

Templates for Agents

Support for automated re-configuration of multi-agent systems

H.M. Boonstra

in co-operation with D.G.A. Mobach

6th December 2000

Abstract

This thesis presents a model for templates which can be used for component based re-design of multi-agent systems. This model is tightly interconnected with the multi-agent factory but the template database should be usable by other systems. (***)

Master's thesis of H.M. Boonstra

Student number: 0630136

Email: hmboonst@cs.vu.nl

Intelligent Interactive Distributed Systems
Department of AI
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	5
1.1	The Multi Agent Factory and Template Retrieval	5
1.2	Assumptions and Requirements on Template Retrieval	6
2	Relevant Literature	7
2.1	Design Patterns	7
2.2	Case-Based Reasoning	8
3	MAF in general	9
3.1	Introduction to the MAF	9
3.2	Components of the MAF	10
4	Modelling Paradigms	10
4.1	JAVA	11
5	Design model	12
5.1	Generic Design Model	12
5.1.1	Subprocesses of design	12
5.1.2	Subprocesses of RQSM	15
5.1.3	Subprocesses of DODM	15
5.2	Design Model for Template Retrieval	16
5.2.1	Template Retrieval I/O	16
5.2.2	Subprocesses of Template Retrieval	18
6	Properties of design objects	19
6.1	Property Structures	19
6.2	Semantic property networks	20
7	The Templates	21
7.1	Template Structure	22
7.1.1	Characteristics	22
7.1.2	Pre-conditions	22
7.1.3	Template Object Properties (TOP)	23
7.1.4	Template Object Description (TOD)	23
7.2	Template Types	24
7.3	Template Object Types	25
7.3.1	Component Templates	25
7.3.2	Knowledge base Templates	25
7.3.3	Information Link Templates	25
7.3.4	Information Type Templates	25
7.4	Template Combinations	26
7.5	Template Use	26
8	Matching	27
8.1	Queries	27
8.2	Matching Heuristics	28
8.3	Matching without interpretation	28
8.4	Matching with variable forms of interpretation	28

9	Implementation	29
9.1	The Agent Top-level	29
9.1.1	Agent incorporation	30
9.1.2	Agent to Agent communication	32
9.1.3	Agent to External World interaction	33
10	Template use	33
10.1	Example of template use	33
11	Conclusions and Future work	34
11.1	Conclusions	35
11.2	Future Work	35
A	Generic Search Example	38
A.1	Agent template	38
A.1.1	Information Types	38
A.1.2	Agent	38
A.2	Generic World Search Template	41
A.2.1	Information Types	41
A.2.2	Generic World Search Agent	42
A.3	Web Search Template	46
A.3.1	Information Types	46
A.3.2	Generic Web Search	48
A.3.3	Generic Web Search AST	49
A.3.4	Generic Web Search WIM	52
A.3.5	Generic Web Search Agent Info	54
A.3.6	Generic Web Search World Info	56
A.4	Computer Parts Template	58
A.4.1	Information Types	58
A.4.2	Computer parts	58
A.4.3	Searched Locations Generation KB	59
A.5	Relevant Location Determination KB	60
A.5.1	Search Objects for Computer Parts	61
A.5.2	Web Locations for Computer Parts	62
B	Java Implementation	62
B.1	The JAVA-Agent objects	62
B.1.1	Agent	63
B.1.2	Argument	64
B.1.3	CompIO	64
B.1.4	Component	65
B.1.5	ComposedComp	65
B.1.6	EO_InfoLink	66
B.1.7	InfoAtom	66
B.1.8	InfoLink	67
B.1.9	InfoObj	67
B.1.10	InfoState	68
B.1.11	InfoType	68
B.1.12	KB	69
B.1.13	Level	69
B.1.14	OA_InfoLink	70
B.1.15	OO_InfoLink	70
B.1.16	PrimitiveComp	70
B.1.17	Relation	70

B.1.18	Rule	71
B.1.19	RuleExpression	71
B.1.20	RuleInference	71
B.1.21	SolutionArray	72
B.1.22	Sort	72
B.1.23	TaskControl	72
B.1.24	UserComp	73
B.2	The JAVA-Template objects	73
B.2.1	Admin	73
B.2.2	Template	73
B.2.3	Assembly	74
B.2.4	Tools	74
C	Overview of Properties	74
C.1	Preconditions	74
C.2	Template Properties	74
C.3	Structural Properties	75
D	Overview of Terms	77
E	Output trace	77

1 Introduction

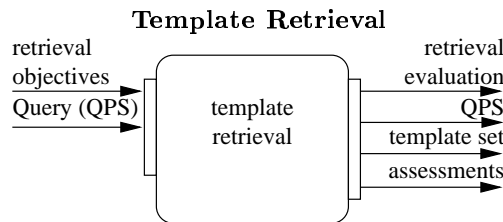
Computer and communication infrastructures are becoming more complex. The amount of information available within computer systems is becoming increasingly difficult to manage. Agents, capable of performing specific tasks with reasonable autonomy, can be of assistance by taking some of the work out of human hands. These agents, however, are complex, knowledge intensive, software entities. Designing agents for specific tasks is a difficult process. The task an agent performs 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., 2000d] and [Brazier et al., 2000b] have been developed which can be refined for specific agents within specific tasks in specific domains.

As the demand for agents with various abilities grows, a system is needed capable of providing support for making the task of developing agents easier. Agents only then have to be constructed when they are needed, and may be destroyed when their job is done. Agents could be tailored specifically for their tasks and domains, and when demands change or new tasks emerge, the existing agents are replaced by more capable agents, or the existing agents are remodeled to fit the new situations.

The Multi-Agent Factory (MAF) presents a means to (re)design and (re)construct agents for specific tasks and domains. The agents are designed using a configuration-based re-design process. Agent-templates, the building blocks which are to be combined, are made available by an intelligent retrieval system. This retrieval system, called `TEMPLATE RETRIEVAL`, and templates are the main subject of this thesis. The goal is to give a model for templates and a retrieval mechanism that can be used in a (re)configuration model for design as described in [Mobach, 2000].

The structure of this thesis is as follows: section 2 discusses literature relevant to the main subjects discussed in this thesis: multi-agent systems and the reuse of design knowledge. Section 3 presents an overview of the multi-agent factory. The two modeling paradigms used in this thesis, declarative modeling of processes (`DESIRE`) and object oriented modeling (`JAVA`) are described in section 4. The conceptual models of generic design and template retrieval are discussed in section 5. Section 6 describes properties used in templates. Section 7 describes the templates, their structure and their usage. In section 8 matching used to retrieve templates is discussed. A discussion of the implementation made to assess the feasibility of combining templates is given in 9. Section 10 discusses the possible use of templates and template retrieval and the final conclusions are given in section 11.

1.1 The Multi Agent Factory and Template Retrieval



The re-design process of the MAF [Mobach, 2000] is modeled as a configuration task, employing a collection of well-defined elements. These elements need to be modeled, collected, stored and retrieved. To support the re-design process the notion of templates is introduced in this thesis. A template in configuration based re-design is a partial solution to a design problem. Partial solutions can be combined by the configuration process eventually yielding a complete (i.e. not partial) description of a specific agent. The partial solution in a template is described at a conceptual level as well as at an operational level. The conceptual description is used for retrieval and may be used to reason about partial solutions. The operational description supports assembly of a working program on completion of design. Templates are stored in a database. To retrieve templates from the database a retrieval mechanism is used which allows queries based on conceptual descriptions

within the templates. The retrieval mechanism together with the database is called *Template Retrieval* (TR).

Templates 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 has a dual description. A conceptual description and an operational description. Combinations 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.

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

1.2 Assumptions and Requirements on Template Retrieval

In this thesis some assumptions and requirements are made on template retrieval and the re-design process using template retrieval. This section discusses these assumptions and requirements.

Task Domain Although the task domain is left open, the modeled object has to have a compositional structure.

Communication Some assumptions are made about communication between the re-design process and template retrieval. Queries must be formulated correctly and the terms used should be known if an retrieval operation is to be successful.

Known properties Properties used to query a template database exist either in the template descriptions or in the semantic property networks belonging to the retrieval mechanism addressed (i.e. the properties are known to the retrieval mechanism used to query that database).

Open Slot Fulfillment It is assumed that each re-design process is capable of analyzing the open slots and filling them with suitable templates or code.

Open Slot Properties Unless the re-design process is able to derive on its own what is to be placed in the open slots, properties have to be defined describing the templates or modules needed to fill these open slots.

Contradicting Properties A template should not contain contradicting properties and combining templates should not generate contradicting properties within the design.

Reflection When an object is designed at a conceptual level a mapping can be made to the design at operational level.

2 Relevant Literature

2.1 Design Patterns

This part is the same as in [Mobach, 2000]

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*, 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.

2.2 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 CBR and the MAF:

- 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 MAF in general

3.1 Introduction to the MAF

This part is the same as in [Mobach, 2000]

The multi-agent factory is a facility that 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 Components of the MAF

This part is the same as in [Mobach, 2000]

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.

4 Modelling Paradigms

This part is the same as in [Mobach, 2000]

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., 1997a]. 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. 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 2 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 an examples of information links: the information link *hypotheses* is used to transfer information on generated hypotheses from HYPOTHESIS DETERMINATION to HYPOTHESIS VALIDATION. The information link *validation results* is used to transfer information on validations of the generated hypotheses from HYPOTHESIS VALIDATION to 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.

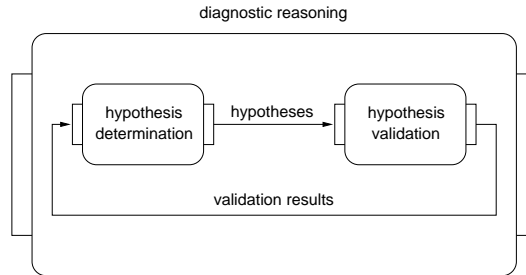


Figure 2. An example of process composition.

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.

4.1 java

This part is the same as in [Mobach, 2000]

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 JVMs is written can execute Java programs. 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 model

This section discusses the generic design model and how this model can be adopted for template retrieval. Section 5.1 discusses the GENERIC DESIGN MODEL and section 5.2 discusses the adoption of this model for TEMPLATE RETRIEVAL.

5.1 Generic Design Model

This part (5.1) is the same as in [Mobach, 2000] but should be summarized

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., 1998b]. A number of generic models have been designed using this framework. Two examples of generic agent models can be found in [Brazier et al., 2000b] and [Brazier et al., 2000a]. 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., 2000c]. 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.

5.1.1 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 what the next search step should be. 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 next step in the overall design process.

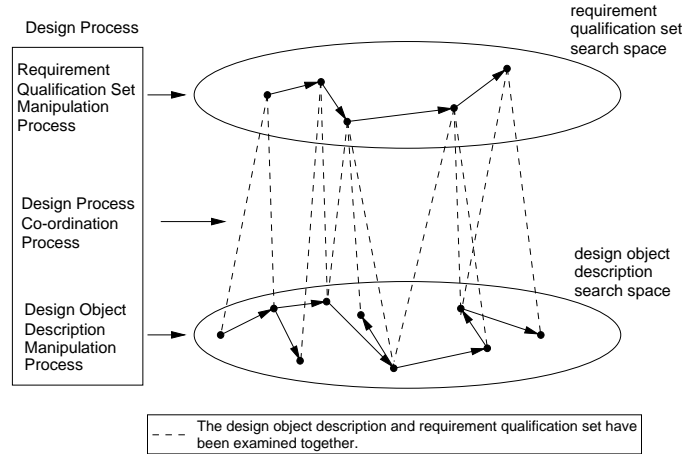


Figure 5. The role of the processes in the design process.

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.

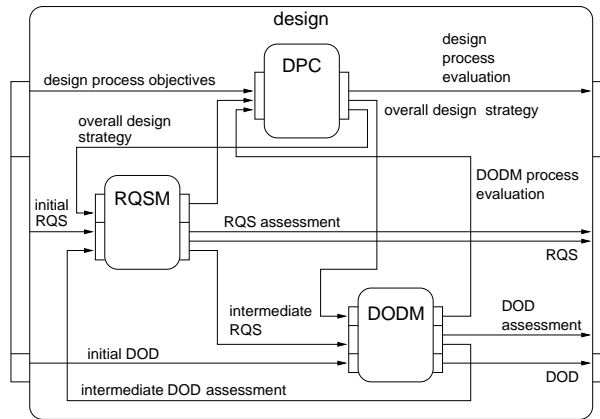


Figure 6. Top-level view of the design task process decomposition.

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.

5.1.2 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.

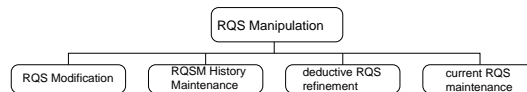


Figure 7. Subprocesses distinguished in RQSM according to the generic design model.

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.

5.1.3 Subprocesses of DODM

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

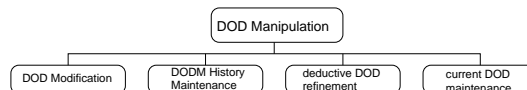


Figure 8. Subprocesses distinguished in DODM according to the generic design model.

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.

5.2 Design Model for Template Retrieval

The process model of `TEMPLATE RETRIEVAL` is based on the model for re-design of compositional systems [Wijngaards, 1999]. The model for re-design of compositional systems describes a system that is capable of designing an (composed) object based on desired function(s) of that object. This model is adopted because retrieval of templates closely resembles a compositional re-design process. A compositional re-design process attempts to find a possible description of a design object that fulfills a set of requirements. Similarly a template retrieval process attempts to find a possible combination of templates that fulfills a set of new requested properties. Both processes produce a composed object based on requested functionality. When a design object description is designed by a re-design process requirements are refined, dropped and reinstated using strategic knowledge until a design object description that fulfills the resulting set of requirements. Analogue to this a combination of templates can be constructed by refining, dropping and reinstating requested properties using strategic knowledge.

Given that the generic model for re-design of compositional systems is used for a specific domain (i.e. the design of templates) and the generic model can be refined, it may form the basis for a model for `TEMPLATE RETRIEVAL`. This involves specializing the processes and instantiation of the knowledge structures. The information used in a re-design process is mapped to information used for the retrieval of templates. The `DESIGN PROCESS OBJECTIVES` used in the re-design model is mapped to `RETRIEVAL PROCESS OBJECTIVES` in the template retrieval model. `REQUIREMENT QUALIFICATION SETS` are mapped to `QUALIFIED PROPERTY SETS`. And `DESIGN OBJECT DESCRIPTIONS` are mapped to `TEMPLATE SETS`.

A mapping between the top-level processes of the generic design model [Brazier et al., 1998a] and processes of a model for `TEMPLATE RETRIEVAL` is made. The process `DESIGN PROCESS CONTROL` is mapped to `RETRIEVAL PROCESS CONTROL`. The process `REQUIREMENT QUALIFICATION SET MANIPULATION` is mapped to `QUALIFIED PROPERTY SET MANIPULATION`. And the process `DESIGN OBJECT DESCRIPTION MANIPULATION` is mapped to `TEMPLATE SET MANIPULATION`.

This section describes input/output information of a model for template retrieval (section 5.2.1). A refinement of the top-level processes of the model for re-design is presented in ??.

5.2.1 Template Retrieval I/O

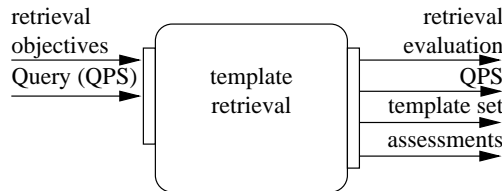


Figure ?? . Template retrieval information flow

`TEMPLATE RETRIEVAL (TR)` has a slightly different input and output interfaces than the model for re-design of compositional systems [Wijngaards, 1999]. In its input interface TR has *retrieval objectives* and a *query*. In its output interface *retrieval evaluations*, a *qualified property set*, *assessments* and a *template set*. In comparison to the re-design model an initial design object description is missing in the input interface of TR. Although the templates, used by configuration based re-design [Mobach, 2000] in an earlier stage of the design process, may play an important role in finding appropriate templates it is thought to be sufficient to refer to those templates within a Qualified Property Set (QPS). This thought is based on the assumption that the templates used in earlier stages of design are available to TR by retrieving them from its own database. This might, however, not be the case when an agent who originally was designed by another MAF is being re-designed. In that case TR would have to retrieve the originally used templates from the other TR¹. The assessments are included in the output of TR because this information is needed by the re-design process to reason about the result of a query (e.g. choose between alternatives returned).

¹This co-operation between multiple template retrievals is not part of this thesis.

Retrieval objectives are used by the re-design process to steer the retrieval process by formulating additional requirements that are not directly related to the templates or the objects described within the templates. Such additional requirements may be about the number of templates retrieved, the desirability of alternatives, the available time for the retrieval or about the level of interpretation. Especially the level of interpretation is important because this states if `TEMPLATE RETRIEVAL` is allowed to deviate from the query if no straight match is found (see section 8.4). The retrieval objectives together with the qualification of the requested properties help the retrieval process to decide when it has found an answer to the query. It also allows the retrieval process to assess the quality of possible answers in order to choose between them and to construct an evaluation.

Retrieval evaluation is used to provide feedback to the re-design process about the retrieval process. This can be a simple *succeeded* or *failed* but further information can be given about this success or failure to explain or shade the success or failure (e.g. *succeeded* but without fulfilling any of the desired properties which were qualified as being soft). The information in the retrieval evaluation can be used by the re-design process to decide if additional information or reformulation of the query is needed. This might be useful even in the case that `TEMPLATE RETRIEVAL` succeeded in retrieving one or more templates. Either because success was marginal or because a better success can be achieved when certain information is available. The re-design process then can decide to make the needed information available or remember to perform the query again when this information becomes available.

Qualified property set are used to communicate a request to `TEMPLATE RETRIEVAL`. The QPS contains a set of requested qualified properties. These qualified properties are used to retrieve a template. A qualification is used to associate a level of desirability with the requested properties. Based on this qualification `TEMPLATE RETRIEVAL` can decide which properties have to be fulfilled in order to succeed and which properties may be dropped. Using qualifications, comparisons can be made between different solutions when different numbers of requested properties are fulfilled.

The template set is a set of one or more templates. When more templates are returned these are alternatives for each other. This means that each of the templates in a template set fulfills the minimal set of requested properties posed by the re-design process in the query. The templates in the set may however differ from each other with respect to the properties that were requested as being soft (see section 8). In combination with the assessments made about the templates the template sets allow the re-design process to choose between alternatives.

A template within a template set can be a composed template. When TR cannot find a single template to fulfill the requested properties, and if it is allowed to interpret the query it tries to find combinations of templates. These combinations are combined in a single template so that properties can be defined over the combination.

Assessments consist of two subparts. One part stating assessments about the qualified property set (QPS) returned (see section 8) and one part stating assessments about the template set returned. Assessments about the QPS relate the initial QPS posed by the re-design process with the resulting QPS returned by `TEMPLATE RETRIEVAL`. If, for example, the re-design process asked for *able to (perform(search))* and `TEMPLATE RETRIEVAL` returned *able to (interact with world)* and *able to (analyse (result))* these returned properties could be related to the requested properties using QPS assessments. Assessments about the TS relate the returned set of templates to the returned QPS. Each of the alternatives within the template set is scored against the QPS by this relation which allows for easy comparison of the found alternatives.

5.2.2 Subprocesses of Template Retrieval

The generic model for re-design of compositional systems distinguishes three processes within design process: *Design Process Co-ordination* (DPC), *Design Object Description Manipulation* (DODM) and *Requirement Qualification Set Manipulation* (RQSM). DPC is responsible for determining the steps taken in a design process. DODM is responsible for manipulating a description of a design object in order to find a design object description fulfilling a set of requirements. RQSM is responsible for manipulating a set of requirements provided by RQSM in order to find a set of requirements for which DODM can find a design object description. The design process is refined for the retrieval of templates. Design process co-ordination is therefore renamed to *Retrieval Process Control*. A design object description is a set of templates and therefore DODM is renamed to *Template Set Manipulation*. A set of qualified requirements is a set of qualified properties in this instantiation and therefore RQSM is renamed to *Qualified Property Set Manipulation*. Figure ?? shows an instantiated version of the top-level view of the re-design task process decomposition.

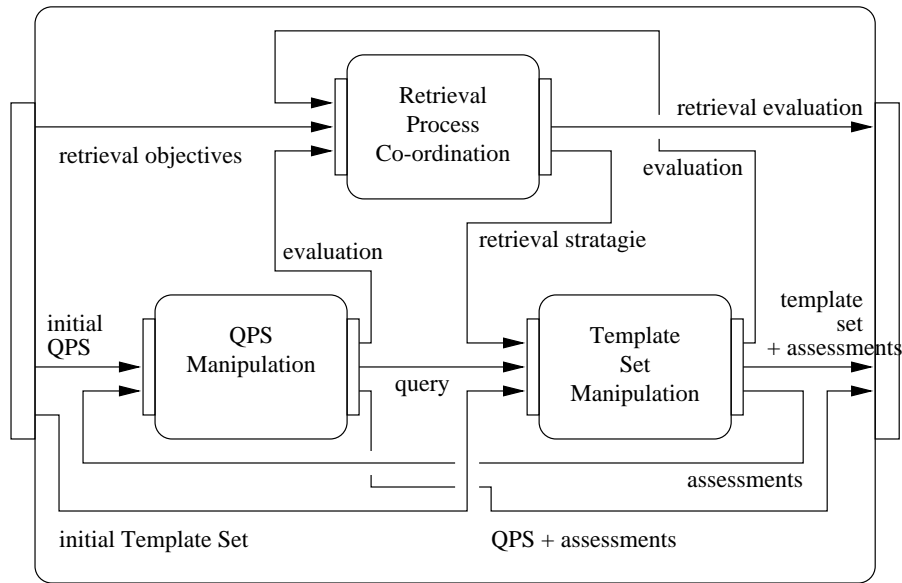


Figure ?. Template retrieval process decomposition

In the following paragraphs the three top-level processes of TEMPLATE RETRIEVAL are discussed:

Retrieval Process Control is responsible for directing the process of retrieving templates. The main goal of RPC is to control the overall retrieval strategy. Based on the retrieval objectives in the input interface of TEMPLATE RETRIEVAL in combination with process evaluations received from QPSM and TSM, RETRIEVAL PROCESS CONTROL (RPC) determines a next step for the retrieval process. RPC monitors the overall retrieval strategie and determines if the retrieval process is satisfying the overall retrieval objectives. RPC produces retrieval evaluations describing the success of the retrieval process and send overall retrieval strategies to the manipulation processes QPSM and TSM.

Qualified Property Set Manipulation is responsible for analyzing, interpreting and manipulating qualified property sets. The main goal of QPSM is manipulating the initial QPS into a QPS for which a TEMPLATE COMBINATION can be found. In its input interface QPSM has an overall retrieval strategy received from RPC, a initial qualified property set and assessments from TSM. Based on the information on its input QPSM tries to manipulate the initial QPS into a set of qualified properties for which TSM can design a combination of templates that fulfills this set of qualified properties. In its output interface it has process evaluations, an intermediate QPS, QPS assessments and a final QPS.

Template Set Manipulation is responsible for retrieving and combining of templates. The main goal of TSM is to retrieve and combine templates into `TEMPLATE COMBINATIONS`. In its input interface TSM has an overall retrieval strategy received from `RPC` and an intermediate QPS received from `QPSM`. Based on this information TSM matches the requested properties from the intermediate QPS and tries to retrieve and combine one or more templates into a `TEMPLATE COMBINATION` that fulfills the request. In its output interface TSM has process evaluations, intermediate `TEMPLATE COMBINATION` assessments, a `TEMPLATE COMBINATION` and final `TEMPLATE COMBINATION` assessments.

6 Properties of design objects

Property: 'a quality or trait belonging and especially peculiar to an individual or thing'
- Merriam-Webster -

This section discusses the properties used in templates. 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*. Both template object properties and pre-conditions are described in the following paragraphs.

The structure of properties is described in the following sections. First the simple structure of property refinements is discussed in section 6.1. After that the structure of semantic property networks is discussed in section 6.2.

Template object properties describe a partial design object in a template. There are two types of template object properties. The first type describes a template at a conceptual level (*conceptual object properties*) and the other describes a template at an operational level (*operational object properties*). Both types of template object description have two parts. One part describes the actual partial object and the other part describes the open slots within the object.

The *object properties* are descriptive properties that describe what the object described in the template can do. For instance that it is able to sort an array of Strings alphabetically. These properties do not all have to be actual *capabilities* of the template object. Some properties may state *abilities* of the template object. These abilities are things that the template object will be able to do if the open slots are filled accordingly to their properties. The object properties allow template retrieval to find templates which fulfill the query (i.e. requested properties) posed by the re-design process by matching the properties from the query with the object properties (see section 8). They allow reasoning about the (partial) design object and about the functionality of combinations of templates.

The *open slot properties* are prescriptive properties and describe what the parts placed in the open slots should be able to do in order to make the combination work. For instance completing a sort with domain information needed to reason about the domain. The open slot properties provide the re-design process with knowledge about what it has to complete.

Pre-conditions describe the context in which a template can be used. These properties are prescriptive and therefore describe what the context should be and/or what it should be able to do in order to make the template work. For instance the type of information that it needs to deduct its conclusions.

6.1 Property Structures

Properties can be structured by linking a property with its refinements (see fig. ??). These refinement structures are similar to the structures used in [Brazier et al., 2000d] and [Wijngaards, 1999].

The association between a property and its refinements is either an AND or an OR. If a refinement is an AND this means that:

- if the more generic property is present the refined properties are also present
- if all the refined properties are present the more generic property is also present

If a refinement is an OR this means that:

- if the more generic property is present at least one of the refined properties is also present
- if at least one of the refined properties is present the more generic property is also present

Refinement of properties can either be specializations or realizations.

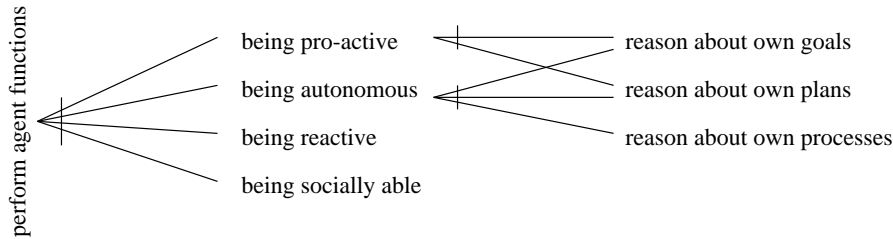


Figure ?? . A sample of a property structure

Specialization When a property is refined through specialization the property is associated with one or more properties that are more specific. An example of refinement through specialization is the refinement of *being weak agent* into *being pro-active*, *being autonomous*, *being reactive* and *being socially able* (see fig. ??).

Realization When a property is refined through realization the property is associated with one or more properties that effectuate the more generic property. These effectuating properties are the properties that actually have to be present in order to allow the more generic properties to be present. An example of combining refinement through realization is the refinement of *active observation in (Internet world)* into *generating requests in (HTTP)* and *parsing (HTML)*.

6.2 Semantic property networks

There are multiple dimensions in which properties can be refined. One dimension is refinement of a generic property into more specific with respect to the abilities and capabilities. An example would be the refinement of *being weak agent* into *being pro-active*, *being autonomous*, *being reactive* and *being socially able*. Another example would be the refinement of *able to search* into *able to direct search*, *able to focus search* and *able to perform search*. There is, however another dimension in which properties can be refined into more specific properties with respect to the task domain. Knowledge about a task domain is used to acquire a set of property structures for a search task domain. An example of this would be the refinement of *being weak agent* into *being search agent* or *being diagnostic agent*. Another example is the refinement of *able to search* into *able to search the web* or *able to search dbs*.

The idea behind the multiple refinement dimensions is that different sets of property structures may be applicable depending on the knowledge that is available about an object that is being designed or the stage in which a design process is situated. With these different sets of property structures different sets of templates can be associated appropriate for that stage of a design process or consistent with the available knowledge (e.g. about the task domain).

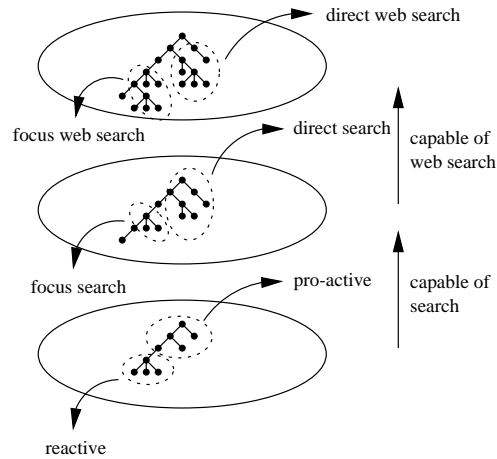


Figure ???. A example of a semantic property network

The semantic property networks are useful for the process of retrieving templates when there is no template that has exactly the same properties as required by the query. Alternatives or possible combinations for the required property can be found by traveling in an upwards (more generic) or in a downward (more specific) direction through the semantic property network (see section 8.4 for an detailed description of matching with interpretation).

7 The Templates

A template is a description of a (partial) design object. A design object description can be partial in the sense that the description can be incomplete, leaving 'holes' in the description. These holes are called *open slots*. The open slots can be completed in different ways allowing the design process to choose. This choice could allow the design process to choose implementations which are optimised for a certain task. Another possibility would be that the filling chosen by the design process implements a specific behaviour which completes the generic behaviour implemented by the template. An example could be a template describing a partial agent capable of the basics needed to search, like directing the search process and reasoning about its success. The actual search algorithm could be an open slot allowing the design process to choose from different methods to search (e.g. breadth first versus depth first).

Apart from templates there are also *modules*. Modules are similar to templates but do not contain open slots. Modules play an important role in the configurational re-design because they are responsible for the specific functionality of the designed object. The templates also contain functionality but this functionality is generic in some sense. Examples of modules are databases with domain knowledge or basic algorithms (e.g. sorting). Modules are not further described in this thesis because of their similarity to templates. Another possible view is that templates are modules with open slots.

First the parts and structure of a template is discussed in section 7.1. Different types of templates are discussed in section 7.2 and different types of template objects in section 7.3. Possibilities and problems of combining templates is discussed in section 7.4 and using templates is discussed in section 7.5.

7.1 Template Structure

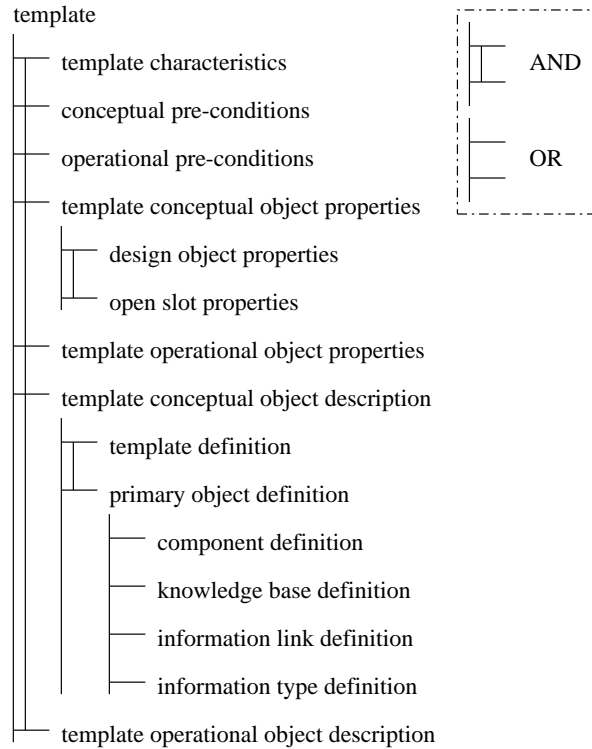


Figure ?? . The structure of a template

A template has four major parts (see Fig. ??). The first part describes the template characteristics. The second part describes the requirements imposed on and assumptions made about the context in which the template to be incorporated. This assumptions and requirements are called *pre-conditions* and can be made about the conceptual object as well as the operational object. The third part of the template describes what the objects can achieve (*conceptual object properties* and *operational object properties*). The fourth part of the template describes the structure of the template objects. This part also has two subparts. One for the description of the conceptual object (*conceptual object description*) and one for the description of the operational object (*operational object description*).

In the following paragraphs the four parts of a template are further described. The template characteristics are described in 7.1.1, the pre-conditions in section 7.1.2, the template object properties in section 7.1.3 and the template object description is described in section 7.1.4.

7.1.1 Characteristics

The characteristics part of the template states information not related to the partial design object described in the template. This contains information such as the name of the template, the creation dates, the creators of the template and version information. Other information concerning only the template and not the partial design object can be added when needed. This part is used for administrative purposes and can be used to refer to the template. Therefore the name in combination with the version should be a unique identifier.

7.1.2 Pre-conditions

Pre-conditions are used to state requirements about objects within the rest of the system (i.e. the context of the object described in the template) or to make assumptions about these objects. These pre-conditions may be used when the partial design object of the template is dependent

on functionality of others or when specific input information is needed for correct operation. The pre-conditions limit the usability of the template and should therefore only be used when the required information types are really necessary and cannot be defined within the template.

Conceptual pre-conditions can be used by both the re-design process and template retrieval. For the re-design process conceptual pre-conditions are important to check if the template fits within the current design or to determine what else is needed apart from filling the open slots to use the template.

Operational pre-conditions can be used to reason about the suitability of the operational object in terms of implementation (e.g. which compiler or VM is needed) or it can be used to determine how the object is to be activated (e.g. parameters or available methods).

7.1.3 Template Object Properties (TOP)

The property part of a template consists of two subparts. One part containing the conceptual properties of the object and one part containing the operational properties of the object. Within both parts a distinction can be made between properties describing the object and properties describing the objects that should be placed in the open slots within the object (see section 6).

Template conceptual object properties describe a template object at a conceptual level. This description can be used to find a template for a query posed by the re-design process (see section 8), to compare different templates in order to find out which template fits the query better or to formulate new queries.

Conceptual *design object* properties describe a partial design object in terms of the functionality it provides. This description also contains the functionality delegated to the open slots. The conceptual design object properties allow the retrieval of templates through matching and they contain the information needed to reason about the usability and position within a design.

Conceptual *open slot* properties describe requirements on the open slots and the required functionality of the open slots. These properties can be used to formulate queries in order to retrieve templates for the open slot or to reason about returned templates and their suitability.

Template operational object properties describe a template object at an operational level. This description can be used when specific implementations are needed (e.g. language, speed) or to assemble the parts when a conceptual design is completed.

The operational *design object* properties describe the operational design object in terms of its name, version, date, implementation language, interfaces, specialities and optimisation.

The operational *open slot* properties describe the internal interface of the open slots. This allows reasoning about the compatibility with other operational objects.

7.1.4 Template Object Description (TOD)

The template object description part of a template consists of two subparts. The first part contains a description of a *conceptual* template object. The second part contains a description of a *operational* template object. Both parts are described in the following paragraphs.

A template conceptual object description always starts with a template definition. This template definition may be used to recognize different templates within a design. If a template is a composed template other templates can be included within the template definition. When the template is a primitive template the template definition is followed by a primary object definition. These definitions are discussed below:

- *The template* definition subpart defines the name of the template. This name must be a unique identifier so that the name can be used to identify subparts of the conceptual object when it is incorporated in a design description. The template definition subpart can also be used to include other templates if the template is a composed template.
- *The component* definition subpart is used to define and describe the components contained by the template. The descriptive part of the subpart is used to define the structural hierarchy of sub components, if a component is a composed component and the knowledge bases if the component is a primitive component. Input and output information types of the component are also defined in the component definition subpart.
- *The knowledgebase* definition subpart is used to define and describe knowledgebases. Within a knowledgebase description rules can be defined.
- *The information link* definition subpart defines information links. It specifies the name, the type, the source component and level, and the destination component and level of the link. Furthermore an information type can be specified to control the type of information transported by the link. Possible link types are object-object, epistemic-object and object-assumption. If the link is an epistemic-object link a relation, an information type and a sort must also be specified in order to allow the link to create the object representation of the epistemic information.
- *The information type* definition subpart can be used to specify information types. An information type has a name and can have other information types, relations and sorts. A relations has a name and can have sorts. A sort also has a name and may have objects.

Within these primary object descriptions other objects (e.g. sub-components of a composed component) and open slots may be defined. The structure of the primary object does not have to be the same as the structure of the operational object as long as the functionality provided by the objects is the same. This allows optimisation of the operational object.

A template operational object description is operational code associated with the object described in a conceptual object description. Having an operational object description for a primary object only suffices because the need for operational object descriptions for sub-objects within the object can be fulfilled by the use of templates. Using templates to make operational object descriptions for sub-objects makes it possible re-use to objects within other templates.

The operational object description as used in the implementation described in section 9 has to be the source code because of the way the assembly line implemented for this example works. The possibility to use compiled code is left open for future work (see section 11.2).

7.2 Template Types

Within the template concept there is a distinction between two types of templates: primitive templates and composed templates. Both templates consist of pre-conditions, a Template Object Description (TOD) and Template Object Properties (TOP). The difference between a primitive template and a composed template lies in the template object description. The template object description of a primitive template contains a template definition and a description of the primary template object (see section 7.3) whereas a composed template contains only a template definition.

The composed templates allows the creator of the templates (e.g. TR self, a user) to group templates at design time without forcing TR or the re-design process to use those combinations because the parts are also available as templates. Another possible use is the grouping of templates by the TR mechanism at retrieval time. The use of multiple templates might be necessary when no single template satisfies the required properties. Allowing TR to group the templates in a composed template makes it easier for the re-design process to reason about the combination of templates because properties of this combination can be defined in the composed template by either the creator or template retrieval.

The use of composed templates makes it possible for the user of the templates to replace one part of the composed template with another template. Another option would be to allow a template to contain more than one primary object (i.e. The toplevel object contained by the template). This would, however, result in poorer maintainability and this would make replacement more intricate, because the contained objects are not distinguished as templates.

7.3 Template Object Types

When a template is a primitive template it contains one primary object at its toplevel. This primary object is the partial design object described by the template. The primary object can either be a component, a knowledge base, an information link or an information type. These types of objects are chosen from the objects made available by DESIRE to model a compositional system.

Although a template can contain only one primary object it can contain definitions of other objects within its primary object definition. These objects should however be strongly related to the generic task of the template object and not contain domain knowledge or specialized functionality if the template is of some generic kind (i.e. not a leaf template). Each template should contain properties describing the template object and an object description as described in appendix C.

In this section the different template object types are presented. Component template objects are discussed in section 7.3.1. Knowledge base template objects, information link template objects and information type objects are discussed in sections 7.3.2-7.3.4.

7.3.1 Component Templates

A component template (e.g. Generic World Search described in A.2) contains definitions for task control, sub components, knowledge bases, information types and/or information links.

Component templates may also contain open slot definitions and references to code objects. A component template should contain property definitions to describe the capabilities and abilities of components and the required properties of the open slots if there are any.

7.3.2 Knowledge base Templates

A knowledge base template (e.g. Search Location Generation in A.4) contains pre-conditions specifying certain information types or sorts and relations needed to formulate the rules. A knowledge base template may contain rule definitions, code object definitions and open slots. If the knowledge base described in the template is a reasoning knowledge base it also contains properties describing the abilities and capabilities of the knowledge base. If the knowledge base is a factual knowledge base it contains properties describing the kind of knowledge contained within these facts.

7.3.3 Information Link Templates

An information link definition contains information about the information type transported by the link and when the link is an epistemic-object link it can also contain information about the epistemic object reflection. The information type which specifies the type of information which can be transported by the link can be an open slot.

Because object-object links are modeled as simple transporting pipelines which can filter content, the need for information link templates comes only from the epistemic-object links. Information links in DESIRE however can be far more complicated. It is however not clear if open slots of the information link type are really needed in the template model.

7.3.4 Information Type Templates

An information type template (e.g. Agent Info as described in A.2) contains definitions for sub information types, sorts and relations. It can also contain open slot definitions and code object

references. It should contain properties describing the information expressed, related or defined by the information type. Requirements regarding the needed templates to complete the information type should be provide if the information type contains open slots.

7.4 Template Combinations

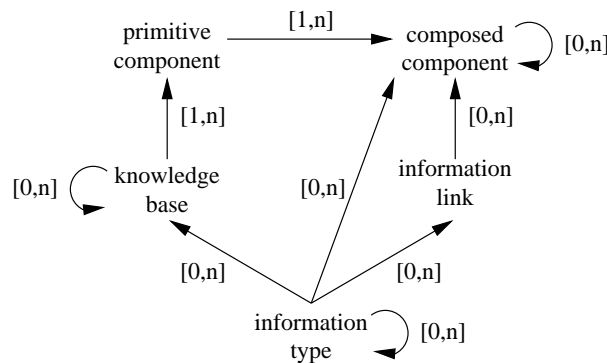
This section discusses the possibilities and problems of combining templates. Template combination is used when an open slot cannot be filled with a single template because no template exists with the right functionality. The absence of one template with the right functionality can have two reasons. The first reason could be that a template with the desired functionality is lacking (i.e. should be present but for some reason isn't). The second reason could be that a combination has to be made, because generic, and for example domain specific information, has to be combined in order to achieve the needed functionality.

Templates can only be combined if they have the same template object types. This restriction is based on the assumption that a template, or a combination of templates, is always placed in one open slot. Only combinations of components, knowledge bases and information types can be made. Information links cannot be combined because this kind of combination is not possible within DESIRE. Before a combination is placed in an open slot the combination is combined in one object of the desired type and this object is placed in the open slot. This does not affect the functionality of the combination and makes the interface between open slots and combinations simpler.

Combinations can either be made by template retrieval or by the re-design process. It might be desirable to let the retrieval process make the combinations, because TR has the knowledge about the abilities of the combinations at his disposal. This knowledge is stored within the semantic property networks (see section 6.2). It might be necessary to store abilities of important combinations in composed templates in order to simplify the retrieval process. Composed templates can be easily retrieved by matching the properties but for reasoning about combinations of templates traversal of the semantic property networks is needed. When a combination is found it therefore might be useful to store this combination within a new composed template in order to allow easy retrieval of this combination when it is needed again.

7.5 Template Use

The usability of a template strongly depends on the quality of the match between the query and the template. Each template has its capabilities and knowledge described by means of properties. These properties are matched with the desired properties in the query. The query cannot be expected to use the exact terms which are used to define the template. The Template Retrieval mechanism must therefore have the ability to refine and reformulate the template properties in order to find the best match. Another possibility would be to formulate all the refinements and alternatives within the template. The matching is done with the problem description part of the template. In order to find the best match some heuristics should be used.



8 Matching

The primary task of template retrieval is finding templates for the re-design process in order to support this process. The needs of the re-design process are expressed as queries. The act of finding the right templates is performed by TEMPLATE SET MANIPULATION by matching requested properties with what the templates offer. When the properties requested in the query (see section ??) are united in one template this matching is straight forward, but when this is not the case a partial match has to be made and templates may have to be combined in order to fulfil the request. This partial matching makes use of heuristics to grade the quality of the match and to give template retrieval the opportunity to choose between different matching candidates. Template retrieval can match in two different ways. It can match what is asked directly with what is offered but it can also try to interpret the query in case there are templates which do not directly offer what is requested but partially or in other terms.

The concept of queries is explained in section 8.1. The concept of heuristics used to grade and choose are discussed in section 8.2. After that matching without interpretation of the query is discussed in section 8.3 and matching with interpretation of the query is discussed in section 8.4.

8.1 Queries

Queries are used to retrieve templates from a template database. A query consists of a Qualified Property Set (QPS) and a set of retrieval objectives. A QPS is used to specify requested properties and their level of desirability. Retrieval objectives may be used to specify additional requirements on the retrieval process (e.g. number of templates, level of interpretation, available time, number of alternatives to be generated).

Qualified Property Sets (QPSs) are used as the main form of communication between the re-design process and template retrieval. The re-design process uses (or has to use?) QPSs to communicate its *query* with template retrieval. Template retrieval, in turn, uses the QPSs to communicate the *results* of the retrieval process back to the re-design process. A *qualified property set* consists of a list of *property qualifications*. Each of these property qualifications consists of a qualification which can either be *hard* or *soft* and the property that is being qualified. A QPS can also contain a hierarchy defining how the properties are related to each other. This hierarchy is used by the re-design process to help TR on its way or by TR to explain how it has refined the properties.

- QPS(<QPS_name>, [<property qualification name>, ...]);
- PQ(<property qualification name>, <qualification>, <property>);
- is specialisation of(<QPS_name>, [<property qualification name>, <property qualification name>, ...]);

When the re-design process sends a QPS to template retrieval this QPS forms a query. The QPS then contains a set of properties that are requested with a specified level of desirability. These properties can be either be desired properties of the object described in the template or of the structure of the templates in the template set. The re-design process can, for example, desire that its returned functionality is distributed over two components, because it has two component open slots which have to be filled.

The QPS sent back to the re-design process after template retrieval contains the result of the query. This set is different from the query in two ways. First the properties have been modified to specify which templates satisfy the requested properties. And secondly the set may have been expended with refinements if no templates could be found that satisfy one or more of the properties

from the query. These refinements are specified using the *is_specialisation_of* relation to inform the re-design process of the steps taken to retrieve the templates returned.

Examples of qualified property sets:

When the re-design process is looking for templates for an agent that is able of search and maybe able to negotiate it may construct the following QPS:

```
QPS('query number 123', ['qp 123-1', 'qp 123-2', 'qp 123-3']);
QP('qp 123-1', 'hard', 'able_to(perform(agent_functions))');
QP('qp 123-2', 'hard', 'able_to(search)');
QP('qp 123-3', 'soft', 'able_to(negotiate)');
```

A QPS can also be used to request a specific template by specifying its name:

```
QPS('query number 124', ['rqp 124-1']);
QP('rqp 124-1', 'hard', 'is_template('harry23')');
```

8.2 Matching Heuristics

When properties partially match with templates in a template database heuristics have to be employed to retrieve matching templates and multiple solutions are found heuristics also have to be used to choose a best match. Semantic property networks can be used to compare properties structurally, functionally or between domains. Examples of heuristics for partial matching are:

- The use of more generic properties: 'If property A is requested and a template has property B and property C can be refined in property A or B then the template might be a match'
- The use of more specific properties: 'If property A is requested and a template has property B and C and property A can be refined in property B and C then the template might be a match'

possible heuristics for choosing between alternative solutions (ranking) are:

- number of matching properties (weighted with use of qualification)
- number of refinements made to retrieve (less refinements is better)

8.3 Matching without interpretation

Matching without interpretation is the most basic form of retrieval offered by template retrieval. A Qualified Property Set (QPS) is analysed and a template with the requested properties is searched. In this search all templates which have the properties that are qualified as being *hard* by the QPS are retrieved. Depending on the number of alternatives which is to be returned the properties that are qualified as being *soft* are used to select one or more of the templates as best matches.

Matching without interpretation may return multiple templates as an answer if the query specifies that combinations are allowed and no single template can be found to fulfil the requested properties. Because no interpretation is used to group the properties these combinations might not be very useful in advanced stage of the design. At the start of the design these alternatives might be interesting because they can provide different starting points.

8.4 Matching with variable forms of interpretation

Matching with variable forms of interpretation allows template retrieval to put more creativity into its matches. At least four different forms of interpretation can be identified:

Property refinement The first possible way to interpret a query is to refine (see section 6.1) the properties in the QPS. This could be done in either an data-driven fashion or in a goal-driven fashion. The choice between either fashion can depend on the number of refinements per property and on the number of templates in the template database. Experience in the refinement of properties can be used to choose between different possible refinements.

SPN traversal The second possible way to interpret a query is to use similarity between different but similar tasks (e.g. same type of task but different task domain). This interpretation is based on the assumption that if a solution works in one domain it might work in other domains as well.

Clair voyant retrieval The thirds possible way to interpret a query is to consider often used combinations when returning an answer. If more templates of often used combination are in an answer it is likely that the rest of the combination is needed as well. Another possible way to predict the use of combinations is by considering earlier requested templates. An example of a predictable combination is an Agent Specific Task (AST) component and a World Interaction Management (WIM) component if a search agent component is requested earlier. Because the search agent component has an AST open slot and a WIM open slot which both depend on the searched environment the WIM component can be predicted if a specific AST component is requested. Some caution is needed in using the predictions because they may influence the creativity of the re-design process [Stacey, 1995].

Property grouping The fourth possibility to interpret a query is the use of knowledge about the design to group the properties together so that templates for those groups can be linked to eachother using the same knowledge. An example for this grouping is the use of the knowledge that the task domain is search for computer parts to group the requested properties (e.g. able to search and completes sort world info) within the search for computer parts plane of the SPN so that right templates are retrieved (***) .

9 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).

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 B. The prototype implementation is presented in Appendix ?? . An output trace the prototype is presented in Appendix ?? . An agent toplevel used in this implementation is discussed in section ??

9.1 The Agent Top-level

The design of the top-level of the agent template (see A.1) consists of the agent component itself, an external world interface component and a communication interface component. This design of the agent top-level template, used as an example is meant to allow easy portability between multi

agent systems and ease of communications between an agent and other agents and between an agent and an external world. The three components on the top-level of the agent template form the basic structure of an agent in a distributed agent systems. These elements can reside on a host where they are connected to the hosts external world interface and communication interface. Together these elements and the host make a multi agent system.

The language conversion knowledge within the interface components of the agent is contained in knowledge bases. These knowledge bases are templates which are placed in open slots. This means that these knowledge bases can easily be replaced with other knowledge bases if another language is to be spoken. This is useful because an agent could have to speak to one host through a shared file and to another directly through a specific network protocol. This leaves the design of the host open as long as the host can reconfigure the agent and implement the interface components of the agent to translate the agent communication and world interaction into communications and world interactions understood by the host.

The following assumptions are made on Multi-Agent System and Agents for Multi-Agent Systems:

- Each agent is contained within a MAS element
- Each active MAS element is contained within a host
- Each host containing MAS elements is a multi agent system
- The hosts can communicate with eachother
- Interconnected hosts form a MAS
- A MAS element can travel from one host to the other
- A MAS element has a unique identifier
- A MAS element can have only one host as location
- A MAS element knows the names of the agents it wants to communicate with
- A host knows the locations named agents
- A MAS element can communicate with another MAS element by communicating with the host interface as where the interface the interface of the other agent
- A MAS element is aware of the possible actions and observations made available by the host
- Each host has its own external world

9.1.1 Agent incorporation

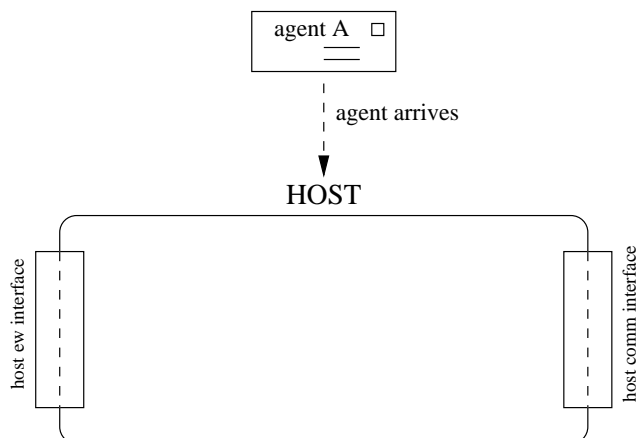


Figure ?? . Arrival of agent at a host

When an agent arrives at a host it first registers with the host. The host then analyses the design rationale of the agent to check if the interface components (i.e. the external world interface and the communication interface) are compatible with the interfaces of the host. If the agents interface components are compatible the agent can be inserted into the MAS directly but the interface components have to be reconfigured if they are not compatible with the host interface:

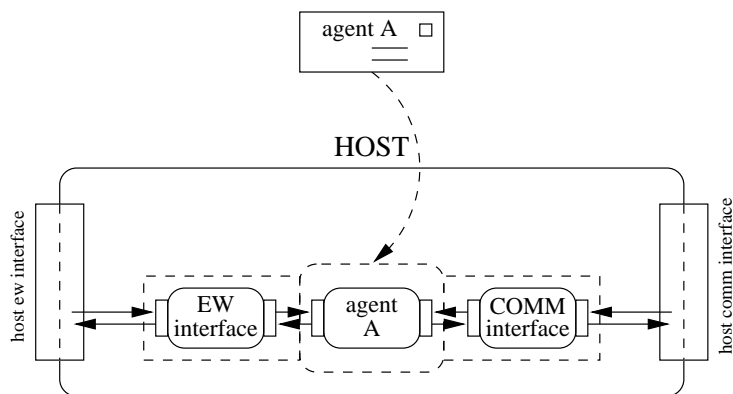


Figure ???. Insertion of agent without adaption

If the interface components of an arriving agent are compatible with the interface of the host then the host can simply connect the interface of the agent with the interface of the host and activate the agent.

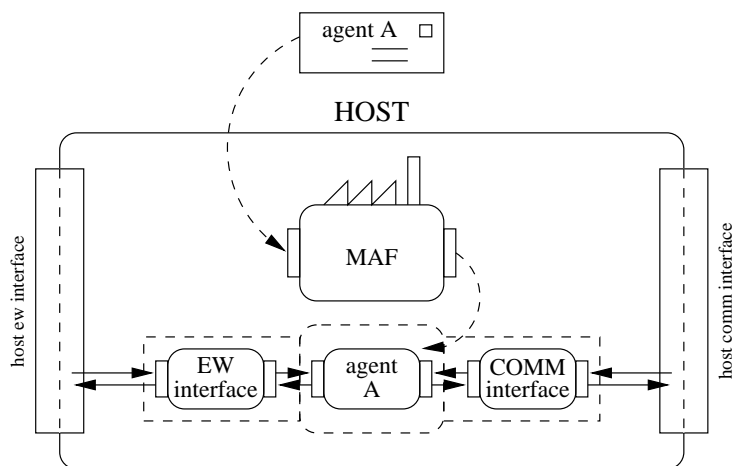


Figure ???. Insertion of agent with adaption

If the interface components of an arriving agent are not compatible with the interface of the host then the agent has to be redesigned to become compatible. The host then sends the agent to its local MAF together with design objectives and a Qualified Property Set which defines the needed modification of the agent. The MAF will reconfigure the interface components thus making the agent able to speak and understand the communication language of the host. For the MAF to complete this task it might be necessary to consult with the host from which the agent originates in order to get additional templates and knowledge about the particular design of the agent. When the agent is adapted its ready to be connected to the interface of the host and activated.

9.1.2 Agent to Agent communication

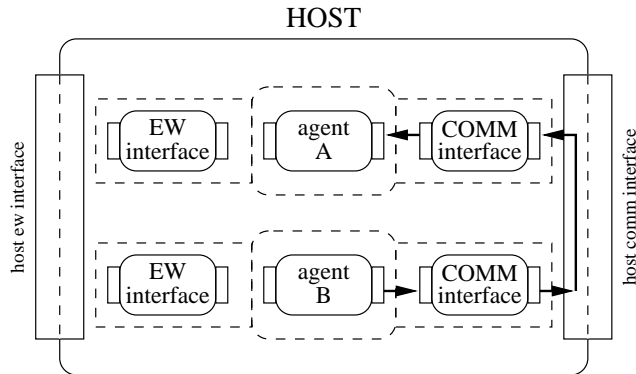


Figure ?? Intra host communication

When an agent wants to communicate with another agent on the host the agent sends its message to its personal communication interface. This communication interface converts the message into a generic agent communication language (e.g. KQML) which is then either converted into a host specific communication language by the part of communication interface implemented by the host and send to the host communication interface or communicated directly to the host communication interface. The host communication interface then passes the message to the designated agent whose communication interface component in turn translates the message into the agent specific language.

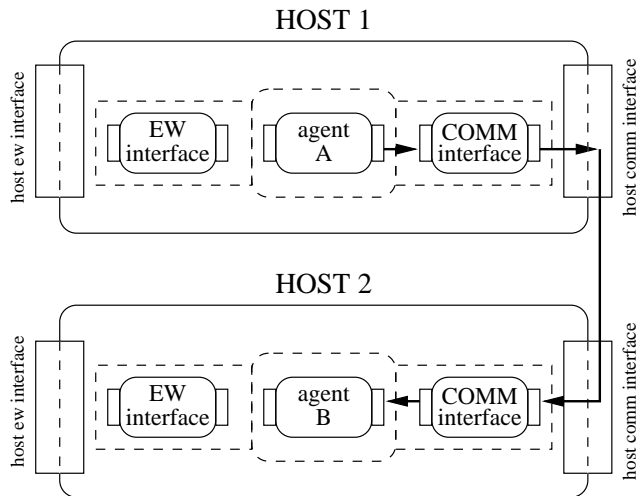


Figure ?? Inter host communication

When an agent wants to communicate with another agent situated on another host the agent sends its message to the other agent as if the agent is on the same host. The host is responsible for locating the other host (where the designated agent is situated) and communicating the message to that host, presuming that the two hosts are able to communicate with each other. When the message is sent to the second host this host translates the message into the host communication language and then sends it to the communication interface of the designated agent. This interface then translates the message into the agent specific communication language and sends the message to the receiving agent.

9.1.3 Agent to External World interaction

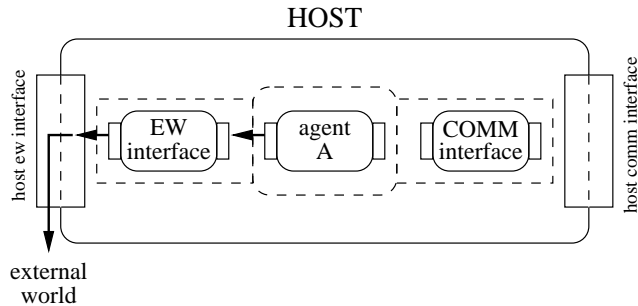


Figure ?? . Interaction between Agent and EW

If an agent wants to observe or perform actions in an external world which is accessible through the interfaces of the host then the agent can send the actions or observations to the external world interface. The external world interface of the agent then translates the actions or observations into the communication language of the host and forwards the translated information to the external world interface of the host. The host then performs the actions or makes the observations and sends the feedback or results back to the external world interface of the agent. This interface then translates the received information back into for the agent understandable terms and forwards the translate information to the agent component.

10 Template use

This sections discusses the usage of templates. First the role of the designing done by Template Retrieval (TR) and the possibilities of using TR in other domains is addressed in the following paragraphs. After that an example based on the example design described in [Mobach, 2000] is given in section 10.1. A simple trace describing how TR could retrieve the templates needed in the example is given in ??.

Designing within TR The retrieval mechanism in TR is allowed to combine templates in order to fulfil the requested properties and can anticipate future queries and return unrequested templates to help te re-design process. It is, however, not desirable that the retrieval process takes over the design tasks because the configurational knowledge lays with the re-design process. The combinations should therefore be kept as simple as possible and the re-design process should not blindly accept the unrequested templates as mandatory and should try to stick to its creativity.

Other usage of TR Although template retrieval is designed with the multi-agent factory in mind the domain is left open. The properties and the property structures can be adapted to fit the needs of other design tasks (e.g. architecture) as long as a compositional framework can be used. The use of operational object description for automated assembly suggest the usage of templates for the design of computer programs but partial blueprints could be used in order to generate an elaborated blueprint for manufacturing.

10.1 Example of template use

To explain the retrieval process an example retrieval based on a query received from the re-design process discussed in [Mobach, 2000] is discussed in this section.

An example query send to `TEMPLATE RETRIEVAL` is the following Qualified Property Set (QPS):

```

qps('qps1', {'pq1'})
pq('pq1', hard, 'p1')
p('p1', able_to(perform(weak_agent_functions)))

```

The qualified property set *qps1* is defined by the first statement and the property qualification PQ1 is associated with *qps1*. In this case only one property qualification is associated with the QPS, but multiple qualification can be associated with one QPS as shown in the result. The second statement defines the property qualification called *pq1*. The qualified property *pq1* qualifies the property *p1* as *hard* (i.e. has to be a property of the returned result). The third and last statement defines the property relation *p1* which relates the identifier *p1* with the property *able_to(perform(weak_agent_functions))*.

Using this query TEMPLATE RETRIEVAL will try to retrieve a template that fulfills the requested properties defined by the query. Because no explicit retrieval objectives are issued RETRIEVAL OBJECTIVE first tries to find a template matching the property *p1*. If no template could be found matching the property *able_to(perform(weak_agent_functions))* TEMPLATE RETRIEVAL may try to refine the property in order to find a template that matches the refined properties. In this example *p1* is refined in *able_to(interact_with(material_world))* identified as *p2* and *able_to(interact_with(agents))* identified as *p3*. For these properties template *t1* can be found and is send back with the assessment that *qps2* is based on *qps1* and that *t1* has property *p1* because it has *p2* and *p3* and *p1* can be refined in *p2* and *p3*.

The resulting QPS TEMPLATE RETRIEVAL sends back is as follows:

```

qps('qps2', {'pq1', 'pq2', 'pq3'})
pq('pq1', hard, 'p1')
pq('pq2', hard, 'p2')
pq('pq3', hard, 'p3')
p('p1', able_to(perform(weak_agent_functions)))
p('p2', able_to(interact_with(material_world)))
p('p3', able_to(interact_with(agents)))

```

The resulting assessments about the qualified property set returned in relation to the initial qualified property set is as follows:

```

based_on('qps2', 'qps1')
has_refinement('p1', and('p2', 'p3'))

```

The resulting assessments about the template set returned in relation to the returned qualified requirement set is as follows:

```

contains_template('ts1', 't1')
fulfills('t1', 'p1', 'qps2')
fulfills('t1', 'p2', 'qps2')
fulfills('t1', 'p3', 'qps2')

```

11 Conclusions and Future work

With the demand for agents with various abilities grows the need for a configuration facility capable of dynamically generating and modifying agents. The Multi-Agent Factory is such a configuration facility. Part of the Multi-Agent Factory is the AGENT DESIGN process. A configuration based model for re-design is presented in [Mobach, 2000]. Template based support for this re-design process is presented in this thesis.

In section 11.1 conclusions are given. Possible future work is discussed in section 11.2.

11.1 Conclusions

'What is a possible model for a configuration facility with the task to dynamically generate and modify agents for specific tasks in specific domains.'

In this thesis in combination with [Mobach, 2000] a model is presented for configuration based re-design of agents. The goal is to achieve a model for a configuration facility with the task to dynamically generate and modify agents with specific tasks in specific domains. To support this configuration facility templates are used. Templates should support the description of various tasks and various types of domain knowledge. Templates should also support the dynamically generation and modification of agents. A model for these templates is presented in this thesis. A model for configuration based agent re-design is presented in [Mobach, 2000].

The template model presented accomodates:

- The use of conceptual object descriptions using DESIRE to models processes and knowledge.
- The disctinction between conceptual descriptions of an object and operational descriptions of an object.
- The automated generation of operational object descriptions (e.g. JAVA code) by making a mapping between conceptual description combinations and combinations of associated operational descriptions.

Advantages of the use of tempaltes are:

- Reuse (re-design) knowledge.
- Easy maintainability by use of partial descriptions which can be individually modified.
- Learning of new (design) knowledge
- Vague and incomplete queries by the use of a generic design model for retrieving templates.

The genericness if the approach taken in this thesis allows the use of templates within other domains as long as the object that are being designed can be modeled using a compositional structure.

Only a selection of the available DESIRE objects are used but no difficulties are expected in adding other objects. The assumption is made that for TEMPLATE RETRIEVAL to succeed the requested properties must be used in a template description or can be refined into properties used in the available tempaltes. The presented model assumes that both the re-design process and TEMPLATE RETRIEVAL can operate autonomously but both processes may fail if insufficient knowledge is available. In those cases human support may be needed in order to continue the re-design process.

This support is achieved by using the modeling paradigm DESIRE and a conceptual description of modeled object using properties. Support for dynamic generation and modification is given by associating a operational object description to the conceptual object description. This operational object description allows a MAF to compile the templates used to design an agent into an operational description of that design.

11.2 Future Work

This thesis presents a first examination of the model for templates described within this thesis. Not all intricacies have been examined and new aspects have been found. The following points are still left open for future research:

- Communication between template retrievals to exchange packages of templates (security, compatibility).
- Addition of the other objects to the template model (e.g. sort, relation, task control).

- Extension of TR with the possibility to add new templates to the database, based on: new designs found by the re-design process, new combinations found by TR or often asked/made combinations.
- Use of MAF/TR in other design environments and or other domains (instead of agents).
- Further specify possible representations for templates and storage within the template database. ■
- Further analyse possible retrieval mechanisms.
- Addition of qualification of open slot properties to allow enhanced configuration and easier formulation of queries.
- Further analysis of the property language used. How is dynamic behaviour described and used in reasoning (e.g. number of items generated ($n=1$ or $n_i=1$), able to reason in parallel).
- Research to indentify possible forms of interpretation of queriesl.
- What may differ within versions of templates without making it a new template.
- To what extend can the operational object deviate from the conceptual description without becomming inconsistent with the conceptual description.
- Using QPS to state undesired properties. This allows TR to rule out templates.
- Extending templates with case-based like knowledge about usable contexts and matching on this information.
- Using UML to model conceptual object descriptions.

Acknowledgments

I would like to thank Niek Wijngaards and Frances Brazier for their time, support, patience and for their ideas without which this never would have existed. I thank Maarten van Steen for his time and for reading this thesis. Also I would like to thank David Mobach for the support throughout the last months and for being a terrible partner which made the co-operation possible and fun.

References

- [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.
- [Benjamins et al., 1996] Benjamins, V. R., Fensel, D., and Straatman, R. (1996). Assumptions of problem-solving methods and their role in knowledge engineering. In Wahlster, W., editor, *Proceedings European Conference on AI (ECAI '96)*. Wiley and Sons.
- [Beys et al., 1996] Beys, P., Benjamins, V. R., and van Heijst, G. (1996). Remedying the reusability - usability trade-off for problem-solving methods. In Gaines, B. R. and Musen, M., editors, *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW96)*.
- [Blair et al., 2000] Blair, G. S., Blair, L., Issarny, V., Tuma, P., and Zarras, A. (2000). The role of software architecture in constraining adaption in component-based middleware platforms. *Lecture Notes in Computer Science*, 1795:164–184.

- [Brazier et al., 1997a] Brazier, F., Dunin-Keplicz, B., Jennings, N., and Treur, J. (1997a). 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., 2000a] Brazier, F. M. T., Cornelissen, F., Jonker, C. M., and Treur, J. (2000a). Compositional specification and reuse of a generic co-operative agent model. *International Journal of Cooperative Information Systems*, 9:171–207.
- [Brazier et al., 2000b] Brazier, F. M. T., Jonker, C. M., and Treur, J. (2000b). Compositional design and reuse of a generic agent model. *Applied Artificial Intelligence*, 14(5):491–538.
- [Brazier et al., 2000c] Brazier, F. M. T., Jonker, C. M., and Treur, J. (2000c). *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., 1998b] Brazier, F. M. T., Jonker, C. M., Treur, J., and Wijngaards, N. J. E. (1998b). On the role of abilities of agents in redesign. In Gaines, M. and Musen, editors, *Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-based Systems workshop*, page 16. Calgary: SRDG Publications.
- [Brazier et al., 2000d] Brazier, F. M. T., Jonker, C. M., Treur, J., and Wijngaards, N. J. E. (2000d). Deliberate evolution in multi-agent systems. In Gero, J. S., editor, *Artificial Intelligence in Design*, pages 633–650. Kluwer Academic Publishers.
- [Brazier et al., 1997b] Brazier, F. M. T., M., D.-K. B., Jennings, N. R., and Treur, J. (1997b). Desire: modelling multi-agent systems in a compositional formal framework. *International Journal of Cooperative Information Systems*, 6(1):67–94.
- [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.
- [Felfering et al., 2000] Felfering, A., Friedrich, G., and Jannach, D. (2000). Conceptual design for configuration and reconfiguration of customizable products. In Rosenman, M. and Simoff, S., editors, *Advances in Conceptual Modelling in Design*, Artificial Intelligence in Design, pages 5–21. workshop 1 notes.
- [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.
- [Kolodner, 1993] Kolodner, J. L. (1993). *Case-Based Reasoning*. Morgan Kaufman.
- [Mobach, 2000] Mobach, D. G. A. (2000). An agent (re)configuration model using templates for agent design. Master’s thesis, Vrije Universiteit Amsterdam.
- [Perez and Benjamins, 1999] Perez, A. G. and Benjamins, V. R. (1999). Overview of knowledge sharing and reuse components: Ontologies and problem-solving methods. In *Proceedings of the IJCAI-99 workshop on Ontologies and Problem-Solving Methods (KRR5)*, pages 1–15.

- [Schank and Abelson, 1977] Schank, R. C. and Abelson, R. P. (1977). *Scripts, Plans, Goals and Understanding*. Lawrence Erlbaum.
- [Stacey, 1995] Stacey, M. K. (1995). Distorting design: Unevenness as a cognitive dimension of design tools. *Adjunct Proceedings of HCI'95*, pages 90–95.
- [Sycara et al., 1999] Sycara, K., Klusch, M., Widoff, S., and Lu, J. (1999). Dynamic service matchmaking among agents in open information environments. *ACM Special Issue on Semantic Interoperability in Global Information Systems*, 28(1):47–53.
- [van Harmelen and ten Teije, 1998] van Harmelen, F. and ten Teije, A. (1998). Characterising problem solving methods by gradual requirements. In *Proceedings of the Eleventh Workshop on Knowledge Acquisition for Knowledge-Based Systems (KAW'98)*, Banff, Alberta.
- [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.

/* index remarked */

A Generic Search Example

/* Generic Search Example met computer parts, etc... */

A.1 Agent template

A.1.1 Information Types

```
generic_obs_to_be_perf
├── infotype(obs_to_be_perf)
generic_obs_results
├── infotype(obs_results)
generic_incoming_comm_info
├── infotype(incoming_comm_info)
generic_outgoing_comm_info
├── infotype(outgoing_comm_info)
```

A.1.2 Agent

pre-conditions
structure

```
//define template
is_template(Agent);
is_toplevel(toplevel);

// define template contents
is_component(agent);
  is_composed(agent);
```

```

    has_input_information_type(agent, generic_incoming_comm_info, 1);
    has_input_information_type(agent, generic_obs_results, 1);
    has_output_information_type(agent, generic_outgoing_comm_info, 1);
    has_output_information_type(agent, generic_obs_to_be_perf, 1);
    is_open_slot(agent);
is_component(user);
    is_primitive(user);
    has_input_information_type(user, generic_outgoing_comm_info, 1);
    has_output_information_type(user, generic_incoming_comm_info, 1);
    has_code_object(user, 'agent_user.class', java);
is_component(external_world);
    is_primitive(external_world);
    has_input_information_type(external_world, generic_obs_to_be_perf, 1);
    has_output_information_type(external_world, generic_obs_results, 1);
    has_code_object(external_world, 'agent_external_world.class', java);
is_task_control(toplevel_tc);
    has_code_object(toplevel_tc, 'agent_toplevel_tc.class', java);

has_subcomponent(toplevel, agent);
has_subcomponent(toplevel, user);
has_subcomponent(toplevel, external_world);

// define information types
is_information_type(generic_obs_to_be_perf);
    is_open_slot(generic_obs_to_be_perf);
is_information_type(generic_obs_results);
    is_open_slot(generic_obs_results);
is_information_type(generic_incoming_comm_info);
    is_open_slot(generic_incoming_comm_info);
is_information_type(generic_outgoing_comm_info);
    is_open_slot(generic_outgoing_comm_info);

has_information_type(toplevel, generic_obs_to_be_perf);
has_information_type(toplevel, generic_obs_results);
has_information_type(toplevel, generic_incoming_comm_info);
has_information_type(toplevel, generic_outgoing_comm_info);

// define information links
is_information_link(selected_obs);
    is_link_type(selected_obs, oo);
    has_information_type(selected_obs, generic_obs_to_be_perf);
    has_source_component(generic_obs_to_be_perf, agent);
    has_source_level(generic_obs_to_be_perf, 1);
    has_destination_component(generic_obs_to_be_perf, external_world);
    has_destination_level(generic_obs_to_be_perf, 1);

is_information_link(obs_results_for);
    is_link_type(obs_results_for, oo);
    has_information_type(obs_results_for, generic_outgoing_comm_info);
    has_source_component(obs_results_for, agent);
    has_source_level(obs_results_for, 1);
    has_destination_component(obs_results_for, user);
    has_destination_level(obs_results_for, 1);

```

```
is_information_link(query_res_to_user);
  is_link_type(query_res_to_user, oo);
  has_information_type(query_res_to_user, generic_outgoing_comm_info);
  has_source_component(query_res_to_user, agent);
  has_source_level(query_res_to_user, 1);
  has_destination_component(query_res_to_user, user);
  has_destination_level(query_res_to_user, 1);
```

```
is_information_link(queries_to_agent);
  is_link_type(queries_to_agent, oo);
  has_information_type(queries_to_agent, generic_incoming_comm_info);
  has_source_component(agent, generic_outgoing_comm_info, user);
  has_source_level(agent, generic_outgoing_comm_info, 1);
  has_destination_component(agent, generic_outgoing_comm_info, agent);
  has_destination_level(agent, generic_outgoing_comm_info, 1);
```

```
has_information_link(toplevel, selected_obs);
has_information_link(toplevel, obs_results_for);
has_information_link(toplevel, query_res_to_user);
has_information_link(toplevel, queries_to_agent);
```

properties

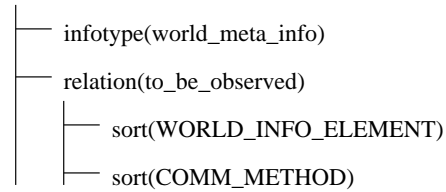
```
has_property(Agent, is(generic));
has_property(Agent, is(toplevel));
has_property(Agent, able_to(interact_with(external_world)));
has_property(Agent, able_to(interact_with(user)));
```

```
// define open slot requirements
has_property(agent, able_to(perform(agent_functions)));
/* open slot infotypes (***) */ generic_obs_to_be_perf
generic_obs_results
generic_incoming_comm_info
generic_outgoing_comm_info
```

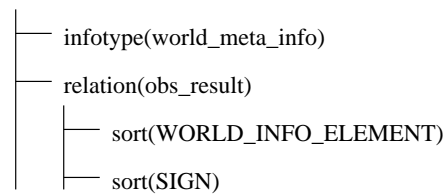
A.2 Generic World Search Template

A.2.1 Information Types

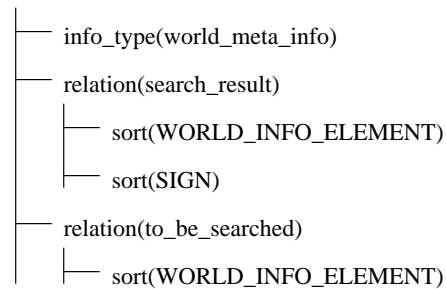
obs_to_be_perf



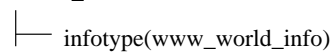
obs_results



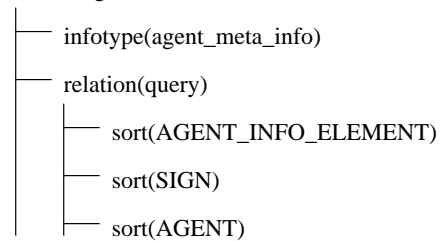
search_info



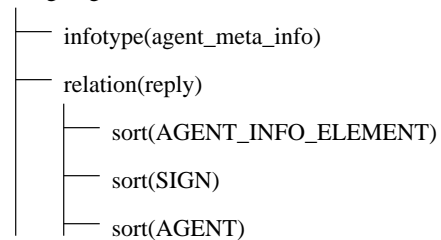
world_info



incoming_comm_info



outgoing_comm_info



```

communicated_info
├── infotype(agent_meta_info)
├── relation(communicated_in)
│   └── sort(AGENT_INFO_ELEMENT)
├── relation(communicated_out)
│   └── sort(AGENT_INFO_ELEMENT)
agent_info
├── infotype(www_agent_info)

```

A.2.2 Generic World Search Agent

pre-conditions

```
/* pre condities */ structure
```

```

//define template
is_template(generic word search agent);
is_component(gws_agent);
    is_composed(gws_agent);
    has_input_information_type(gws_agent, incoming_comm_info, 1);
    has_input_information_type(gws_agent, obs_results, 1);
    has_output_information_type(gws_agent, outgoing_comm_info, 1);
    has_output_information_type(gws_agent, obs_to_be_perf, 1);

// define template contents
is_component(aim);
    is_primitive(aim);
    has_code_object(aim, 'gws_aim.class', java);
    has_input_information_type(aim, communicated_info, 1);
    has_input_information_type(aim, incoming_comm_info, 1);
    has_output_information_type(aim, communicated_info, 1);
    has_output_information_type(aim, outgoing_comm_info, 1);
is_component(wim);
    is_composed(wim);
    is_open_slot(wim);
    has_input_information_type(wim, search_info, 1);
    has_input_information_type(wim, obs_results, 1);
    has_output_information_type(wim, search_info, 1);
    has_output_information_type(wim, obs_to_be_perf, 1);
is_component(ast);
    is_composed(ast);
    is_open_slot(ast);
    has_input_information_type(ast, world_info, 1);
    has_input_information_type(ast, agent_info, 1);
    has_output_information_type(ast, world_info, 1);
    has_output_information_type(ast, agent_info, 1);
has_subcomponent(gws_agent, aim);
has_subcomponent(gws_agent, wim);
has_subcomponent(gws_agent, ast);
is_task_control(gws_agent.tc);

```

```

    has_code_object(gws_agent_tc, 'gws_agent_tc.class', java);
has_task_control(gws_agent, gws_agent_tc);

// define information links: communicated info to aim
is_information_link(communicated_info_to_aim);
    is_link_type(communicated_info_to_aim, oo);
    has_information_type(communicated_info_to_aim, agent_info);
    has_source_component(communicated_info_to_aim, gws_agent);
    has_source_level(communicated_info_to_aim, 1);
    has_destination_component(communicated_info_to_aim, aim);
    has_destination_level(communicated_info_to_aim, 1);

is_information_link(observation_results);
    is_link_type(observation_results, oo);
    has_information_type(observation_results, world_info);
    has_source_component(observation_results, gws_agent);
    has_source_level(observation_results, 1);
    has_destination_component(observation_results, wim);
    has_destination_level(observation_results, 1);

is_information_link(results_to_aim);
    is_link_type(results_to_aim, eo);
    has_eo_relation(results_to_aim, communicated_out);
    has_eo_infotype(results_to_aim, communicated_info);
    has_eo_sort(results_to_aim, AGENT_INFO_ELEMENT);
    has_information_type(results_to_aim, agent_info);
    has_source_component(results_to_aim, ast);
    has_source_level(results_to_aim, 2);
    has_destination_component(results_to_aim, aim);
    has_destination_level(results_to_aim, 1);

is_information_link(observation_request_to_wim);
    is_link_type(observation_request_to_wim, eo);
    has_eo_relation(observation_request_to_wim, to_be_searched);
    has_eo_infotype(observation_request_to_wim, search_info);
    has_eo_sort(observation_request_to_wim, WORLD_INFO_ELEMENT);
    has_information_type(observation_request_to_wim, world_info);
    has_source_component(observation_request_to_wim, ast);
    has_source_level(observation_request_to_wim, 2);
    has_destination_component(observation_request_to_wim, wim);
    has_destination_level(observation_request_to_wim, 1);

is_information_link(selected_observations);
    is_link_type(selected_observations, oo);
    has_information_type(selected_observations, world_info);
    has_source_component(selected_observations, wim);
    has_source_level(selected_observations, 1);
    has_destination_component(selected_observations, gws_agent);
    has_destination_level(selected_observations, 1);

is_information_link(communication_info_to_ast);
    is_link_type(communication_info_to_ast, oa);
    has_information_type(communication_info_to_ast, agent_info);
    has_source_component(communication_info_to_ast, aim);

```

```

has_source_level(communication_info_to_ast, 1);
has_destination_component(communication_info_to_ast, ast);
has_destination_level(communication_info_to_ast, 2);

is_information_link(info_to_be_communicated);
is_link_type(info_to_be_communicated, oo);
has_information_type(info_to_be_communicated, agent_info);
has_source_component(info_to_be_communicated, aim);
has_source_level(info_to_be_communicated, 1);
has_destination_component(info_to_be_communicated, gws_agent);
has_destination_level(info_to_be_communicated, 1);

is_information_link(observations_to_ast);
is_link_type(observations_to_ast, oa);
has_information_type(observations_to_ast, world_info);
has_source_component(observations_to_ast, wim);
has_source_level(observations_to_ast, 1);
has_destination_component(observations_to_ast, ast);
has_destination_level(observations_to_ast, 2);

// define information type
is_information_type(world_info);
is_open_slot(world_info);

is_information_type(world_meta_info);
is_sort(WORLD_INFO_ELEMENT);
is_meta(WORLD_INFO_ELEMENT);
has_information_type(WORLD_INFO_ELEMENT, world_info);
has_sort(world_meta_info, WORLD_INFO_ELEMENT);

is_information_type(obs_to_be_perf);
has_information_type(obs_to_be_perf, world_meta_info); is_relation(to_be_observed);
has_sort(to_be_observed, WORLD_INFO_ELEMENT);
has_sort(to_be_observed, COMM_METHOD);
has_relation(obs_to_be_perf, to_be_observed);

is_information_type(obs_results);
has_information_type(obs_results, world_meta_info);
is_relation(obs_result);
has_sort(obs_result, WORLD_INFO_ELEMENT);
has_sort(obs_result, SIGN);
has_relation(obs_results, obs_result);

is_information_type(search_info);
has_information_type(search_info, world_meta_info);
is_relation(to_be_searched);
has_sort(to_be_searched, WORLD_INFO_ELEMENT);
has_relation(search_info, to_be_searched);
is_relation(search_result);
has_sort(search_result, WORLD_INFO_ELEMENT);
has_sort(search_result, SIGN);
has_relation(search_info, search_result);

is_information_type(agent_info);

```

```

is_open_slot(agent_info);

is_information_type(agent_meta_info);
is_sort(AGENT_INFO_ELEMENT);
is_meta(AGENT_INFO_ELEMENT);
has_information_type(AGENT_INFO_ELEMENT, agent_info);
has_sort(agent_meta_info, AGENT_INFO_ELEMENT);

is_information_type(incoming_comm_info);
has_information_type(incoming_comm_info, agent_meta_info);
is_relation(query);
has_sort(query, AGENT_INFO_ELEMENT);
has_sort(query, SIGN);
has_sort(query, AGENT);
has_relation(incoming_comm_info, query);

is_information_type(outgoing_comm_info);
has_information_type(outgoing_comm_info, agent_meta_info);
is_relation(reply);
has_sort(reply, AGENT_INFO_ELEMENT);
has_sort(reply, SIGN);
has_sort(reply, AGENT);
has_relation(outgoing_comm_info, reply);

is_information_type(communicated_info);
has_information_type(communicated_info, agent_meta_info);
is_relation(communicated_out);
has_sort(communicated_out, AGENT_INFO_ELEMENT);
has_relation(communicated_info, communicated_out);
is_relation(communicated_in);
has_sort(communicated_in, AGENT_INFO_ELEMENT);
has_relation(communicated_info, communicated_in);
properties
has_property(gws_agent, capable_of(performing(agent_functions)));
has_property(gws_agent, able_to(behave_pro-active));
has_property(gws_agent, able_to(search));
has_property(gws_agent, able_to(search_pro-active));

has_property(gws_agent, capable_of(interacting_with_agents));
has_property(gws_agent, able_to(perform_search));
has_property(gws_agent, able_to(interact_with_world));
has_property(gws_agent, able_to(analyse_results));
has_subproperty(perform_search, interact_with_world);
has_subproperty(perform_search, analyse_results);
has_property(gws_agent, able_to(direct_search));
has_property(gws_agent, able_to(focus_search));

// define open slot requirements (***)
wim
ast
world_info
agent_info

```

A.3 Web Search Template

/* Informatie types agent_info en world_info hebben nog hun eigen templates, maar moeten misschien worden samengevoegd, omdat ze beide gebruik maken van de informatie types web_search_objs, web_loc_info, price_info en time_info */

A.3.1 Information Types

www_world_info

- infotype(web_locs)
- infotype(web_store_info)
- infotype(web_content)
- infotype(searched_web_locations)
- infotype(web_search_objective_loc)

web_locs

- infotype(web_loc_info)
 - relation(location)
 - sort(LOCATIONS)

web_store_info

- infotype(web_search_objs)
- infotype(web_loc_info)
- infotype(price_info)
- relation(search_object_in_site)
 - sort(SEARCH_OBJECTS)
 - sort(PRICE)
 - sort(LOCATION)

web_content

- sort(CONTENTS)
 - object(inhoud1)
 - object(inhoud2)

searched_web_locations

- infotype(web_loc_info)
 - relation(searched_location)
 - sort(LOCATIONS)

```

web_search_objective_loc
├── infotype(web_search_objs)
├── infotype(web_loc_info)
├── infotype(price_info)
├── relation(objective_location)
│   ├── sort(SEARCH_OBJECTS)
│   ├── sort(LOCATION)
│   └── sort(PRICE)
www_agent_info
├── infotype(web_query)
├── infotype(www_comm_method)
├── infotype(web_reply)
web_query
├── infotype(web_search_objs)
├── infotype(price_info)
├── infotype(time_info)
├── relation(find)
│   ├── sort(SEARCH_OBJECTS)
│   ├── sort(PRICE)
│   └── sort(LOCATION)
www_comm_method
├── sort(COMM_METHOD)
│   ├── object(http)
│   └── object(ftp)
web_reply
├── infotype(web_search_objs)
├── infotype(web_loc_info)
├── infotype(price_info)
├── relation(found)
│   ├── sort(SEARCH_OBJECTS)
│   ├── sort(LOCATIONS)
│   └── sort(PRICE)
web_search_objs
├── infotype(search_obj_for_comp_parts)
web_loc_info
├── infotype(web_locs_for_comp_parts)

```

```

price_info
├── sort(PRICE)
│   ├── object(cheap)
│   ├── object(normal)
│   └── object(expensive)
time_info
├── sort(TIME)
│   ├── object(one_minute)
│   ├── object(two_minutes)
│   └── object(three_minutes)

```

A.3.2 Generic Web Search

pre-conditions

structure

```

//define template
is_template(Generic Web Search);
  include_template(web_search_ast);
  include_template(web_search_wim);
  include_template(web_search_agent_info);
  include_template(web_search_world_info);

```

properties

```

/* ast en wim samen geven wel nieuwe functionaliteit (e.g. active search (search - focus control))
*/ has_property('Generic Web Search', capable_of(pro-active_behaviour));
has_property('Generic Web Search', capable_of(search));
has_property('Generic Web Search', capable_of(pro-active_search));
has_property('generic Web Search', capable_of(pro-active_web_search));
has_subproperty(pro-active_behaviour, pro-active_search);
has_subproperty(search, pro-active_search);
has_subproperty(pro-active_search, pro-active_web_search);

has_property('Generic Web Search', able_to(focus_search));
has_property('Generic Web Search', able_to(direct_search));
has_property('Generic Web Search', able_to(perform_search));

```

A.3.3 Generic Web Search AST

pre-conditions

structure

```
//define template
is_template(Web Search AST);

// define template contents
is_component(web_search);
  is_composed(web_search);
  has_input_information_type(web_search, www_agent_info, 1);
  has_input_information_type(web_search, www_world_info, 1);
  has_output_information_type(web_search, www_agent_info, 1);
  has_output_information_type(web_search, www_world_info, 1);
  is_task_control(web_search_tc);
    has_code_object(web_search_tc, 'web_search_tc.class', java);
  has_task_control(web_search, web_search_tc);

// define contained components
is_component(relevant_location_determination);
  is_primitive(relevant_location_determination);
  has_output_information_type(relevant_location_determination, web_locs, 1);
  is_knowledgebase(rel_loc_det_kb);
    is_open_slot(rel_loc_det_kb);
  has_knowledgebase(relevant_location_determination, rel_loc_det_kb);

is_component(searched_location_generation);
  is_primitive(searched_location_generation);
  has_input_information_type(searched_location_generation, web_locs, 1);
  has_output_information_type(searched_location_generation, searched_web_locations, 1);
  is_knowledgebase(sea_loc_gen_kb);
    is_open_slot(sea_loc_gen_kb);
  has_knowledgebase(searched_location_generation, sea_loc_gen_kb);

is_component(spd);
  is_primitive(spd);
  has_code_object(spd, 'web_search_spd.class', java);
  has_input_information_type(spd, web_search_objective_loc, 1);
  has_input_information_type(spd, searched_web_locations, 1);
  has_output_information_type(spd, web_locs, 1);
  has_output_information_type(spd, web_reply, 1);
  is_knowledgebase(spd_kb);
    has_code_object(spd_kb, 'web_search_spd_kb.class', java);
  has_knowledgebase(spd, spd_kb);

is_component(sra);
  is_primitive(sra);
  has_code_object(sra, 'web_search_sra.class', java);
  has_input_information_type(sra, web_store_info, 1);
  has_input_information_type(sra, web_query, 1);
  has_output_information_type(sra, web_search_objective_loc, 1);
  is_knowledgebase(sra_kb);
    has_code_object(sra_kb, 'web_search_sra_kb.class', java);
```

```

has_knowledgebase(sra, sra_kb);

has_subcomponent(web_search, relevant_location_determination);
has_subcomponent(web_search, searched_location_generation);
has_subcomponent(web_search, spd);
has_subcomponent(web_search, sra);

// define information links: relevant locs to spd
is_information_link(relevant_locs_to_spd);
  has_link_type(relevant_locs_to_spd, oo);
  has_information_type(relevant_locs_to_spd, web_locs);
  has_source_component(relevant_locs_to_spd, relevant_location_determination);
  has_source_level(relevant_locs_to_spd, 1);
  has_destination_component(relevant_locs_to_spd, spd);
  has_destination_level(relevant_locs_to_spd, 1);

is_information_link(spec_loc_obs_to_sra);
  has_link_type(spec_loc_obs_to_sra, oo);
  has_information_type(spec_loc_obs_to_sra, web_store_info);
  has_source_component(web_store_info, web_search);
  has_source_level(web_store_info, 1);
  has_destination_component(web_store_info, sra);
  has_destination_level(web_store_info, 1);

is_information_link(search_obj_to_sra);
  has_link_type(search_obj_to_sra, oo);
  has_information_type(search_obj_to_sra, web_query);
  has_source_component(search_obj_to_sra, web_search);
  has_source_level(search_obj_to_sra, 1);
  has_destination_component(search_obj_to_sra, sra);
  has_destination_level(search_obj_to_sra, 1);

is_information_link(search_results_from_spd);
  has_link_type(search_results_from_spd, oo);
  has_information_type(search_results_from_spd, web_reply);
  has_source_component(search_results_from_spd, spd);
  has_source_level(search_results_from_spd, 1);
  has_destination_component(search_results_from_spd, web_search);
  has_destination_level(search_results_from_spd, 1);

is_information_link(search_locs_from_spd);
  has_link_type(search_locs_from_spd, oo);
  has_information_type(search_locs_from_spd, web_locs);
  has_source_component(search_locs_from_spd, spd);
  has_source_level(search_locs_from_spd, 1);
  has_destination_component(search_locs_from_spd, web_search);
  has_destination_level(search_locs_from_spd, 1);

is_information_link(interpr_search_res);
  has_link_type(interpr_search_res, oo);
  has_information_type(interpr_search_res, web_search_objective_loc);
  has_source_component(interpr_search_res, sra);
  has_source_level(interpr_search_res, 1);
  has_destination_component(interpr_search_res, spd);

```

```

has_destination_level(interpr_search_res, 1);

is_information_link(locs_to_be_searched);
  has_link_type(locs_to_be_searched, oo);
  has_information_type(locs_to_be_searched, web_locs);
  has_source_component(locs_to_be_searched, spd);
  has_source_level(locs_to_be_searched, 1);
  has_destination_component(locs_to_be_searched, searched_location_generation);
  has_destination_level(locs_to_be_searched, 1);

is_information_link(searched_locs_info);
  has_link_type(searched_locs_info, oo);
  has_information_type(searched_locs_info, searched_web_locations);
  has_source_component(searched_locs_info, searched_location_generation);
  has_source_level(searched_locs_info, 1);
  has_destination_component(searched_locs_info, spd);
  has_destination_level(searched_locs_info, 1);

has_information_link(web_search, relevant_locs_to_spd);
has_information_link(web_search, spec_loc_obs_to_sra);
has_information_link(web_search, search_obj_to_sra);
has_information_link(web_search, search_results_from_spd);
has_information_link(web_search, search_locs_from_spd);
has_information_link(web_search, interpr_search_res);
has_information_link(web_search, locs_to_be_searched);
has_information_link(web_search, searched_locs_info);

properties
has_property(web_search_ast, able_to(generate_relevant_locations));
has_property(web_search_ast, able_to(reason_about(relevant_locations)));

has_property(web_search_ast, able_to(maintain_searched_locations));

has_property(web_search_ast, able_to(direct_search));
has_property(web_search_ast, able_to(analyse_search_results));

// define open slot requirements /* ? qualified ? */
sea_loc_gen_kb
has_property(sea_loc_gen_kb, capable_of(maintaining_searched_locations));
  has_property(sea_loc_gen_kb, capable_of(transforming(locations, searched_locations));
  has_property(sea_loc_gen_kb, has_input_information_type('web_locs');
  has_property(sea_loc_gen_kb, has_input_relation('location');
  has_property(sea_loc_gen_kb, has_output_information_type('searched_web_locations'));
  has_property(sea_loc_gen_kb, has_output_relation('searched_locations'));

rel_loc_det_kb
has_property(rel_loc_det_kb, capable_of(generating_relevant_locations));
  has_property(rel_loc_det_kb, has_knowledge_about(relevant_locations));
  has_property(rel_loc_det_kb, capable_of(transforming('true', location));
  has_property(rel_loc_det_kb, has_output_information_type('web_locs');
  has_property(rel_loc_det_kb, has_output_relation(location));

```

A.3.4 Generic Web Search WIM

pre-conditions

structure

```
//define template
is_template(Web Search WIM);

// define template toplevel
is_component(www_wim);
  is_composed(www_wim);
  has_input_it(www_wim, search_info, 1);
  has_input_it(www_wim, obs_results, 1);
  has_output_it(www_wim, search_info, 1);
  has_output_it(www_wim, obs_to_be_perf, 1);
  is_task_control(www_wim_tc);
  has_code_object(www_wim_tc, 'www_wim_tc.class', java);
  has_task_control(www_wim, www_wim_tc);

// define contained components
is_component(observation_formulation);
  is_primitive(observation_formulation);
  has_input_it(observation_formulation, search_info, 1);
  has_output_it(observation_formulation, obs_to_be_perf, 1);
  is_knowledgebase(obs_form_kb);
  has_code_object(obs_form_kb, 'obs_form_kb.class', java);
  has_knowledgebase(observation_formulation, obs_form_kb);

is_component(information_extraction);
  is_primitive(information_extraction);
  has_code_object(information_extraction, 'information_extraction.class', java);
  has_input_it(information_extraction, obs_results, 1);
  has_output_it(information_extraction, search_info, 1);

has_subcomponent(www_wim, observation_formulation);
has_subcomponent(www_wim, information_extraction);

// define information links
is_information_link(search_locs_to_of);
  has_link_type(search_locs_to_of, oo);
  has_information_type(search_locs_to_of, search_info);
  has_source_component(search_locs_to_of, www_wim);
  has_source_level(search_locs_to_of, 1);
  has_destination_component(search_locs_to_of, observation_formulation);
  has_destination_level(search_locs_to_of, 1);

is_information_link(obs_res_to_ie);
  has_link_type(obs_res_to_ie, oo);
  has_information_type(obs_res_to_ie, obs_results);
  has_source_component(obs_res_to_ie, www_wim);
  has_source_level(obs_res_to_ie, 1);
  has_destination_component(obs_res_to_ie, information_extraction);
  has_destination_level(obs_res_to_ie, 1);
```

```
is_information_link(selected_obs_from_of);
  has_link_type(selected_obs_from_of, oo);
  has_information_type(selected_obs_from_of, obs_to_be_perf);
  has_source_component(selected_obs_from_of, observation_formulation);
  has_source_level(selected_obs_from_of, 1);
  has_destination_component(selected_obs_from_of, www_wim);
  has_destination_level(selected_obs_from_of, 1);
```

```
is_information_link(search_results_from_ie);
  has_link_type(search_results_from_ie, oo);
  has_information_type(search_results_from_ie, search_info);
  has_source_component(search_results_from_ie, information_extraction);
  has_source_level(search_results_from_ie, 1);
  has_destination_component(search_results_from_ie, www_wim);
  has_destination_level(search_results_from_ie, 1);
```

```
has_information_link(www_wim, search_locs_to_of);
has_information_link(www_wim, obs_res_to_ie);
has_information_link(www_wim, selected_obs_from_of);
has_information_link(www_wim, search_results_from_ie);
```

properties

```
has_property(web_search_wim, able_to(interact_with(world_wide_web)));
has_property(web_search_wim, able_to(extract_information_from(world_wide_web)));
has_property(web_search_wim, able_to(initiate_observations_in(world_wide_web)));
  has_subability(interact_with(world_wide_web), extract_information_from(world_wide_web));
  has_subability(interact_with(world_wide_web), initiate_observations_in(world_wide_web));
```

```
// define open slot properties
```

```
has_property(observation_formulation, able_to(formulation_action_in(HTTP)));
has_property(information_extraction, able_to(extract_information_from(HTML)));
```

A.3.5 Generic Web Search Agent Info

pre-conditions

structure

```
//define template
is_template(generic web search agent info);

// define template contents
is_information_type(www_agent_info);
  has_information_type(www_agent_info, web_query);
  has_information_type(www_agent_info, www_comm_method);
  has_information_type(www_agent_info, web_reply);

is_information_type(web_query);
  has_information_type(web_query, web_search_objs);
  has_information_type(web_query, price_info);
  has_information_type(web_query, time_info);
  is_relation(find);
    has_sort(find, SEARCH_OBJECTS);
    has_sort(find, PRICE);
    has_sort(find, LOCATION);
  has_relation(web_query, find);

is_information_type(www_comm_method);
  is_sort(COMM_METHOD);has_object(COMM_METHOD, 'http');
  has_object(COMM_METHOD, 'ftp');
  has_sort(www_comm_method, COMM_METHOD);

is_information_type(web_reply);
  has_information_type(web_reply, web_search_objs);
  has_information_type(web_reply, web_loc_info);
  has_information_type(web_reply, price_info);
  is_relation(found);
    has_sort(found, SEARCH_OBJECTS);
    has_sort(found, LOCATIONS);
    has_sort(found, PRICE);
  has_relation(web_reply, found);

is_information_type(web_search_objs);
  is_open_slot(web_search_objs);

is_information_type(web_loc_info);
  is_open_slot(web_loc_info);

is_information_type(price_info);
  is_sort(PRICE);
  has_object(PRICE, 'cheap');
  has_object(PRICE, 'normal');
  has_object(PRICE, 'expensive');
  has_sort(price_info, PRICE);

is_information_type(time_info);
  is_sort(TIME);
```

```
    has_object(TIME, 'one_minute');
    has_object(TIME, 'two_minutes');
    has_object(TIME, 'three_minutes');
    has_sort(time_info, TIME);

properties has_property(www_agent_info, expresses(www_search_agent_info
/* etc? (***) */
// define open slot properties
has_property(web_search_objs, completes_sort(SEARCH_OBJECTS));
    has_property(web_search_objs, contains_concepts_for(domain));
    has_property(web_search_objs, has_concept_type('web_search_locations'));

has_property(web_loc_info, completes_sort(LOCATIONS));
    has_property(web_loc_info, contains_concepts_for(domain));
    has_property(web_loc_info, has_concept_type('web_locations'));
    has_property(web_loc_info, has_concept_type('web_shop_locations'));
```

A.3.6 Generic Web Search World Info

pre-conditions

structure

```
//define template
// define information types: www world info
is_information_type(www_world_info);
  has_information_type(www_world_info, web_locs);
  has_information_type(www_world_info, web_store_info);
  has_information_type(www_world_info, web_content);
  has_information_type(www_world_info, searched_web_locations);
  has_information_type(www_world_info, web_search_objective_loc);

is_information_type(web_locs);
  has_information_type(web_locs, web_loc_info);
  is_relation(location);
  has_sort(location, LOCATIONS);
  has_relation(web_locs, location);

is_information_type(web_store_info);
  has_information_type(web_store_info, web_search_objs);
  has_information_type(web_store_info, web_loc_info);
  has_information_type(web_store_info, price_info);
  is_relation(search_object_in_site);
  has_sort(search_object_in_site, LOCATIONS);
  has_sort(search_object_in_site, CONTENTS);
  has_relation(web_store_info, search_object_in_site);

is_information_type(web_content);
  is_sort(CONTENTS);has_object(CONTENTS, 'inhoud1');
  has_object(CONTENTS, 'inhoud2');
  has_sort(web_content, CONTENTS);

is_information_type(searched_web_locations);
  has_information_type(searched_web_locations, web_loc_info);
  is_relation(searched_location);
  has_sort(searched_location, LOCATIONS);
  has_relation(searched_web_locations, searched_location);

is_information_type(web_search_objective_loc);
  has_information_type(web_search_objective_loc, web_search_objs);
  has_information_type(web_search_objective_loc, web_loc_info);
  has_information_type(web_search_objective_loc, price_info);
  is_relation(objective_location);
  has_sort(objective_location, SEARCH_OBJECTS);
  has_sort(objective_location, LOCATIONS);
  has_sort(objective_location, PRICE);
  has_relation(web_search_objective_loc, objective_location);

is_information_type(web_search_objs);
  is_open_slot(web_search_objs);

is_information_type(web_loc_info);
```

```

is_open_slot(web_loc_info);

is_information_type(price_info);
is_sort(PRICE);
  has_object(PRICE, 'cheap');
  has_object(PRICE, 'normal');
  has_object(PRICE, 'expensive');
has_sort(price_info, PRICE);

is_information_type(time_info);
is_sort(TIME);
  has_object(TIME, 'one_minute');
  has_object(TIME, 'two_minutes');
  has_object(TIME, 'three_minutes');
has_sort(time_info, TIME);

properties
has_property(www_world_info, expresses(www_search_world_info));
has_property(www_world_info, expresses(web_location_info));
  has_property(www_world_info, defines(location));
  has_property(www_world_info, edxpresents(web_store_info));
  has_property(www_world_info, defines(search_object_in_site));
/* etc? (***) Sort's PRICE and TIME */

```

A.4 Computer Parts Template

A.4.1 Information Types

search_obj_for_comp_parts

```
|— sort(SEARCH_OBJECTS)
|   |— object(athlon500)
|   |— object(pent550)
|   |— object(iiyama17)
|   |— object(philips15)
```

web_locs_for_comp_parts

```
|— sort(LOCATION)
|   |— object(www_mycom_nl)
|   |— object(www_memo_nl)
```

A.4.2 Computer parts

pre-conditions

structure

```
//define template
is_template(Comp_parts);
include_template('search_obj_for_comp_parts');
include_template('web_locs_for_comp_parts');
include_template('sea_loc_gen_kb');
include_template('rel_loc_det_kb');
```

properties

```
/* combinatie kan niet meer dan som der delen, dus hier geen properties */
```

A.4.3 Searched Locations Generation KB

pre-conditions

```
available_sort('LOCATIONS');
available_input_relation('location'); /* misschien niet nodig, maar wel handig om te weten of ver-
standig om te eisen? */
```

structure

```
//define template
is_template(searched_location_generation_kb);
```

```
// define template contents
is_knowledgebase(sea_loc_gen_kb);
  has_code_object('sea_loc_gen_kb.class', java);
```

properties

```
has_property(searched_location_generation, capable_of(maintaining_searched_locations));
/* of */
has_property(searched_location_generation, able_to(maintain_searched_locations));
/* of is de kennisbank 'capable of maintaining' en de component die de kennisbank bevat 'able to
maintain' */
has_property(searched_location_generation, capable_of(transforming(location, searched_location)));
has_property(searched_location_generation, has_input_information_type('web_locs'));
has_property(searched_location_generation, has_input_relation('location'));
has_property(searched_location_generation, has_output_information_type('searched_web_locations'));
has_property(searched_location_generation, has_output_relation('searched_location'));
```

A.5 Relevant Location Determination KB

pre-conditions

```
available_information_atom('true():truthvalue');
```

structure

```
//define template
```

```
is_template(relevant_location_determination_kb);
```

```
// define template contents
```

```
is_knowledgebase(rel_loc_det_kb);
```

```
  has_code_object(rel_loc_det_kb.class', java);
```

properties

```
has_property(relevant_location_determination, capable_of(generating_relevant_locations));
```

```
has_property(relevant_location_determination, has_knowledge_about(relevant_locations));
```

```
has_property(relevant_location_determination, capable_of(transforming('true', location)));
```

```
has_property(relevant_location_determination, has_output_information_type('web_locs'));
```

```
has_property(relevant_location_determination, has_output_relation('location'));
```

A.5.1 Search Objects for Computer Parts

pre-conditions

structure

```
//define template
is_template(search_objects_for_computer_parts);

// define template contents
is_information_type(search_obj_for_comp_parts);
is_sort(SEARCH_OBJECTS);
  has_object(SEARCH_OBJECTS, 'athlon550')
  has_object(SEARCH_OBJECTS, 'pent500');
  has_object(SEARCH_OBJECTS, 'iiyama17');
  has_object(SEARCH_OBJECTS, 'philis15');has_sort(search_obj_for_comp_parts, SEARCH_OBJECTS);
```

properties

```
has_property(search_objects_for_computer_parts, contains_concepts_for(domain('computer_parts')));
has_property(search_objects_for_computer_parts, contains_concepts_for(domain('processors')));
has_property(search_objects_for_computer_parts, contains_concepts_for(domain('monitors')));
  has_subdomain(computer_parts, processors);
  has_subdomain(computer_parts, monitors);

has_property(search_objects_for_computer_parts, concept_type('search_objects'));
has_property(search_objects_for_computer_parts, concept_type('computer_parts'));
  has_subconcept(search_objects, computer_parts);

has_property(search_objects_for_computer_parts, completes_sort(SEARCH_OBJECTS));
```

A.5.2 Web Locations for Computer Parts

pre-conditions

structure

```
//define template
is_template(web_locations_for_computer_parts);

// define template contents
is_information_type(web_locs_for_comp_parts);
  is_sort(LOCATIONS);
    has_object(LOCATIONS, 'www_mycom_nl');
    has_object(LOCATIONS, 'www_memo_nl');
    has_sort(web_locs_for_comp_parts, LOCATIONS);
properties
has_property(web_locations_for_computer_parts, contains_concepts_for(domain('computer_parts')));
has_property(web_locations_for_computer_parts, contains_concepts_for(domain('processors')));
has_property(web_locations_for_computer_parts, contains_concepts_for(domain('monitors')));
  has_subdomain(computer_parts, processors);
  has_subdomain(computer_parts, monitors);

has_property(web_locations_for_computer_parts, concept_type('web_locations'));
has_property(web_locations_for_computer_parts, concept_type('web_shop_locations'));
  has_subconcept(web_locations, web_shop_locations);

has_property(web_locations_for_computer_parts, completes_sort(LOCATIONS));
```

B Java Implementation

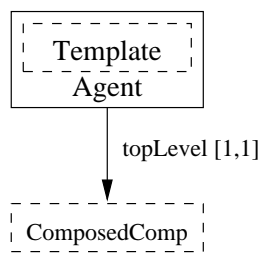
B.1 The JAVA-Agent objects

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

- **Agent** (extends *Template*): Template for an 2-agent system containing an user, an external world and one agent
- **Argument**: An argument for an *InfoAtom* or a *RuleExpression*
- **CompIO**: An input or out of a *Component*
- **Component**: A generic component
- **ComposedComp** (extends *Component*): A composed component
- **EO_InfoLink** (extends *InfoLink*): An epistemic-object link
- **InfoAtom**: An information atom from a *InfoState*
- **InfoLink**: A generic information link
- **InfoObj**: A information object for a *Sort* or a *Argument*
- **InfoState**: An information state of a *Component*
- **InfoType**: An information type
- **KB**: A knowledge base of a *PrimitiveComp*

- **Level:** A input or output level of a *Component*
- **OA_InfoLink** (extends InfoLink): An object-assumption link
- **OO_InfoLink** (extends InfoLink): An object-object link
- **PrimitiveComp** (extends Component): A primitive component
- **Relation:** A relation for a *RuleExpression* or an *InfoAtom*
- **Rule:** A rule for a *KB*
- **RuleExpression:** An expression making the conditions and conclusions of a *Rule*
- **RuleInference:** An inference engine used by the *KB*
- **SolutionArray:** A collection of possible solutions used by *RuleInference*
- **Sort:** A sort
- **TaskControl:** The task control of a *Component*
- **UserComp** (extends component): A component which interfaces with the user

B.1.1 Agent



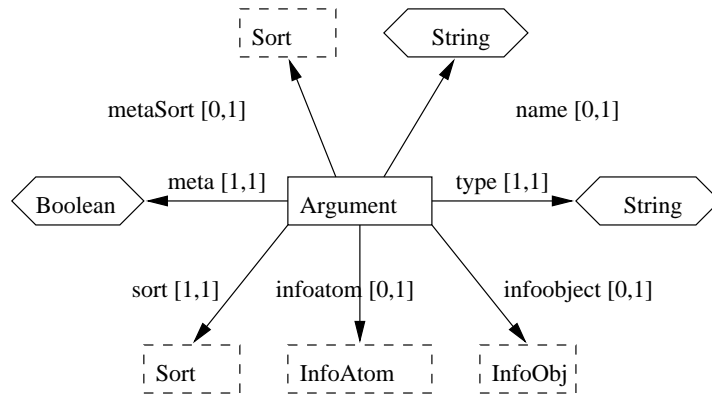
public class agent extends template

Object Design: The agent template is a template containing a composed component. This composed component contains three components representing the user, an agent and the external world. The user and the external world are implemented as commandline interfaces. The agent is an open slot which has to be filled with an agent template.

Choices and Alternatives: The external world is implemented as a commandline interface (i.e. a User Component). The external world could also be an alternative specification which translates the agent communications into actions in the 'real world'.

Possible Improvements: The external world and the user component should be open slots so that different interfaces can be used to implement these components.

B.1.2 Argument



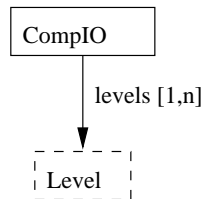
public class Argument

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.

Choises 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. Futhermore 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 als 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.

B.1.3 CompIO



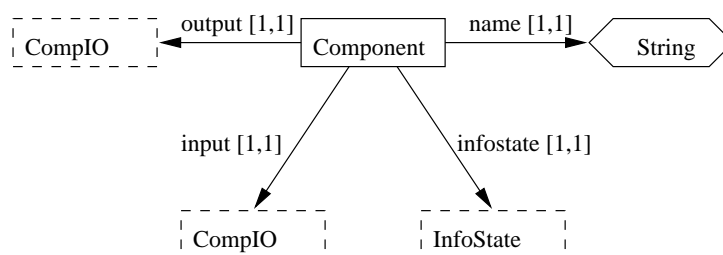
public class CompIO

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.

Choises 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.

B.1.4 Component



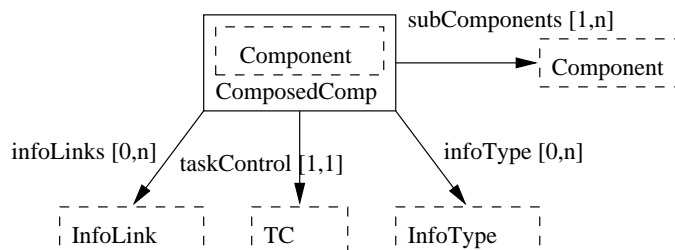
public class Component

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*.

Choises 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 dos not yet contain facilities for setting the task control focus and the extent of its task. For better task control this should be implemented.

B.1.5 ComposedComp



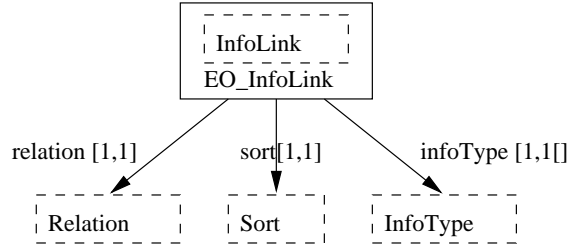
public class ComposedComp extends Component

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.

Choises and Alternatives:

Possible Improvements:

B.1.6 EO_InfoLink



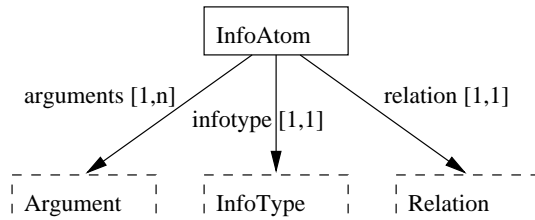
public class EO_InfoLink extends InfoLink

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.

B.1.7 InfoAtom



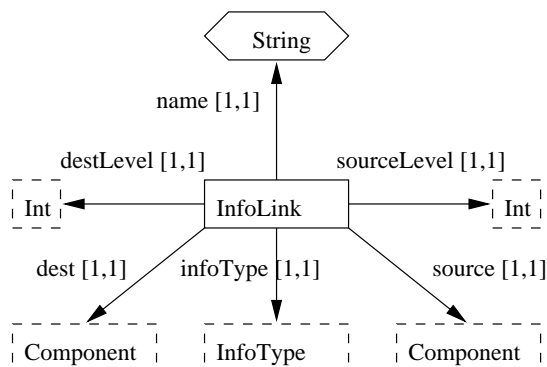
public class InfoAtom

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:

B.1.8 InfoLink

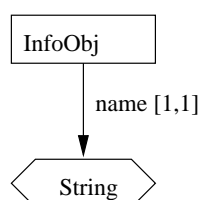


public class InfoLink

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*).

Choises and Alternatives: The information type is used to filter the information flowing trough 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.

B.1.9 InfoObj



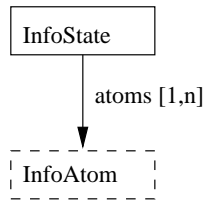
public class InfoObj

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.

Choises and Alternatives:

Possible Improvements:

B.1.10 InfoState



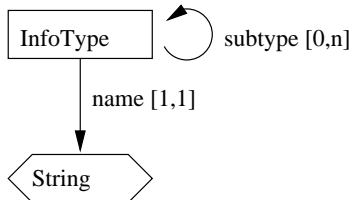
public class InfoState

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 a output atom. The information state always contains the internal information atom *true* of the information type *truthvalue*.

Choises 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.

B.1.11 InfoType



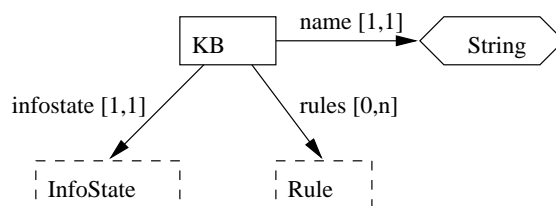
public class InfoType

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.

Choises 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.

B.1.12 KB



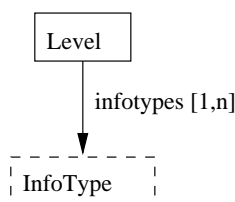
public class **KB**

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.

Choises and Alternatives:

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.

B.1.13 Level



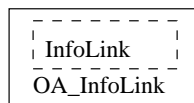
public class **Level**

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.

Choises and Alternatives:

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).

B.1.14 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.

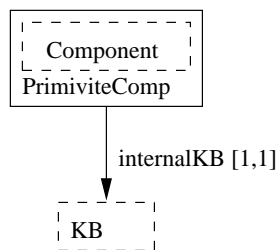
B.1.15 OO_InfoLink



*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.

B.1.16 PrimitiveComp



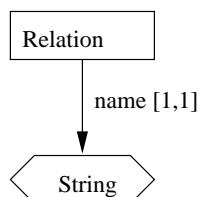
*public class **PrimitiveComp** extends Component*

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:

Possible Improvements:

B.1.17 Relation



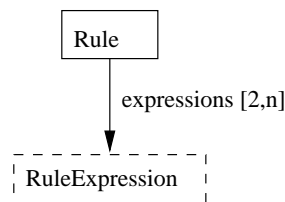
*public class **Relation***

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

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

Possible Improvements:

B.1.18 Rule



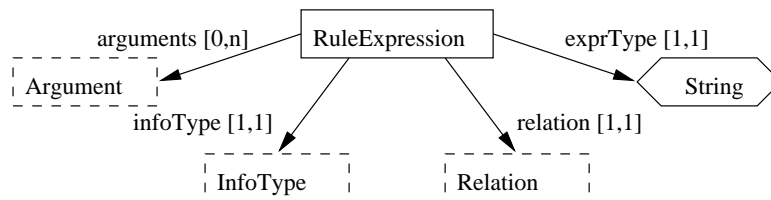
public class Rule

Object Design:

Choises and Alternatives:

Possible Improvements:

B.1.19 RuleExpression



public class RuleExpression

Object Design:

Choises and Alternatives:

Possible Improvements:

B.1.20 RuleInference

public class RuleInference

Object Design:

Choises and Alternatives:

Possible Improvements:

B.1.21 SolutionArray

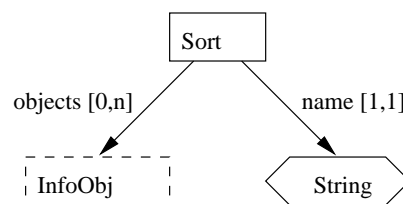
public class SolutionArray

Object Design:

Choises and Alternatives:

Possible Improvements:

B.1.22 Sort



public class Sort

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

Choises and Alternatives:

Possible Improvements:

B.1.23 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.

Choises 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.

B.1.24 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:

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.

B.2 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.

B.2.1 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:

B.2.2 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:

Possible Improvements:

B.2.3 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.

Choises and Alternatives:

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.

B.2.4 Tools

public class Tools

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

Choises and Alternatives:

Possible Improvements:

C Overview of Properties

The following properties can be used to define templates. These properties are based on the structure of DESIRE and on the properties used to describe DOD as in [Wijngaards, 1999]. First the properties used to describe the preconditions will given, then the properties used to describe the functionality of the template and third the properties used to describe the DOD.

C.1 Preconditions

- `available_sort(<sort>)`
- `available_input_relation(<relation>)`
- `available_information_atom(<info_atom>)`

C.2 Template Properties

- Generic
 - `has_property(<object>, <property>)`
 - `capable_of(<capability>)`
 - `able_to(<ability>)`
- Sort
 - `contains_concepts_for(domain(<domain>))`

- concept_type(<type>)
- completes_sort(<sort>)
- KB
 - expresses(<information>)
 - completes_sort(<sort>)
- Relation
 - relates(<sort> {'<sort>'})
 - defines(<sort>)
- Component
 - capable_of(<capability>)
 - able_to(<ability>)

C.3 Structural Properties

- Template
 - is_template(<name>)
 - is(<template>, <generic> | <toplevel>)
 - includes_template(<template>, <template>)
 - has_code_object(<template>, <filename>, <codetype>)
- Component
 - is_component(<name>)
 - is_composed(<component>)
 - is_primitive(<component>)
 - is_open_slot(<component>)
 - has_input_information_type(<component>, <infotype>, <level>)
 - has_output_information_type(<component>, <infotype>, <level>)
 - has_task_control(<component>, <taskcontrol>)
 - has_knowledgebase(<component>, <knowledgebase>)
 - has_subcomponent(<component>, <component>)
 - has_information_link(<component>, <infolink>)
 - has_information_type(<component>, <infotype>)
 - has_code_object(<component>, <filename>, <codetype>)
- Task control
 - is_task_control(<name>)
 - has_rule(<taskcontrol>, <rule>)
 - has_code_object(<taskcontrol>, <filename>, <codetype>)
- Knowledge base
 - is_knowledgebase(<name>)
 - is_open_slot(<knowledgebase>)

- has_rule(<knowledgebase>, <rule>)
- has_code_object(<knowledgebase>, <filename>, <codetype>)
- Information link
 - is_information_link(<name>)
 - is_open_slot(<infolink>)
 - has_link_type(<infolink>, <type>)
 - has_information_type(<infolink>, <infotype>)
 - has_source_component(<infolink>, <component>)
 - has_source_level(<infolink>, <level>)
 - has_destination_component(<infolink>, <component>)
 - has_destination_level(<infolink>, <component>)
 - has_eo_relation(<infolink>, <relation>)
 - has_eo_infotype(<infolink>, <infotype>)
 - has_eo_sort(<infolink>, <sort>)
 - has_code_object(<infolink>, <filename>, <codetype>)
- Information type
 - is_information_type(<name>)
 - is_open_slot(<infotype>)
 - has_information_type(<infotype>, <infotype>)
 - has_sort(<infotype>, <sort>)
 - has_relation(<infotype>, <relation>)
 - has_code_object(<infotype>, <filename>, <codetype>)
- Sort
 - is_sort(<name>)
 - has_subsort(<sort>, <sort>)
 - has_object(<sort>, <name>)
 - has_function(<sort>, <function>)
 - is_meta(<sort>)
 - has_information_type(<sort>, <infotype>)
 - has_code_object(<sort>, <filename>, <codetype>)
- Relation
 - is_relation(<name>)
 - has_sort(<relation>, <sort>)
 - has_code_object(<relation>, <filename>, <codetype>)
- Function
 - is_function(<name>)
 - has_sort(<function>, <sort>)
 - has_code_object(<function>, <filename>, <codetype>)

D Overview of Terms

property statement about a quality or trate of the partial design object

template description of a (partial) design object

desire

multi agent system (MAS)

required qualification set (RQS)

design object description (DOD)

template toplevel

E Output trace

/ memorandi remarked */*