_output_information_type( component_name, information_type_name )
is_open_slot( knowledge_base_name )
is_open_slot( information_link_name )
is_open_slot( component_name )
is_open_slot( information_type_name )
is_primitive( component_name )
is_task_control( task_control_name )
is_information_type( information_type_name )
has_subcomponent( component_name, component_name )
has_information_type( component_name, information_type_name )
is_information_link( information_link_name )
has_source_component( information_link_name, component_name )
has_destination_component( information_link_name, component_name )
has_source_level( information_link_name, level_number )
has_destination_level( information_link_name, level_number )
has_information_link( component, information_link_name )
```

Figure 30. The knowledge elements used to describe the design object.

The example template used in this trace describes several information types not further mentioned in this trace. The components and information links described in the template are shown in figure 31.

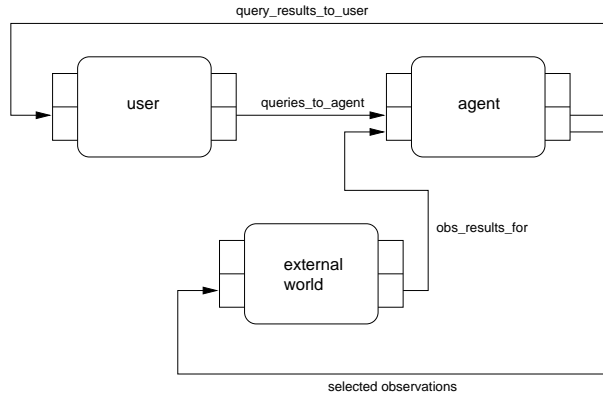


Figure 31. A visual representation of the conceptual description, constructed using the example template.

- The language elements describing how the templates are used in the modification process. When the template has been applied, information is recorded relating template information to the DOD. A description is made on which open slot in which template has been filled with the current template. In this example, the template has been used to fill an empty DOD, which is also described in the DOD.

template_usage('t1', empty_DOD, no_template)

- The language elements describing the properties of the design object description. These properties are taken from the properties contained in the applied templates. This includes properties of the elements itself, and the open slot properties describing the properties desired from the elements used to fill in the open slot.

```

has_property( agent, is( toplevel ) )
has_property( agent, able_to( perform( agent_functions ) ) )
has_property( agent, able_to( interact_with( user ) ) )
has_property( agent, able_to( interact_with( external_world ) ) )

```

Figure 32. The template-derived properties added to the DOD from the example template.

Information regarding the progress of the process is sent to the co-ordination process.

DOD Modification Process Co-ordination

Standard activation, no important changes.

DOD validation

The validation process checks to see if the new DOD satisfies the requirements communicated by the RQSM process. The properties of the design object description match with the properties stated within the RQS. Information regarding the validation is communicated to the co-ordination process.

DOD Modification Process Co-ordination

The co-ordination process decides that the modification process in this cycle is finished. DESIGN PROCESS CO-ORDINATION is informed about the progress.

DESIGN PROCESS CO-ORDINATION

The information communicated by the DODM process indicates that the dod modification cycle has been completed. The RQSM process now has to modify the rqs to continue the process.

6.5 Assembly process

The *Assembly* process is responsible for converting the final conceptual design object description into a detailed operational agent specification. Within the final design object description, information is present indicating which templates were used in the final design description and how they are used. This information is used by the assembly process to retrieve the used

templates and assemble the operational object descriptions that are present in the templates. This means that the assembly process also has to be able to communicate with the template retrieval process (see figure 33). This has some potential implications for the template retrieval process. Template retrieval has to be able to communicate 'composed' templates that were possibly constructed during the design process prior to the assembly process. This information concerning composed templates that were constructed 'when needed' has to be retained in the template retrieval process long enough for the assembly process to retrieve them.

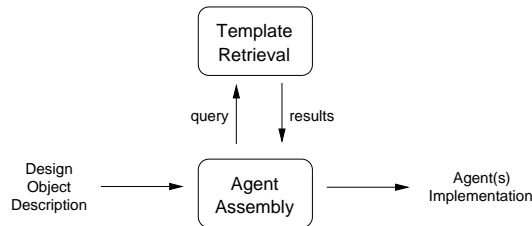


Figure 33. The assembly process also makes use of the template retrieval process.

The assembly is able to create detailed, operational descriptions using different languages. For example, if the client of a multi-agent factory asks for a number of agents that are able to run on a Java Virtual Machine, the design-centre within the multi-agent factory has already communicated this to the template retrieval process to make sure that the returned templates are available in the desired implementation language.

The operational object description in a template has the same open slots as were defined in the conceptual object description in a template. The assembly process can use the knowledge on how the *conceptual* open slots were filled in to fill in the open slots within the *operational* object description.

7 Agent configuration example

Section 7.1 describes a sample agent configuration process. The example covers multiple template application cycles, and is less detailed than the example in section 6.4. Each design-cycle description within the trace consists of a high-level overview of the DODM, RQSM and DPC processes. For a more detailed description of the RQSM and DODM processes within the design-centre, see section 6.3. Section 7.2 describes the prototype agent design process.

7.1 Trace

This section describes the design process configuring the example agent. The trace is fairly high-level, and emphasizes the interaction between RQSM and DODM, and the usage of the template retrieval process.

Design Process Co-ordination

Overall design process objectives are received, which indicate that a new agent design is to be made. The language used for the final, detailed description of the agent is to be Java [Flanagan, 1999]. No additional constraints are imposed on the design process. An overall design strategy is communicated to the other processes.

Requirement Qualification Set Manipulation

The initial qualified requirements communicated to the design-centre indicate that an agent needs to be designed that is capable of searching computer stores on the Internet for information concerning the various computer parts available on the sites. The resulting agent needs to be able to receive information on the computer part and the preferred price, and return information indicating at which stores on the Internet the part is available at that (or a comparable) price. RQSM selects the requirement that is currently believed to be most applicable. In this case this is the qualified requirement indicating that an agent is needed that is *able to*

perform weak agent functions. This is the most basic template available. A qualified property set is created using this qualified requirement. Retrieval objectives are generated indicating that the template retrieval process should return one template in the template set, and that templates should only be returned if the operational object description is available in Java. The information is communicated to TR.

A useful template has been found by TR and RQSM uses the QPS information returned by TR to create a new RQS. An indication is also added to indicate to DODM which template is to be used. The RQS is communicated to DODM and DESIGN PROCESS CO-ORDINATION is informed about the progress of the RQSM process.

Design Process Co-ordination

The co-ordination process decides that DODM has to use the information communicated by RQSM to modify the design object description (currently empty). DODM and RQSM are informed about the decision.

Design Object Description Manipulation

The RQS communicated by RQSM is used to retrieve a template which can be used in the DOD. The template is inserted into the DOD and provides the components, information links and information types shown in figure 34. The *user* and *external_world* components are fully specified by the template. Also indicated in the figure are the open slots: the *agent* component is an open slot and the information types:

- generic_observations_to_be_performed
- generic_observations_results
- generic_incoming_communication_info
- generic_outgoing_communication_info

are open slots. The contents of these open slots are not yet specified, and need to be filled by applying additional templates. The *agent* template allows for an additional agent template to be inserted in the design. The information type open slots allow that the specific information provided and needed by the agent can be further instantiated by additional templates.

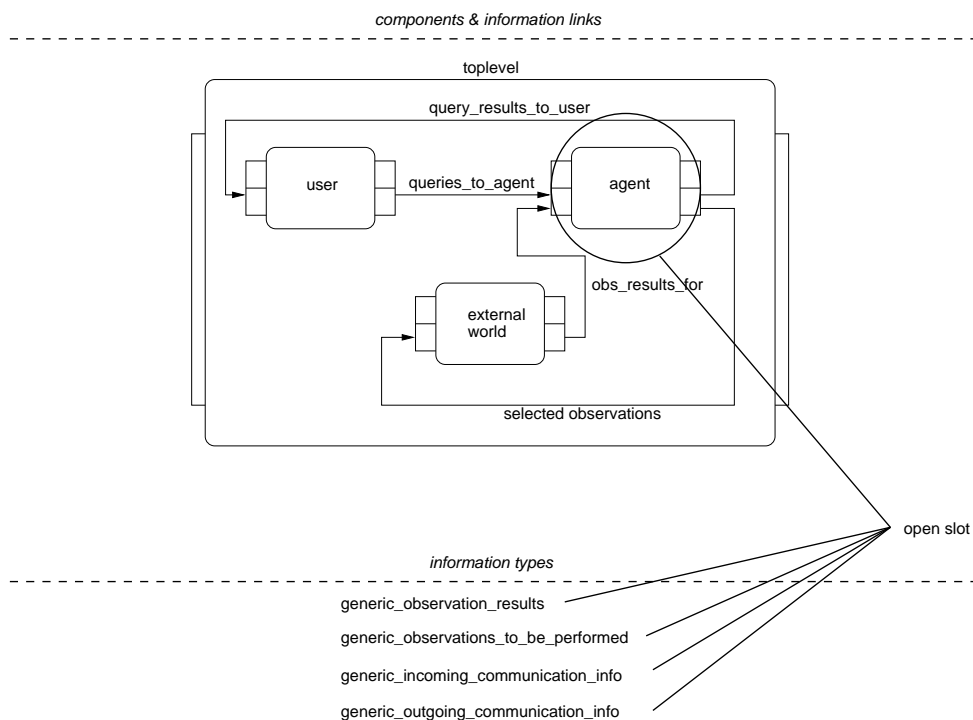


Figure 34. The elements present in the template applied by DODM.

The open slots in the DOD are examined and the AGENT component open slot is selected. The properties of the open slot indicate that a component that has the ability to perform agent functions needs to be inserted. The open slot information is communicated to RQSM and DESIGN PROCESS CO-ORDINATION is informed about the progress of the DODM process.

Design Process Co-ordination

The co-ordination process decides that RQSM is to continue the design process. DODM and RQSM are informed about the decision.

Requirement Qualification Set Manipulation

The information regarding the open slots communicated by DODM and the RQS information are used by RQSM to construct a new QPS. The QPS contains properties indicating that a template is desired that is *able to perform agent functions* (information from the open slot) and is also *able to search* (information from the requirements). Retrieval objectives are determined. The QPS and the retrieval objectives are communicated to TR.

A template has been found by TR and the QPS information is used to modify the current RQS. The RQS is communicated to DODM and DESIGN PROCESS CO-ORDINATION is informed about the progress of the RQSM process.

Design Process Co-ordination

The co-ordination process decides that DODM has to use the information communicated by RQSM to modify the design object description. DODM and RQSM are informed about the decision.

Design Object Description Manipulation

The second template is a composed template: templates are provided for the AGENT component open slot and for the information type open slots that are also present in the current DOD. TR has provided these additional templates (not requested by RQSM) because they are useful in conjunction with the desired SEARCH AGENT template, and the retrieval objectives did not indicate that TR should only return templates with the desired properties.

The templates (see figure 35) are applied to the DOD and the open slots are filled in. The *search agent* component is used to fill in the *agent* component open slot. The information type *world_info* is used to fill in the *generic_observations_to_be_performed* and *generic_observations_results* open slots. The information type *agent_info* is used to fill in the *generic_incoming_communication_info* and *generic_outgoing_communication_info* open slots. The new templates, however, introduce new open slots. The information types *agent_info* and *world_info* have to be filled in and also two components, WORLD INTERACTION MANAGEMENT and AGENT SPECIFIC TASK.

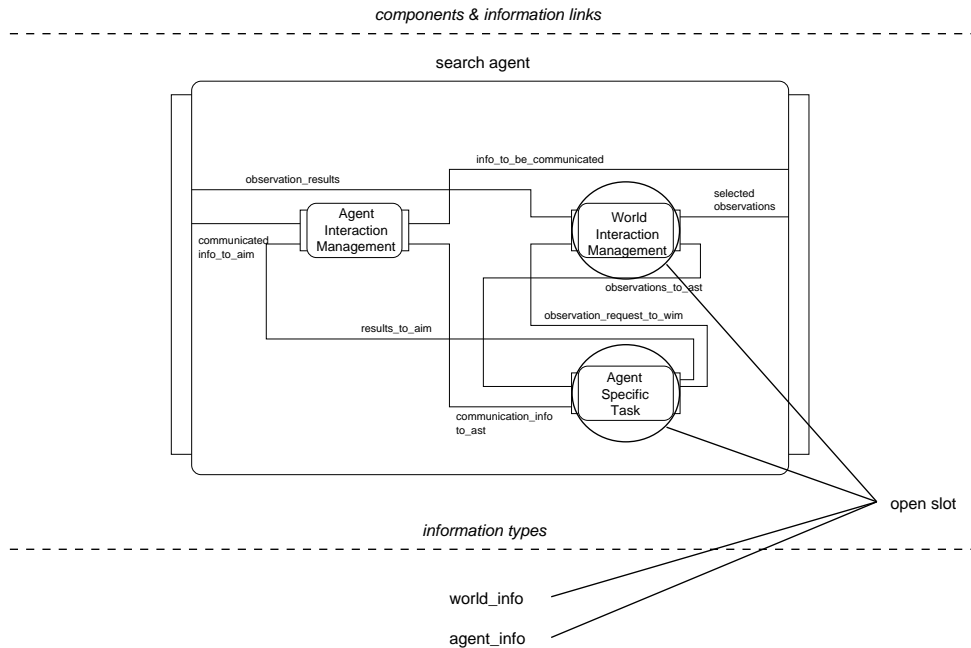


Figure 35. The template information used in the second DOD modification cycle.

The AGENT SPECIFIC TASK open slot is selected, possibly because DODM reasons that once specific information about the task the agent will perform (in this case, *where* is to be searched) has become available, the other open slots will be easier to fill in. The open slot information is communicated to RQSM and DESIGN PROCESS CO-ORDINATION is informed about the progress of the DODM process.

Design Process Co-ordination

The co-ordination process decides that RQSM is to continue the design process. DODM and RQSM are informed about the decision.

Requirement Qualification Set Manipulation

The open slot information communicated by DODM and the RQS information are used to create a QPS. The properties used in the QPS indicate that this time a template is desired that has the properties: *able to search stores on the Internet* and *able to perform agent specific task*. The QPS is communicated to TR.

TR finds a template and the QPS information is used by RQSM to modify the RQS. The RQS is then sent to DODM.

Design Process Co-ordination

The co-ordination process decides that DODM is to continue the design process. DODM and RQSM are informed about the decision.

Design Object Description Manipulation

The retrieved template is again a composed template (see figure 36). Within the template a component template is available for filling in the AGENT SPECIFIC TASK open slot: WWW SEARCH AGENT SPECIFIC TASK, a component template for filling in the WORLD INTERACTION MANAGEMENT open slot: WWW SEARCH WORLD INTERACTION MANAGEMENT, an information type template for the *world_info* information type: *www-world_info*, and an information type template for the *agent_info* information type: *www-agent_info*. DODM can use these templates to fill in the open slots currently present in the DOD. Analysis of the new DOD reveals that new open slots are present in the design. The information types that are to contain infor-

mation regarding the specific objects that are to be searched: *www_search_objects*, and the locations of those objects are not yet filled in: *www_location_info*. The open slot information is communicated to RQSM and DESIGN PROCESS CO-ORDINATION is informed about the progress of the DODM process.

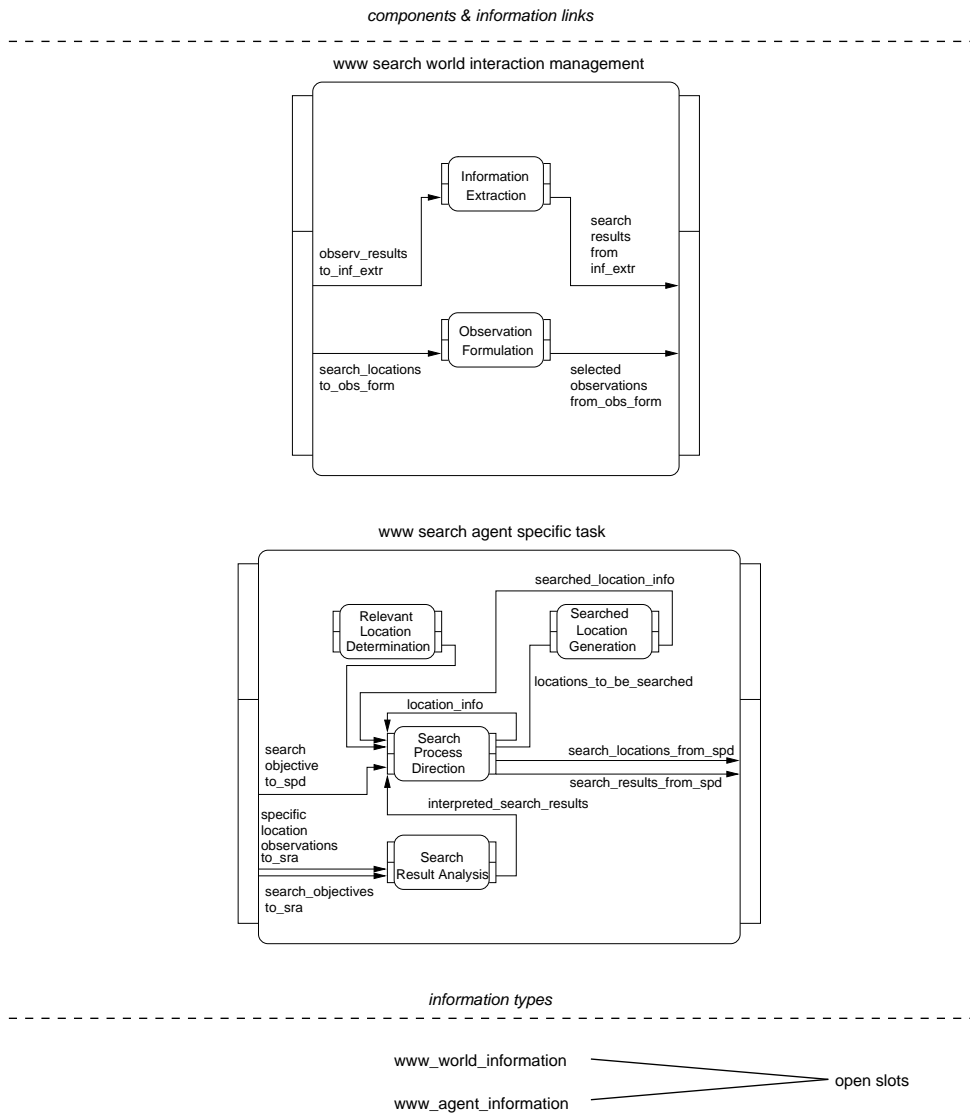


Figure 36. The template information used in the third DOD modification cycle.

Design Process Co-ordination

The co-ordination process decides that RQSM is to continue the design process. DODM and RQSM are informed about the decision.

Requirement Qualification Set Manipulation

The information regarding the open slots communicated by DODM and the RQS information are used by RQSM to construct a new QPS. The QPS contains properties indicating that templates are desired that are *able to provide computer part knowledge* and is also *able to provide computer part location knowledge*. Retrieval objectives are determined. The QPS and the retrieval objectives are communicated to TR.

A (composed) template has been found by TR containing the requested information type templates and the QPS information is used to modify the current RQS. The RQS is commu-

nicated to DODM and DESIGN PROCESS CO-ORDINATION is informed about the progress of the RQSM process.

Design Process Co-ordination

The co-ordination process decides that DODM is to continue the design process. DODM and RQSM are informed about the decision.

Design Object Description Manipulation

The templates found by the template retrieval process (see figure 37) are applied to the design. The *www_search_objects* open slot information type is filled in using *search_objects_for_computer_parts* and the *www_location_info* open slot information type is filled in using *www_locations_for_computer_parts*. The design object description is analyzed to determine if any open slots are still present. No open slots are found and the design object description also satisfies the requirements communicated by RQSM. DESIGN PROCESS CO-ORDINATION is informed that the DODM process is finished and that an agent was successfully configured.

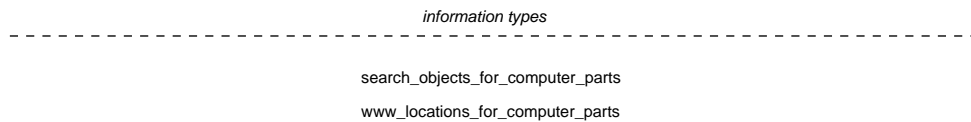


Figure 37. The template information used in the fourth DOD modification cycle.

Design Process Co-ordination

The co-ordination process concludes that the agent design has been completed. The design process is halted.

7.2 The resulting agent

The example agent used in this thesis is a prototype agent for searching within computer stores on the Internet. This section presents an overview of the processes within the prototype agent. The subprocesses of the agent are shown in the figures used in Section 7.1. The USER and EXTERNAL WORLD processes have not been further specified, but are assumed to contain functionality for communicating with the user and the external world (the Internet), respectively.

The search agent, as task, consists of the following steps. The names of the processes responsible for performing the tasks are given within parentheses:

- Receiving a query from the user (AGENT INTERACTION MANAGEMENT).
- Determining a location that is to be searched (RELEVANT LOCATION DETERMINATION, SEARCHED LOCATION GENERATION and SEARCH PROCESS DIRECTION).
- Retrieving the webpage(s) from the location (OBSERVATION FORMULATION).
- Extracting relevant information from the location information (INFORMATION EXTRACTION).
- Determining if the information contains an answer to the query issued by the user (SEARCH RESULT ANALYSIS).
- Communicating an answer to the user (AGENT INTERACTION MANAGEMENT).

A prototype Java implementation of this agent is presented in Appendix C.

8 Implementation

In this thesis, the Java programming language was used to implement a prototype set of classes representing agent elements. These classes were used to create sample templates to simulate

a simple agent construction process. The primary goal of this prototype implementation was to attempt to use the *open slot* concept within Java code.

The classes defined to represent elements of agents are based on the elements used in the DESIRE framework to create agent specifications (see section 4.1).

The implementation presented in this thesis was created for evaluation purposes. The concepts used do not provide the same level of functionality as the DESIRE framework itself. The emphasis of the implementation lies in the attempt to create an implementation capable of providing a way to combine pre-defined Java templates into a functional agent.

Two important classes used to combine templates are the *Admin* class and the *Assembly* class. The *Admin* class is used to centrally register the objects used in the agent. The *Assembly* class uses the *Admin* to find the open slot elements in the templates and connect the elements from other templates with these open slots.

An elaborate description of the classes is given in Appendix A. The prototype implementation is presented in Appendix C. An output trace the prototype is presented in Appendix B.

9 Conclusions and future work

This section presents the final conclusions and a discussion of topics for further research.

9.1 Conclusions

The multi-agent factory model discussed in this thesis presents a possible model for a system capable of configuration-based (re-)design of agents. The emphasis of this thesis is on the design-centre *within* the multi-agent factory. The model for this design-centre is based on a generic model for design, and a refinement of that model, the model for (re-)design of compositional systems. The most important characteristic of the multi-agent factory design-centre is the ability to use templates for agent design. Templates are partial agent descriptions providing a specific level of functionality, while leaving other parts open. The agent templates are available to the design process through an intelligent retrieval process. A conceptual design description of an agent is built by retrieving templates and adding these to the parts of the description that are left open by already inserted templates.

Another process within the multi-agent factory, the *Assembly* process, is able to convert this conceptual design into an operational design, also by making use of the template retrieval process. Each template used by the design process also contains, in addition to the partial conceptual description used during the design process, a partial operational description corresponding to the conceptual description. A prototype implementation is presented, in which the problem of assembling partial operational objects from templates is examined.

The example agent used in this thesis has shown, for a restricted case, that it is possible to create an agent design using the aforementioned template based design process. The Java implementation of the example agent has shown that the *template* and *open slot* notions can also be applied when creating a detailed operational agent specification.

In Section 3, five desiderata are presented concerning the provided functionality of the multi-agent factory. The first and second desiderata indicated that the design process should be able to create an agent despite incomplete or vague specifications, and that an evaluation of the agent creation process should be given. The design process used within the multi-agent factory is based on the generic model for design. One of the types of information available on the output of the generic design model is an assessment of the artefact description with respect to the required functionality. This information could be used to present an evaluation of the agent creation process, provided that the assembly process within the multi-agent factory does not change the functionality of the agent design.

The third desideratum indicates that a multi-agent factory should be able to interact with other similar facilities to exchange design knowledge and template information. The emphasis of this thesis was on the design process within a single multi-agent factory. The interaction between multiple multi-agent factories and template retrieval facilities remains for future research.

The fourth desideratum addresses the ability of a multi-agent factory to store design knowledge about successful agent design descriptions. The multi-agent factory currently uses

a template retrieval process to *retrieve* template knowledge, but could possibly also use it to *store* (partial) agent design descriptions. This topic also remains for future research.

The fifth desideratum presented in Section 3 describes that the conceptual design object description created by the multi-agent factory design process should be usable by the assembly process within the multi-agent factory to create an operational detailed agent description. Although a detailed description of the agent assembly process was not presented, the assembly process was able to assemble an agent using the same templates and open slot information used in the multi-agent factory design process, as demonstrated by the prototype implementation. A further detailed analysis of the agent assembly process remains for future research.

9.2 Future work

The emphasis within this thesis is on the design-centre process in the multi-agent factory, and the interaction of the design process with the template retrieval process. *Factory Management* and *Account Management* are briefly mentioned, and the *Assembly* process is examined by construction of a set of templates in Java. These three processes should be examined and described in more detail.

The main subject in this thesis is the situation of the template retrieval process within the design-centre and the interaction between the design-centre and the template retrieval process. However, the level of interaction described addresses only basic use of templates in the design-centre. A number of issues should be further investigated:

- In the future, the design-centre should be able to engage in an intelligent dialogue with the template retrieval process to find useful and practical agent templates. This higher-level form of interaction between the processes would provide more flexibility.
- A concise process- and knowledge decomposition of the design-centre is not provided in this thesis. Instead, the most important features that are central to a design process which uses a template retrieval process are described. The interfaces of the template retrieval process are currently located in the history management processes within the design process, based on the analogy between *template retrieval* and *design history*. However, other locations for the template retrieval process could also be attempted, possibly based on analogies with other processes.
- The design-centre uses templates to design agents based on the assumption that the retrieved templates match with the templates already used in the design. A practical design-centre should however also be able to use templates which are not 100 % compatible with the existing design. It should be possible for the design-centre to apply low-level conceptual design elements to connect templates together. This type of modification requires extensive knowledge about the changes made to specific templates in the design, as the subsequent application of additional templates could also need modification due to the previously applied modifications.
- The RQSM process within the design-centre currently only uses information from qualified property sets to modify the qualified requirement sets, under the assumption that the most relevant conceptual object properties from the templates are also present in the qualified property sets returned by the template retrieval process. It may be useful for RQSM to examine the conceptual object properties in the template set information communicated by TR, as this will give RQSM access to *all* the property information of the returned templates, allowing RQSM to judge for itself which properties are considered useful.
- It could be useful for the multi-agent factory to have the ability to add results from the design process as templates to the template retrieval process. This would ensure that newly designed (parts of) agents can be used in other cases and also possibly by other multi-agent factories.
- The multi-agent factory design process could benefit from access to the knowledge used by TR. For example, the semantic property networks used by TR could be used by the design process to get insight into the structure of the property knowledge present in TR. This would enable the design process to make better informed decisions concerning the use of TR in the design process.

Acknowledgements

First of all I would like to thank my graduation partner and good friend Hidde Boonstra. The many hours we spent together was time well spent. Many thanks also go to my supervisors, Niek Wijngaards and Frances Brazier, for constantly providing support, inspiration and comments during the project. Furthermore, I would like to thank the second reader, Maarten van Steen, for his comments. I would also like to thank Arno for all his friendship over the course of many years and for his absence during my graduation, which made concentrating on my work a lot easier. Many thanks also go to AI'95 and Esther for making the last five years a very pleasant time. Finally I would like to thank my parents, brother and sister for their support and for making my graduation possible. Anyone who I have failed to mention here and deserves to be in this list: my apologies and thank you.

References

- [mer, 2000] (2000). Merriam-webster online. URL: <http://www.m-w.com>.
- [pat, 2000] (2000). Patterns home page. URL: <http://hillside.net/patterns/patterns.html>.
- [Alexander, 1964] Alexander, C. (1964). *Notes on the Synthesis of Form*. Harvard University Press.
- [Boonstra, 2000] Boonstra, H. M. (2000). Templates for agents: support for automated re-configuration of multi-agent systems. Master's thesis, Vrije Universiteit Amsterdam.
- [Brazier et al., 2000a] Brazier, F., Jonker, C., Treur, J., and Wijngaards, N. (2000a). Deliberate evolution in multi-agent systems. In Gero, J., editor, *Proceedings of the Sixth International Conference on AI in Design, AID'2000*, pages 633–650. Kluwer Academic Publishers.
- [Brazier et al., 2000b] Brazier, F. M. T., Cornelissen, F., Jonker, C. M., and Treur, J. (2000b). Compositional specification and reuse of a generic co-operative agent model.
- [Brazier et al., 1997] Brazier, F. M. T., Dunin-Kepicz, B., Jennings, N., and Treur, J. (1997). Desire: Modelling multi-agent systems in a compositional formal framework. *International Journal of Cooperative Information Systems*, 6(Special Issue on Formal Methods in Cooperative Information Systems: Multi-Agent Systems):67–94.
- [Brazier et al., 2000c] Brazier, F. M. T., Jonker, C. M., and Treur, J. (2000c). Compositional design and reuse of a generic agent model. *AAI*, 14(5):491–538.
- [Brazier et al., 2000d] Brazier, F. M. T., Jonker, C. M., and Treur, J. (2000d). *Design of Intelligent Multi-Agent Systems*. Course Material, department of Artificial Intelligence, Vrije Universiteit, Amsterdam.
- [Brazier et al., 1998a] Brazier, F. M. T., Jonker, C. M., Treur, J., and Wijngaards, N. J. E. (1998a). Compositional design of a generic design agent. In Luger, G. and Interrante, L., editors, *Proceedings of the AAAI Workshop on Artificial Intelligence and Manufacturing: State of the Art and Practice*, pages 30–39. AAAI Press.
- [Brazier et al., 1994] Brazier, F. M. T., Langen, P. H. G. v., Ruttkay, Z., and Treur, J. (1994). On formal specification of design tasks. In Gero, J. S. and Sudweeks, F., editors, *Artificial Intelligence in Design '94*, pages 535–552. Kluwer Academic Publishers.
- [Brazier et al., 1998b] Brazier, F. M. T., Langen, P. H. G. v., and Treur, J. (1998b). Strategic knowledge in design: a compositional approach. *Knowledge-based Systems, (Special Issue on Strategic Knowledge and Concept Formation)*, 11:405–416.
- [Buschmann et al., 1996] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. (1996). *Pattern-Oriented Software Architecture*. John Wiley and Sons Ltd, Chichester, UK, 1996.
- [Dennett, 1987] Dennett, D. (1987). *The Intentional Stance*. MIT Press/Bradford Books, Cambridge, MA.
- [Flanagan, 1999] Flanagan, D. (1999). *Java in a Nutshell*. O'Reilly, third edition.

- [Gabriel, 1996] Gabriel, R. P. (1996). *Patterns of Software: Tales from the Software Community*. Oxford.
- [Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: elements of reusable object-oriented software*. Addison Wesley Longman.
- [Jennings et al., 1998] Jennings, N., Sycara, K., and Wooldridge, M. (1998). A roadmap of agent research and development. *Journal of Autonomous Agents and Multi-Agent Systems*, 1:275–306.
- [Kolodner, 1993] Kolodner, J. L. (1993). *Case-Based Reasoning*. Morgan Kaufman.
- [Schank and Abelson, 1977] Schank, R. C. and Abelson, R. P. (1977). *Scripts, Plans, Goals and Understanding*. Lawrence Erlbaum.
- [Watson and Marir, 1994] Watson, I. and Marir, F. (1994). Case-based reasoning: a review. *The knowledge engineering review*, 9(4):327–354.
- [Wijngaards, 1999] Wijngaards, N. J. E. (1999). *Re-design of Compositional Systems*. PhD thesis, Vrije Universiteit Amsterdam.
- [Wooldridge and Jennings, 1995] Wooldridge, M. and Jennings, N. (1995). Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2).

Appendices

Appendix A contains a description of the classes used to create the prototype Internet search agent. Appendix B presents a sample trace of the prototype Internet search agent. Appendix C contains the actual Java code.

A Java class descriptions

B Example trace

C Java implementation

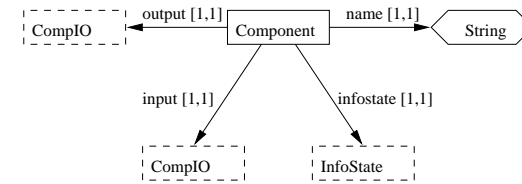
The package `nl.vu.cs.iids.maf.agent` contains the following objects:

- **Component:** A generic component
- **CompIO:** An input or out of a *Component*
- **Level:** A input or output level of a *Component*
- **PrimitiveComp** (extends *Component*): A primitive component
- **KB:** A knowledge base of a *PrimitiveComp*
- **Rule:** A rule for a *KB*
- **RuleExpression:** An expression making the conditions and conclusions of a *Rule*
- **RuleInference:** An inference engine used bij the *KB*
- **SolutionArray:** A collection of possible solutions used bij *RuleInference*
- **UserComp** (extends *Component*): A component which interfaces with the user
- **ComposedComp** (extends *Component*): A composed component
- **TaskControl:** The task control of a *Component*
- **InfoState:** An information state of a *Component*
- **InfoLink:** A generic information link
- **EO_InfoLink** (extends *InfoLink*): An epistemic-object link
- **OA_InfoLink** (extends *InfoLink*): An object-assumption link
- **OO_InfoLink** (extends *InfoLink*): An object-object link
- **InfoAtom:** An information atom from a *InfoState*
- **InfoType:** An information type
- **Sort:** A sort
- **Argument:** An argument for an *InfoAtom* or a *RuleExpression*
- **InfoObj:** A information object for a *Sort* or a *Argument*
- **Relation:** A relation for a *RuleExpression* or an *InfoAtom*

The object descriptions below discuss each object and the relations between them. Each object description consists of:

- A figure displaying the object and its relations with other objects. The following notation is used in the figures: the object that is described is displayed in a solid rectangular box. If a dashed rectangular box is present *inside* the object box, the object is derived from that object. Dashed rectangular boxes that are displayed *outside* the object box indicate other objects. A diamond shaped box indicates a primitive type, such as a string or a boolean type. Arrows indicate relationships between the displayed objects (and types). Each arrow has a name and an arity.
- A short description of the object design.
- A short discussion of possible choices made and alternatives encountered during design and implementation of the object.
- A short description of possible improvements that could be made.

A.1 Component



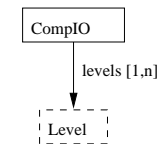
public class Component

Object Design: This class provides basic component variables and methods that are used in all the derived classes. *Component* contains references to an *InfoState* object, an input object and an output object. A component is identifiable by its *name*.

Choices and Alternatives: This class could also have contained implementations for its input and output, but the choice was made to view the input/output as separate objects.

Possible Improvements: At this point, the implementation does not yet contain facilities for setting the task control focus and the extent of its task. For better task control this should be implemented.

A.2 CompIO



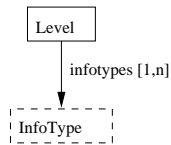
public class CompIO

Object Design: Objects of this class represent the input and output interfaces of components (objects of the class *Component* or one of its derived classes). A *CompIO* consists of a number of information levels (default = 2), which are objects of the class *Level*.

Choices and Alternatives: The i/o interface of the component could also have been implemented directly in the *Component* class, eliminating the need for *CompIO* objects. Because a *Component* always has one output and one input interface, this would not have made the implementation less clear.

Possible Improvements: Objects of this class could benefit from information that would make them more aware of their place, such as if it is an input or an output, which component it is connected to, or which links are connected to which levels.

A.3 Level



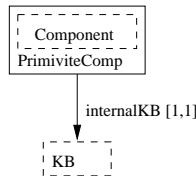
public class Level

Object Design: A level object is part of an input or output object of a Component. Each level contains a list of information types. This list is used to determine which information can pass through this level. Information links connect to an input or output *level* of a component.

Choices and Alternatives: None considered.

Possible Improvements: References to the Rule object are currently stored in a list. At present the rules will always be fired in a fixed order. An improvement would be to select the rules in a random order, to eliminate possible influences to the reasoning process of rule order. Also, the current implementation does not check consistency of the list of rules. At present, it could occur that one rule in the knowledge base derives `vehicle.color(yellow)`, while another rule derives `not vehicle.color(yellow)`. An improvement would be to check for these sorts of inconsistencies (and possible other conflicts in the kb).

A.4 PrimitiveComp



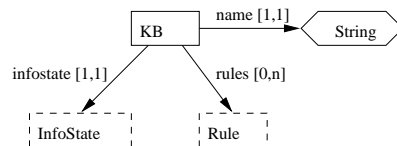
public class PrimitiveComp extends Component

Object Design: This class extends the Component class so that objects can be created that act as components resembling the primitive components used in the DESIRE framework. A primitive-component object contains a reference to a knowledge base that is activated whenever the component itself is activated.

Choices and Alternatives: None considered.

Possible Improvements: None considered.

A.5 KB



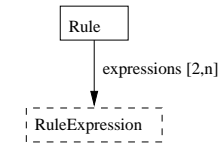
public class KB

Object Design: An object of the KB class represents a knowledge base used by primitive components in the multi-agent system to reason. A knowledge base contains a list of Rule objects and a reference to the information state which the kb will use as its information source and destination.

Choices and Alternatives: None considered.

Possible Improvements: The knowledge base currently does not use the input and output information types of the primitive component it resides in to check if the information it derives is of a valid type. An improvement would be to add this checking. Adding this check would improve stability and increase the ability of the programmer to control the information that is derived in the system.

A.6 Rule



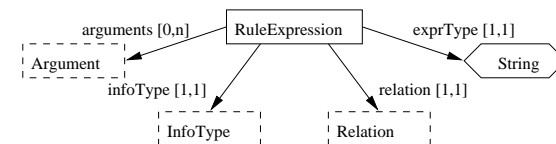
public class Rule

Object Design: A Rule is modelled as a collection of RuleExpressions. A string can be passed to the constructor. This string is parsed and converted to a Rule-object. This parser is considered an auxiliary constructor and should not be trusted to generate a legal Rule-object.

Choices and Alternatives: The distinction between conditions and conclusions is made in the RuleExpression and not in the Rule. Also the truth-value of an expression is stored within the RuleExpression. A possible alternative would be to keep the RuleExpressions simple and make these distinctions within the Rule.

Possible Improvements: None considered.

A.7 RuleExpression



public class RuleExpression

Object Design: A RuleExpression represents a condition or conclusion of a Rule. A RuleExpression has a Relation and Arguments of that Relation. These Arguments can be Constants or Variables (See also Argument).

Choices and Alternatives: The expressions are modelled as Relations with Arguments. This structure is also used to model information atoms of an InfoState. This structure could be reused for both RuleExpression and InfoAtom.

Possible Improvements: None considered.

A.8 RuleInference

public class RuleInference

Object Design: RuleInference is designed to contain a input InfoState and Rule which is combined to an resulting InfoState. The inference uses an SolutionArray to find possible InfoAtoms for the conditions of the Rule. The found SolutionArray is then used to add new InfoAtoms to the resulting InfoState.

Choices and Alternatives: RuleInference is a simple inference engine which is written from scratch. An alternative would be to use a existing inference engine.

Possible Improvements: The use of faster and better inference algorithms.

A.9 SolutionArray

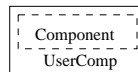
public class SolutionArray

Object Design: SolutionArray is an auxiliary datastructure for RuleInference. It contains an array of solutions. Each solution consists of a set of possible instances of the variables.

Choices and Alternatives: None considered.

Possible Improvements: None considered.

A.10 UserComp



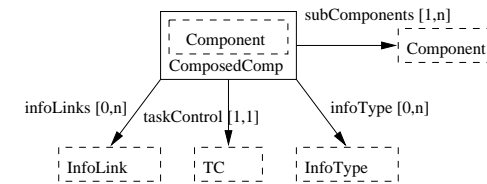
public class UserComp

Object Design: Objects of this class provide an interface to the user. When activated, the components displays its information state on the screen, and provides the user with the ability to add information atoms to the infostate. This component can be used to avoid having to implement complex alternative implementations, such as an interface with the external world. It can also be used when a designed multi-agent system contains a user agent.

Choices and Alternatives: None considered.

Possible Improvements: The current implementation of this component is very basic. An improvement would be to present a more sophisticated interface to the user. It should be possible for the user to add a complete information atom at once, instead of separately adding the information to construct information atoms.

A.11 ComposedComp



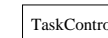
public class ComposedComp extends Component

Object Design: This class extends the Component class to make it suitable to contain sub-component that perform subtasks. An object from this class contains a list with subcomponents, a list with information links that connect these components to each other and to the object itself, a list with information types used by the information links and i/o's of the subcomponents, and a reference to a task control object to direct the activation of the objects contained in this object.

Choices and Alternatives: None considered.

Possible Improvements: None considered.

A.12 TaskControl



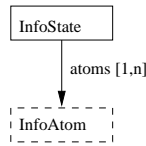
public class TaskControl

Object Design: A task-control object represent the task control of a composed component. It contains a list of components and information links. It is used to activate the objects listed in the order in which they appear in the list. A special *stop* indicator can be inserted into the list. When the task control object reaches this indicator it halts at that position. When the task control is reactivated, it resumes at that position. When the end of the list is reached, it loops back to the beginning.

Choices and Alternatives: A list implementation was used because no advanced task control knowledge was needed for the prototype agent that was built. The task control object is also not able to check if tasks have succeeded or failed.

Possible Improvements: in the DESIRE framework the task control of a component consists of a set of rules in a knowledge base. These rules provide much more control over the activation sequence of the components and links than a simple list. Also, in DESIRE, task control knowledge is able to check if components have succeeded or failed in their tasks, and task control is also able to change the focus and extent of components. If this implementation is to be used to build more complex agents, this functionality is certainly desired.

A.13 InfoState



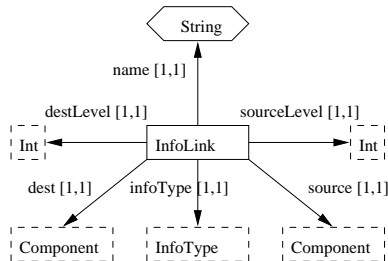
public class InfoState

Object Design: An information state consists of one or more information atoms combined with their truth value and a status describing if the information atom is an input atom, an internal atom or an output atom. The information state always contains the internal information atom *true* of the information type *truthvalue*.

Choices and Alternatives: The truth value and status of an information atom are described in the information state. An alternative is to represent the truth value and status within the information atom. This alternative is not used because an information atom can be present in different information states with different truth values. An information state may contain the same information atom with different truth values if they do not have the same status.

Possible Improvements: The methods to get or find a information atom within the information state use linear search algorithms. This could be improved by choosing a different representation for the list of information atoms in order to allow faster search algorithms.

A.14 InfoLink



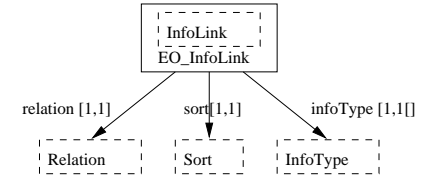
public class InfoLink

Object Design: An info link has a name and consists of source component, an indication of the source level, a destination component, an indication of the destination level and possibly an information type. The generic class InfoLink is not meant to be used. Always use one of its subclasses (e.g. OO_InfoLink, OA_InfoLink, EO_InfoLink).

Choices and Alternatives: The information type is used to filter the information flowing through the link. This filtering is done by the link itself but it could also be done by the methods which returns the information through an input or an out. Although this might be faster, because the list of information types is only processed once, this method is not used, because it is the information link which only lets information of a certain type through.

Possible Improvements: None considered.

A.15 EO_InfoLink



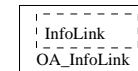
public class EO_InfoLink extends InfoLink

Object Design: An EO_InfoLink is a specified version of the generic information link InfoLink containing the extra information needed to make the transformation from object level to meta. This information consists of a relation, a sort and an information type.

Choices and Alternatives: The sort of the encapsulated relation has to be set by the caller of the information link before the link is activated. This is used as sort of the meta description. This implementation is chosen because no meta description exists within the information type object and because the EO and OA information links are only used to transport information from between levels. An alternative would be to further implement the InfoType object and use the information then available.

Possible Improvements: The information needed to make the object description should be available from the InfoType object.

A.16 OA_InfoLink



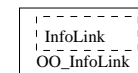
public class OA_InfoLink extends InfoLink

Object Design: The OA_InfoLink only contains the objects contained by the class it extends (i.e. InfoLink). It does however contain information on how to transform an information type from a meta level to object.

Choices and Alternatives: None considered.

Possible Improvements: None considered.

A.17 OO_InfoLink



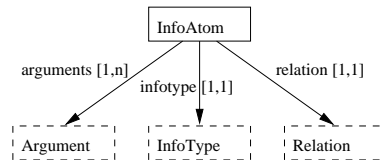
public class OO_InfoLink extends InfoLink

Object Design: Apart from the objects contained by the InfoLink class the OO-InfoLink does not contain any other objects.

Choices and Alternatives: None considered.

Possible Improvements: None considered.

A.18 InfoAtom



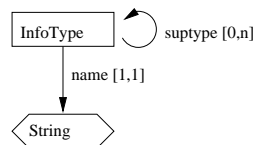
public class InfoAtom

Object Design: Information atoms are objects of the InfoAtom class. These objects represent units of information that (knowledge bases of) components use to reason with. An information atom has a type, a relation and a list of argument objects. Information atoms can be used by knowledge bases to derive new information atoms which are then inserted into the information state of the component. Information atoms are the information elements that flow through the multi-agent system.

Choices and Alternatives: Information atoms are not aware whether they are input information atoms or output information atoms. They also do not know their own truthvalue. This information is stored inside the infostate in which they exist. When copying information atoms it could be more efficient to store this information inside the atoms itself, instead of looking this up in the infostate. This could also be beneficial when information atoms occur in more places than information states alone (such as information link buffers).

Possible Improvements: None considered.

A.19 InfoType



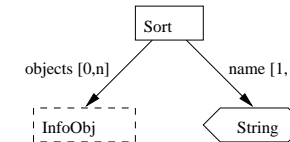
public class InfoType

Object Design: These objects are used to structure the information available to the multi-agent system. All information atoms present in the information states of the components in the multi-agent system are of a specific type of information. This structuring of information allows for structuring of the information flow within the multi-agent system. For example, information links have the ability to only pass information of a certain type. Information types can contain additional information type objects. These included information types are then available through the information type in which they are included.

Choices and Alternatives:

Possible Improvements: In the DESIRE framework, information types are also used to define the knowledge available to the multi-agent system. Information types defined in DESIRE contain SORTS, which in turn contain ground atom definitions. This implementation does not implement these structures. This means that there is no checking on the validity of the information atoms in the system, and also not on the rules in the knowledge bases.

A.20 Sort



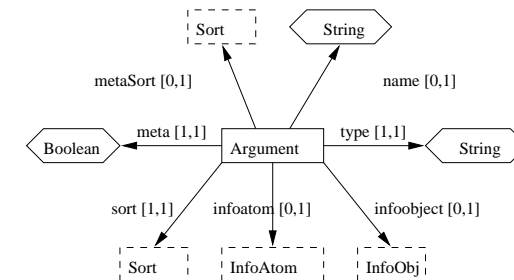
public class Sort

Object Design: A Sort object represents a sort as used in the DESIRE framework.

Choices and Alternatives: None considered.

Possible Improvements: None considered.

A.21 Argument



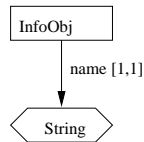
public class Argument

Object Design: An argument is used as an argument of a relation. An argument can either be a variable or a constant. An argument is used in an InfoState as a constant only and in a rule or an inference as a variable or a constant. When an argument is a variable it consists of an ArgumentName and a Sort. When an argument is a constant it consists of an InfoObj and a Sort, or a MetaSort and an InfoAtom when it is a meta argument.

Choices and Alternatives: An argument is not bound to a relation because more relations can use the same argument. An alternative would be to give a relation a list of arguments. An argument can be either an object argument or a meta argument. When an argument is a meta argument it can contain an InfoAtom (i.e. an relation with arguments) but it can still be used as an object argument. This way the inference engine does not have to differentiate between object and meta arguments. This is done to make implementation easier.

Possible Improvements: Variables and constants could be made different object types extending the argument object. Furthermore a different object type could be made for a meta argument. This would make the argument code more intuitive because the methods don't have to execute different parts of the code for different situations. This would also force other objects to differentiate between object and meta arguments resulting in code which does not rely on the argument to mask that it might be a meta argument.

A.22 InfoObj



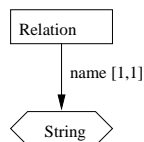
public class InfoObj

Object Design: Objects of the class InfoObj represent the ground atoms of information used in information atoms. It are the most basic object level information objects available. Examples are units of time or colors.

Choices and Alternatives: None considered.

Possible Improvements: None considered.

A.23 Relation



public class Relation

Object Design: A Relation object consists only of a name represented by a string. It represent the relation used in the DESIRE framework.

Choices and Alternatives: This implementation does not view the relation as an object containing arguments in a certain order. The relation and its arguments are bound together in objects encompassing the relation, such as an information atom or a rule expression.

Possible Improvements: None considered.

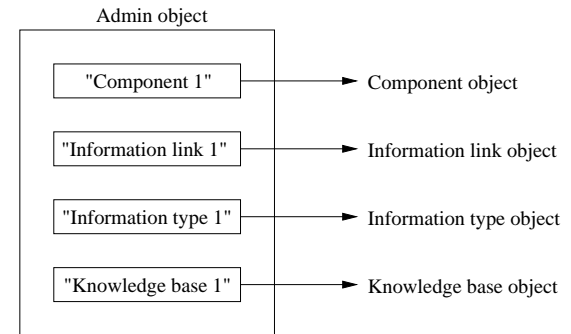
The JAVA-Template objects

The package nl.vu.cs.iids.maf.agent contains the following objects:

- **Admin:** an administration for keeping track of the objects in the multi-agent system.
- **Template:** provides methods used for defining templates.

- **Assembly:** provides functionality to insert templates into open slots in the multi-agent system.
- **Tools:** provides frequently used methods, such as alternative printing functions.

A.24 Admin



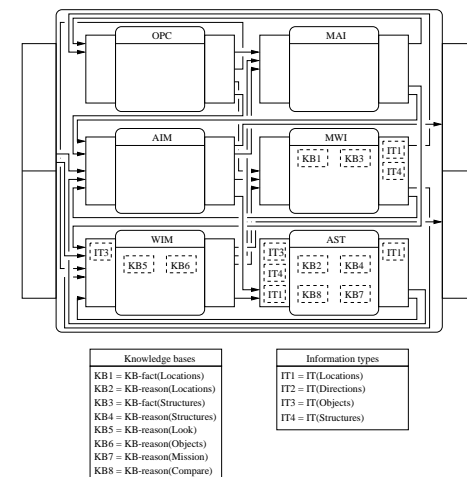
public class Admin

Object Design: Only one admin object is used per multi-agent system. It is used to store references (by name) to components, information links, information types and knowledge bases.

Choices and Alternatives: The Admin is implemented using the java Hashtable class.

Possible Improvements: None considered.

A.25 Template



public class Template

Object Design: The template class is used as the basis for all template definition classes. It provides basic functions to add objects like links and components to a template. These functions automatically register the objects to the Admin object.

Choices and Alternatives: None considered.

Possible Improvements: None considered.

A.26 Assembly

public class Assembly

Object Design: The assembly object is used to link templates together. It is capable to insert components into open slot components, information types into open slot information types and knowledge bases into primitive open slot components with an open slot knowledge base.

Choices and Alternatives: None considered.

Possible Improvements: The assembly is not able to insert knowledge bases into components that already have a knowledge. It could be beneficial to add rules to an already existing knowledge base. For example, a component could already have basic, commonly used rules, and these could be further completed by adding domainspecific rules.

A.27 Tools

public class Tools

Object Design: This class contains commonly used methods (currently some printing functions) which can be used by the other classes.

Choices and Alternatives: None considered.

Possible Improvements: None considered.