Automated servicing of agents[*]

Frances M.T. Brazier; Niek J.E. Wijngaards
Intelligent Interactive Distributed Systems Group,
Faculty of Sciences, Vrije Universiteit Amsterdam;
de Boelelaan 1081a; 1081 HV Amsterdam, The Netherlands
frances@cs.vu.nl; niek@cs.vu.nl

Abstract

Agents need to be able to adapt to changes in their environment. One way to achieve this, is to service agents when needed. A separate servicing facility, a multi-agent factory, is capable of automatically modifying agents. This paper discusses the feasibility of automated servicing.

1 Introduction

Agents typically operate in dynamic environments. Agents come and go, objects appear and disappear, and cultures and conventions change. Whenever an environment of an agent changes to the extent that an agent is unable to cope with (parts of) the environment, an agent needs to adapt. Changes in the social environment of an agent, for example, may require modifications to existing agents. A new agent communication language, or new protocols for auctions, are examples of such changes. An agent may be able to detect gaps in its abilities; it may not be able to fill these gaps with its own built-in learning mechanisms. Whether the need for servicing is detected by an agent itself, or by another agent (automated or human) is irrelevant to the concept involved: external assistance may be needed to perform the necessary modifications.

This paper discusses the feasibility of a service for automated revision. In Section 2, needs for adaptation are discussed. An automated servicing facility, a multi-agent factory, is described in Section 3. An example of adapting an agent, based on an existing prototype automated servicing service, is provided in Section 4. The feasibility of such a service for automated revision is discussed in Section 5, in which the multi-agent factory is also compared to related approaches. The results presented in this paper are discussed in Section 6.

2 Adaptive Agents

Both static and mobile agents may encounter the need for adaptation. In this section an example is used to illustrate a few situations in which external adaptation is feasible.

The focus in this example is on an information gathering agent. The information gathering agent is assumed to be mobile. Its task is to find information for a researcher about travel arrangements needed to attend a conference. To this purpose the agent communicates with three other agents (a personal assistant agent, a travel agent, a bank agent) and interacts with the world-wide web.

Example 1. The personal assistant agent informs the information gathering agent about its preferences with respect to travel agents, and about the researcher's travel preferences. The personal assistant has acquired some of this information directly from the researcher, and has acquired some over the course of time from the researcher and from its own experience. The information gathering agent maintains a profile of the personal assistant agent, and adapts this profile on the basis of interaction with the personal assistant agent (e.g., as also encountered in negotiation settings (Bui, Kieronska, and

[*] Editors: Daniel Kudenko &Eduardo Alonso. Proceedings of the AISB-01 Symposium on Adaptive Agents and Multi-agent systems, at the Agents & Cognition AISB-01 conference. Published by the society for the study of artificial intelligence and the simulation of behaviour. ISBN 1.902956.17.0, pages 54 - 64. March 2001.

Venkatesh, 1996)). Note that in this example personification is not aimed at personalising an agent's representation of a human user (e.g., see Wells and Wolfers, 2000; Soltysiak and Crabtree, 1998), but the profile of the personal assistant.

Example 2. The information gathering agent consults the World-Wide-Web to find dates and a location for the aforementioned conference. The conference page is annotated in an ontology that is unfamiliar to the agent. For example, instead of an ontology expressed in XML (Bray, Paoli, Sperberg-McQueen, and Maler, 2000), an ontology is expressed in OIL (Fensel, Horrocks, van Harmelen, Decker, Erdman, and Klein, 2000). One way to approach this problem is to have the information gathering agent acquire understanding of this ontology. Another option is to use an intermediary agent (e.g., brokers/matchmakers (Wong and Sycara, 2000) to find an agent capable of translating between ontologies, e.g. via SOAP, (the Simple Object Access Protocol by Box, Ehnebuske, Kakivaya, Layman, Mendelsohn, Frystyk Nielsen, Thatte, and Winer, 2000). In this last case the agent needs to "travel" and collaborate with its new assistant during interaction with the conference site.

The information gathering agent also wishes to discuss possible travel arrangements with the travel agent. The travel agent indicates that it uses a specific language and protocol to discuss travel arrangements. The information gathering agent needs to acquire this agent language and protocol. This is comparable to the acquisition of a new ontology sketched above.

Example 3. During the discussion between the information gathering agent and the travel agent, the issue of credit rating arises. The information gathering agent needs to prove to the travel agent that it is trustworthy and has an acceptable credit rating. In addition to security clearance on its own trustworthiness, the information gathering agent needs additional information from the personal assistant agent. The information gathering agent needs permission to ask a bank for this information and the name and address of a specific bank agent. The bank agent, in turn, requires certificates and a guarantee from the information gathering agent that specific security measures are in place, before it will provide any other information. If the information gathering agent does not have this functionality it may be possible to add this functionality to the agent. (Please note that adding functionality may not be the only measure that needs to be taken in this case.)

In each of the situations sketched above, the agent may be adapted by an automated servicing process. The types of adaptation involved are:

? Personalisation: an agent can be provided with profiles specific to its current co-operation partners.
? Domain and languages: an agent can be adapted to include knowledge about a specific domain to understand a specific agent communication language and protocol.
? Functionality: new functionality or characteristics can be added to (or deleted from) an agent.

3 A multi-agent factory

An automated agent servicing facility, a multi-agent factory, is described in this section. The multi-agent factory, in essence, re-designs descriptions of agents. Previous research (Brazier, Jonker and Treur, 2000; Brazier, Jonker, Treur and Wijngaards, 2000) focussed on automated redesign of multi-agent systems at a detailed (conceptual) level. The automated servicing service described in this paper is an extension of this work in two aspects.

The first distinction with the previous work is that the multi-agent factory as presented in this paper is not primarily focussed on re-designing agents on the basis of first principles on a conceptual level, as described in (Brazier, Jonker, Treur and Wijngaards, 2000). The multi-agent factory uses building blocks to construct, and adapt, agents given specific templates. Templates are skeletons that describe the architecture of a (larger) part of an agent. Components are the building blocks. Templates and components are combined according to pre-defined rules.

The second distinction with previous work is a broadening of the scope of the re-design process. The multi-agent factory modifies not only the conceptual description of an agent, but also its operational code. This necessitates knowledge about the relationship between the conceptual description and operational description in a template or component.

On the basis of needs for adaptation, the automated servicing process re-configures templates and components at both levels. Re-configuration (an instance of a re-design process) of an agent first takes place at the conceptual level: templates and components are removed and added until a satisfactory conceptual agent description is acquired. On the basis of the configuration of templates and components in the conceptual description of an agent an operational description of an agent is generated.

To facilitate the automated re-design of agents, a number of assumptions have been made on the descriptions of an agent (Section 3.1). In addition, the multi-agent factory has a library of agent building blocks, the so-called templates and components (Section 3.2). The configuration task of the multi-agent factory (Section 3.5) is based on knowledge of the characteristics and properties (Section 3.3), and the availability of templates and components (Section 3.4).

3.1 Assumptions on the design of agents

The feasibility of an automated service for revision of agents depends largely on the assumptions imposed on the design of the agents. The most important underlying assumptions for an agent adaptation service used in this paper are as follows.

The first assumption is that agents have a compositional structure. A compositional structure greatly facilitates the possibilities of adding, removing and changing parts of an agent. This principle is used throughout software design, ranging from describing processes (e.g., JSD (Jackson, 1975)), via object-oriented programming (e.g., Booch 1991; Wieringa, 1996; Pressman, 1997) to component-based programming (e.g., Hopkins, 2000).

The second assumption is that re-usable parts of agents can be identified: templates (i.e., skeletons) and components (i.e., building blocks). The multi-agent factory can build an agent by correctly configuring templates and components. This assumption relates to design patterns (e.g., Gamma, Helm, Johnson and Vlissides, 1994; Peña-Mora and Vadhavkar, 1996; Riel, 1996) and libraries of software with specific functionality (e.g., problem-solving models (Schreiber, Akkermans, Anjewierden, de Hoog, Shadbolt, van de Velde, and Wielinga, 1999) or generic task models (Brazier, Jonker, Treur, 1996)).

The third assumption is that templates and components contain descriptions at two levels of abstraction: a conceptual description and an operational description. This assumption circumvents two problems. On the one hand, it is difficult to determine a conceptual description on the basis of an operational description of (part of) a system (e.g., Jackson, 1995). On the other hand, it is again difficult to determine the operational description of (part of) a system on the basis of a conceptual description (e.g., Rumbaugh, Jacobson, and Booch, 1999).

The fourth assumption is that properties and knowledge of properties are available to describe templates and components. Templates and components need to be described to facilitate their retrieval (e.g., as is done in work on describing classes of diagnostic (non-user-interactive) problem-solving methods by Benjamins (1995).)

A fifth assumption is that no commitments have been made to specific languages and/or ontologies. The languages used for the descriptions of templates and components on both their levels of abstraction are left open, as are the descriptions, and contents, of the properties and knowledge on properties to describe templates and components. The multi-agent factory is explicitly developed to be an open-architecture.

3.2 Templates and components

Templates and components are the building blocks with which agents are constructed. Templates are skeletons which describe an architecture of a (larger) part of an agent. A template may usually be combined with a number of (other) templates and/or components. A component is a building block.

For each conceptual description, a number of operational descriptions may be devised. These operational descriptions may differ in the operational language (e.g., C, C++, Java), but also in, for example, the efficiency of the operational code. A template or component contains only one combination of a conceptual description and associated operational description.

Templates and components are configurable. However, templates or components cannot be combined indiscriminately. The *open slot* concept is used to regulate the ways in which templates and components may be combined. An open slot in a template or component has associated properties at both levels of abstraction that prescribe the properties of the entity to be 'inserted'.

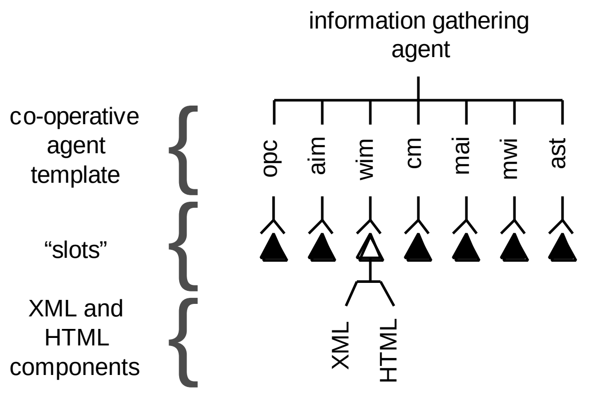


Figure 1. Graphical representation of templates and components and their slots.

This has been depicted in Figure 1, in which templates and components have slots (this specific agent is described in more detail in Section 4). Most often, templates and components are inserted in open slots of other templates, and components. Components are often inserted in open slots of other components. An open slot has to have counterparts in both the conceptual description and the associated operational description.

Templates specify the architecture of the agent. In figure 1, the information gathering agent is shown to consist of seven processes (as explained in Section 4). Each of these processes has a slot, which is filled by a combination of templates and/or components. The open slot for the world interaction management process (wim), is shown to be filled with two components, which provide specific functionality to interact with web pages annotated in HTML and XML.

The existence of open slots implies that a structure-preserving relationship between conceptual and operational descriptions with respect to slots is needed. To be more precise, at the very least an "open-slot preserving" relationship is needed, so that each open slot in a conceptual description is also in the associated operational description. In addition, the operational description has to fulfil the functional and behavioural properties stated in the conceptual description, and may have its own associated properties.

The open-slot preserving relationship between the conceptual and operational descriptions implies that templates and components are combined in the same configuration at both levels of abstraction. The two-stage revision process facilitates the generation of operational code: on the basis of the configuration of templates and components at a conceptual level, the operational code is generated in a relatively straightforward manner, as is explained in the next section.

3.3 Details of templates and components

The building blocks used by the multi-agent factory, templates and components, have the same structure, as depicted in Figure 2. This structure does not make a commitment to specific conceptual or operational description languages, but includes types of information that are also included in structures designed to describe design patterns (e.g., Gamma, Helm, Johnson, and Vlissides, 1994).

- characteristics
- conceptual pre-conditions
- operational pre-conditions
- conceptual level properties
  - template or component properties
  - open slot properties
- operational level properties
  - template or component properties
  - open slot properties
- conceptual level description
- operational level description

Figure 2. Structure of templates and components used by the multi-agent factory.

The characteristics of a building block describe its name, creation dates, authors, version information. This information is not related to the conceptual or operational descriptions inside the building block.

The pre-conditions contain assumptions and requirements that have to be satisfied by the environment (i.e., the open slot and the template containing that open slot) in which the building block is to be placed. For example, a building block which contains a specific sorting algorithm, may require as its input an unordered list of elements, where each element consists of an unknown part and an explicit key. In addition, the pre-conditions describe which languages are used in the conceptual level and operational level descriptions.

The properties of templates and components at the conceptual level are divided into properties concerning the templates and components, and properties concerning open slots. Examples of conceptual level properties of a skeleton for an agent are: it is autonomous, it is capable of communicating with other agents, it is capable of interacting in the world, it is capable of retaining information on other agents and the world. Conceptual level properties of open slots are, for example, that a specific open slot contains an agent communication language syntax expressed in XML. Template properties at the operational level include properties such as: an agent is a process, the size is so many bytes, the datastructure is of a specific class. Properties of open slots are, for example, that the first argument in a specific open slot contains the input-information for a specific process, the second argument contains a pointer to a data structure of a specific class for the results.

The conceptual level description and the operational description of a building block are not specified in more detailed, as no commitments to languages are made. One restriction, however, is that each open slot needs to appear in both descriptions.

3.4 Retrieving building blocks

The multi-agent factory is able to retrieve templates and components on the basis of needs for adaptation. The redesign process inside the multi-agent factory analyses needs for adaptation and transforms these into requirements (on structure, functionality, and behaviour) on agents to be constructed. The agent design is a configuration of templates and components that satisfies these requirements.

Matching requirements on structure, functionality, and behaviour of (parts of) agents against properties of templates and components is not trivial. Requirements can be incomplete, conflicting, or vague. To solve this problem, a matching process is needed which has some understanding of the properties involved.

Properties are related to each other in property networks. This allows generic properties to be, for example, refined into a number of sets of more refined properties. Two assumptions are made: if a more generic property of an agent holds, then at least one set of refined properties holds. If all refined properties of one set hold, then the more generic property also holds.

A number of refinements may exist for a specific property, each of which can be included in a refinement 'tree. Refinement trees can be combined into property networks. In these networks, it is possible to explore alternative refinements of a property. For example, the property that a specific algorithm is a sorting algorithm can be refined into more specific properties on efficiency, e.g. sorting algorithms in linear time, in O(nlog(n)) time, etc. Alternatively, the same property may be refined into more specific properties on the number of keys used: one key, one primary key and one secundairy key, etc. Yet another alternative is that this property is, in itself, a refined property of a property that an algorithm is a classification algorithm.

The matching process has variable forms of interpretation. One form is that no interpretation is used at all (syntactical matching), so that a required property needs to be explicitly present in a building block. An alternative is to use property refinement: a high level property (e.g., an algorithm which orders a list of elements) for which no building block can be found, can be refined into a more specific property (e.g., an algorithm which orders an array of elements in $O(n^2)$ time.), for which a building block can be found. Usually, a building block will exist with a more specific property, which can then fulfil the desired property.

A more elaborate means of query interpretation is by traversing semantic property networks. This usually returns a 'good guess', but not necessarily an optimal answer as a building block with similar properties is returned. The notion of 'similar' can be tuned (e.g., what distance to travel through property networks).

3.5 The process of adaptation

The multi-agent factory is able to adapt an existing agent on the basis of needs for adaptation. The multi-agent factory re-designs agents. The multi-agent factory first obtains an initial set of required properties (the needs for adaptation) and a description of the agent to be adapted.

The initial set of required properties is analysed and manipulated (e.g., interpreted, conflicts are resolved, etc.) to form a set of refined required properties that are still related to the initial set, yet are more specific. This may already involved checking the library of templates and components for the presence of templates and components with specific properties (it makes no sense to require, for example, a sorting algorithm in O(1/n) time if there are no such building blocks in the library).

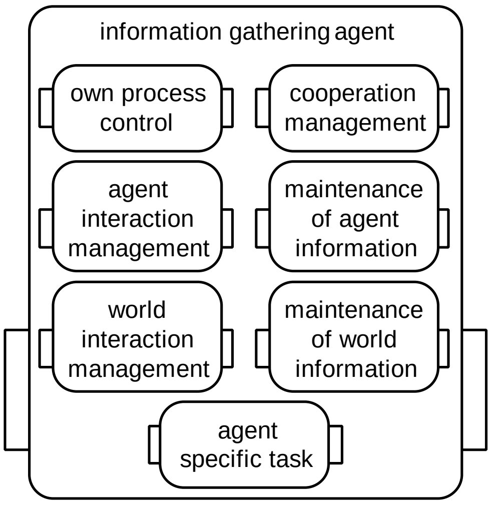
On the basis of such a more specific set of required properties, the conceptual description of the agent is adapted. Building blocks are inserted, moved, and/or deleted, until the required properties are satisfied if possible. Additional adaptation of the set of required properties may be necessary (if, for example, the requirements prove to be conflicting). A new set of required properties may be constructed, based on the previous set of required properties and evaluations of the success or failure in constructing a satisfactory conceptual description.

At some point in this cycle, it is decided whether a satisfactory conceptual description of an agent has been constructed, which satisfies a specific set of required properties (based on the initial set of required properties). If this point is reached, the multi-agent factory focuses on adapting the operational description of the agent.

The operational description of an agent is based on the configuration of templates and components in the conceptual description of the agent. If problems occur in combining operational descriptions from templates and components, either other templates and components are used (with the same conceptual description & properties, but different operational description and properties) or a different conceptual description of the agent is needed. The process described above is then repeated with additional requirements.

4 Automated servicing of an information gathering agent

In this paper an example is given of an agent that requires servicing. The adaptation of an information gathering agent in this example is based on an existing prototype automated servicing service. The conceptual descriptions of the templates and components are specified in the DESIRE knowledge-level modelling language (Brazier, Jonker, and Treur, 1998) and the operational descriptions are in Java.



information gathering agent
own process control
cooperation management
agent interaction management
maintenance of agent information
world interaction management
maintenance of world information
agent specific task

Figure 3. The seven processes inside the information gathering agent. Each process, including the agent itself, has an interface. Between processes, information transfer is defined (not shown).

The information gathering agent used in this example is based on a template containing a generic co-operative agent model (Brazier, Jonker, and Treur, 1996). Figure 3 illustrates the seven processes distinguished in this generic model. This architecture models an agent that:

? reasons about its own processes (component Own Process Control, or opc),
? communicates with other agents (component Agent Interaction Management, or aim),
? maintains information about other agents (component Maintenance of Agent Information, or mai),
? interacts with the external world (component World Interaction Management, or wim),
? maintains information about the external world (component Maintenance of World Information, or mwi),
? participates in project co-ordination (component Cooperation Management, or cm) and
? the agent's specific tasks (component Agent Specific Tasks, or ast).

This model for a co-operative agent includes components for management of its own processes, interaction with other agents including co-operation, interaction with the external (material) world, and an agent's more specific tasks. In this model, a co-operative agent receives messages from other agents, and observations in the external world (its input). It sends messages to other agents and directs its own observations and actions in the external world (its output).

In this section two examples are given of adaptation of the information gathering agent. In the first example, the information gathering agent is adapted to include functionality for understanding a new language (Section 4.1). In the second example, the information gathering agent is adapted to include new functionality for (more) secure communications and co-operations (Section 4.2).

4.1 Shallow adaptation

Figure 4 depicts the information gathering agent and shows that both the agent interaction management process (aim) and the co-operation management process (cm) are open slots. The open slot of the agent interaction management process is filled by two components providing functionality for understanding a communication language with the personal assistant and an information provider (e.g., which provides access to the world-wide-web). The open slot of the co-operation management process is filled by two components providing functionality for understanding how to co-operate (protocols) with the personal assistant and an information provider.

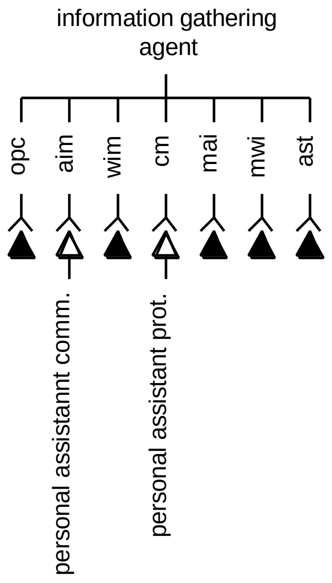


Figure 4. Partial description of the information gathering agent. The open slots for the agent interaction management and co-operation management processes are filled with two components each: agent communication languages and protocols.

One of the needs for adaptation identified in Section 2 was that the information gathering agent needed to interact with the travel agent. The travel agent was able to indicate that a specific language and protocol was to be employed. One way for the information gathering agent to approach this problem is to use the multi-agent factory to have itself changed, such that it can understand the languages and protocols needed for interaction with the travel agent.

In the first case, the multi-agent factory searches its libraries of templates and components and is able to find two components that support the functionality required. In addition, these two components contain descriptions on two levels of abstractions, and each description needs to be linked to the, already existing, description of the information gathering agent. This results in a description of the information gathering agent, as depicted in Figure 5.

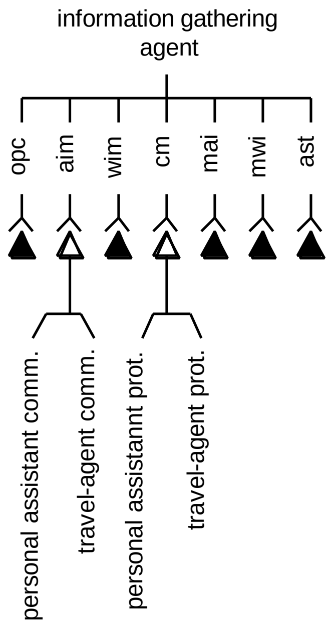


Figure 5. The information gathering agent is adapted to include functionality on a language and protocol for interaction with the travel agent.

First the new components are inserted into the conceptual description of the agent. Once this has been achieved successfully, the operational parts of the components are inserted into the operational description of the agent.

4.2 Deep adaptation

In another example in Section 2, one of the needs for adaptation arises from communication with a bank agent. This agent requires that the information gathering agent uses specific security functionality. Again, the information gathering agent uses the multi-agent factory to have itself adapted.

The multi-agent factory now has two goals in adapting the information gathering agent. Specific security functionality needs to be added, and functionality for understanding a language and protocol shared with the bank agent. The latter case has been described in the previous subsection.

Adapting the information gathering agent to include specific security functionality is translated by the multi-agent factory to the need to adapt the agent to include functionality for secure communication and secure co-operation (as in both processes security-related awareness is needed). Two templates have been retrieved from the library available to the multi-agent factory: a template for secure communication and a template for secure co-operation. Both templates can be used together, as can be derived form their characteristics, and both templates can be embedded in the current configuration of templates and components.

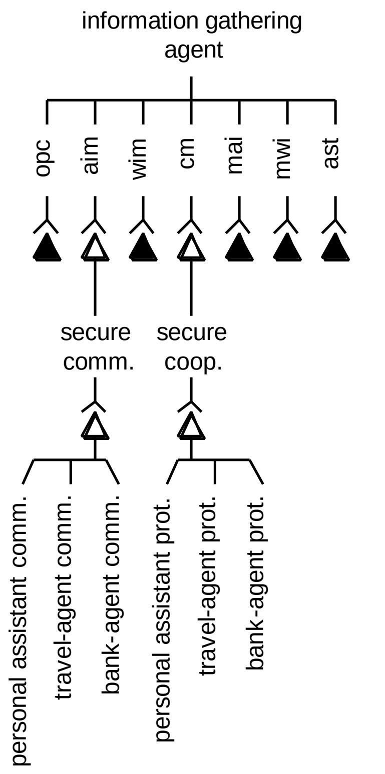


Figure 6. The information gathering agent is adapted to include functionality on secure communication and co-operation, and functionality on understanding a language and protocol for interaction with the bank agent.

As is shown in figure 6, the information gathering agent is modified in a non-trivial manner: the two new templates are inserted into two of the open slots of the top-most template, and the original fillings of these open slots are inserted into the open slots of the new templates.

5 Feasibility

The feasibility of the multi-agent factory hinges on a number of aspects. These aspects are briefly described in Section 5.1. A comparison of the multi-agent factory to other approaches in constructing agents in described in Section 5.2.

5.1 Crucial aspects

A number of aspects are crucial to the feasibility of the multi-agent factory. These aspects are mainly concerned with templates and components. Inserting templates or components in an open slot of a template or component involves understanding

? the properties associated with the open slot, which prescribe properties of the entities to be inserted,
? how properties relate to each other,
? how the conceptual description of the template or component to-be-inserted, can be connected to the open slot (this may involve a mapping between two different conceptual description languages),
? how the operational description of a template or component to-be-inserted, can be connected to the open slot (this may involve a mapping between two different operational description languages), and
? how multiple components can be inserted into one open slot (cf. to stacking blocks), especially when different description languages are employed.

Experience with the current prototype has increased confidence in the feasibility of the multi-agent factory, but more research is needed (and is being conducted).

5.2 Comparison to other approaches

The multi-agent factory is compared to component-based development, agent construction kits, software reusability, case-based reasoning, and configuration design.

The multi-agent factory's approach to combining templates and components seems similar to the approach taken in component-based development (CBD) of software (Hopkins, 2000; Sparling, 2000). One distinction with our approach is that our approach includes annotations of templates and components at two levels of abstraction (conceptual and operational). In CBD, interfaces are described for components (which are independent of an operational language); this correlates to the open slots in templates and components. From our perspective CBD provides a useful means to describe operational descriptions of the building blocks used by the multi-agent factory.

Currently a relatively large number of tools and/or frameworks exists for the (usually semi-automatic) creation of agents (not automated adaptation). Examples include e.g. AgentBuilder (Reticular Systems, 1999), D'agents/AgentTCL (Gray, Kotz, Cybenko, and Rus, 1997), and ZEUS (Nwana, Ndumu, Lyndon, and Collis, 1999). All of these approaches commit to a specific operational description of agents, and usually also commit to a specific conceptual description of their agents. The multi-agent factory does not make such commitments, which makes the multi-agent factory more general purpose (with all the common advantages and disadvantages).

The multi-agent factory aims at pragmatically circumventing a number of issues related to software reusability (e.g., Biggerstaff and Perlis, 1997). A major problem is annotating reusable pieces of software such that they can be retrieved at a later time (by other people) and reused with a minimal number of changes. In the multi-agent factory the latter is endeavoured as well. The former is currently solved in a pragmatic way: templates and components are annotated, and, when needed, a mapping is provided to other annotations. This, however, is not a scalable solution, and, as such, one of our current foci of research. An important decision concerning standardisation is that the multi-agent factory does not aim to adhere to one specific standard, but a number of standards.

In case-based reasoning approaches (e.g., Kolodner, 1993; Watson and Marir, 1994) libraries of cases are consulted to find a case which matches a problem, upon which the retrieved case is adapted. This approach differs from the multi-agent factory in that cases are modified internally, instead of combined with other cases. Techniques for retrieving cases from case libraries are, of course, relevant to retrieving templates and components from libraries.

The approaches taken by design-as-configuration (e.g., as described in (Stefik, 1995), CommonKads (Schreiber, Akkermans, Anjewierden, de Hoog, Shadbolt, van de Velde, and Wielinga, 1999), and elevator configuration (Schreiber and Birmingham, 1996)) focus on constructing a satisfactory configuration of elements on the basis of a given set of requirements (also named: constraints). In most of these approaches no explicit manipulation of requirements is present, nor is a multi-levelled description of the elements taken into account. Models and theories on configuration-based design are relevant to the multi-agent factory, in particular to the processes involved in combining conceptual and operational descriptions.

6 Discussion

An automated servicing process for agent adaptation is described in this paper. This servicing process is realised by a multi-agent factory. Agents are constructed from templates and components. Adapting an agent entails adapting the configuration of templates and components.

The five assumptions underlying our approach are: (1) agents have a compositional structure, (2) re-usable parts of agents can be identified, (3) two levels of descriptions are used: conceptual and operational, (4) properties and knowledge of properties are available, and (5) no commitments are made to specific languages and/or ontologies.

The main advantage of a multi-agent factory as an automated servicing process is that an agent can easily obtain new functionality, without obliging the agent itself to have its own adaptation mechanism. During their lifetime agents acquire new skills and knowledge.

The multi-agent factory is still being researched; the current research focuses on:

? building a library of templates and components,
? designing description languages for properties of, and knowledge on the use of, templates and components,
? learning from experiences with different conceptual and operational description languages,
? designing and implementing more extensive prototypes of the multi-agent factory,
? investigating security and trust in using a multi-agent factory.

Acknowledgements

The authors wish to thank the graduate students Hidde Boonstra and David Mobach for their explorative work on the application of a multi-agent factory for an information retrieving agent.

References

G. Booch. *Object oriented design with applications*. Redwood City: Benjamins Cummins Publishing Company. 1991.

D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Frystyk Nielsen, S. Thatte, D. Winer. Simple Object Access Protocol (SOAP) 1.1. *W3C Note*, http://www.w3.org/TR/SOAP/, 08 May 2000.

V. R. Benjamins. Problem-Solving Methods for Diagnosis and their Role in Knowledge Acquisition. *International Journal of Expert Systems: Research & Applications*, 8(2):93-120, 1995.

T. J. Biggerstaff, and A. J. Perlis (eds.). *Software Reusability. Volume 1, Concepts and models*. New York: ACM Press, 1997.

T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0, (Second Edition), *W3C Technical Report 20001006*, http://www.w3.org/TR/2000/REC-xml-20001006, 2000.

F. M. T. Brazier, J. M. Jonker, and J. Treur. Modelling project coordination in a multi-agent framework. In: *Proc. Fifth Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WET ICE'96*. Los Alamitos: IEEE Computer Society Press, pp. 148-155, 1996.

F. M. T. Brazier, C. M. Jonker, and J. Treur. Principles of Compositional Multi-agent System Development. In: J. Cuena (ed.), *Proceedings of the 15th IFIP World Computer Congress, WCC'98, Conference on Information Technology and Knowledge Systems, IT&KNOWS'98*, pp. 347-36, 1998.

F. M. T. Brazier, C. M. Jonker, and J. Treur. Compositional Design and Reuse of a Generic Agent Model. *Applied Artificial Intelligence Journal*, 14:491-538, 2000.

F. M. T. Brazier, C. M. Jonker, J. Treur, and N. J. E. Wijngaards. Deliberate Evolution in Multi-Agent Systems. In: J. Gero (ed.), *Proceedings of the Sixth International Conference on AI in Design, AID'2000*. Kluwer Academic Publishers. pp. 633-650, 2000.

H. H. Bui, D. Kieronska, and S. Venkatesh. Learning other agents' preferences in multiagent negotiation. In: *Proceedings of the National Conference on Artificial Intelligence (AAAI-96)*, pp. 114-119, 1996

D. Fensel, I. Horrocks, F. van Harmelen, S. Decker, M. Erdmann, and M. Klein. OIL in a nutshell. In: R. Dieng (ed.), *Proceedings of the 12th European Workshop on Knowledge Acquisition, Modelling, and Management (EKAW'00)*, Springer-Verlag, Lecture Notes in Artificial Intelligence, 1937:1-16, 2000.

R. Gray, D. Kotz, G.e Cybenko, and D. Rus. Agent Tcl. In: W. Cockayne, and M. Zyda (eds.), *Mobile Agents: Explanations and Examples*, Manning Publishing, 1997.

E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of reusable object-oriented software*. Reading, Massachusetts: Addison Wesley Longman, 1994.

J. Hopkins. Component Primer. *Communications of the ACM*, 43(10):27-30, 2000.

M. A. Jackson. *Principles of Program Design*, Academic Press. 1975.

M.A. Jackson. *Software Requirements and Specifications*. Wokingham, England: Addison-Wesley, 1995.

J. L. Kolodner. *Case-Based Reasoning*. San Mateo, California: Morgan Kauffman, 1993.

H. Nwana, D. Ndumu, L. Lyndon, and J. Collis. ZEUS: A Tookit and Approach for Building Distributed Multi-agent Systems. In: *Proceedings of the Third International Conference on Autonomous Agents (Autonomous Agents'99)*, pp. 360-361, 1999.

F. Peña-Mora, and S. Vadhavkar. Design Rationale and Design Patterns in Reusable Software Design. In: J. S. Gero, and F. Sudweeks (eds.), *Artificial Intelli-*

gence in Design (AID'96), Dordrecht: Kluwer Academic Publishers, pp. 251-268, 1996.

R. S. Pressman. *Software Engineering: A practitioner's approach*. Fourth Edition, McGraw-Hill Series in Computer Science, New York: McGraw-Hill Companies Inc. 1997.

Reticular Systems Inc. AgentBuilder: An integrated toolkit for constructing intelligent software agents. *White Paper,* http://www.agentbuilder.com, February 1999.

A. J. Riel. *Object-Oriented Design Heuristics*. Reading Massechusetts: Addison Wesley Publishing Company, 1996.

J. Rumbaugh, I. Jacobson, G. Booch. *The unified modeling language reference* manual. Reading, Massachusetts: Addison Wesley, 1999.

G. Schreiber, H. Akkermans, A. Anjewierden, R. de Hoog, N. Shadbolt, W. Van de Velde, and B. Wielinga. *Knowledge Engineering and Management, the CommonKADS Methodology*. MIT press, 1991.

G. Schreiber, and W. P. Birmingham (eds.). Special Issue on Sisyphus-VT. *International Journal of Human-Computer Studies (IJHCS),* 44, 1996.

S. Soltysiak, and B. Crabtree. Knowing Me, Knowing You: Practical Issues in the Personalisation of Agent Technology. In: *Proceedings of the third international conference on the practical applications of intelligent agents and multi-agent technology* (PAAM98), London, 1998.

M. Sparling. Lessons learned through six years of component-based development. *Communications of the ACM*, 43(10):47-53, 2000.

M. Stefik. *Introduction to Knowledge Systems*. San Francisco, California: Morgan Kaufmann Publishers Inc., 1995

I. Watson, and F. Marir. Case-based reasoning: a review. *The Knowledge Engineering Review,* 9(4):327-354, 1994.

N. Wells, and J. Wolfers. Finance with a Personalized Touch. *Communications of the ACM,* Special Issue on Personalization, 43(8):31-34, 2000.

R. J. Wieringa *Requirements Engineering: Frameworks for Understanding*. Wiley and Sons, 1996.

H.-C. Wong, and K. Sycara. A Taxonomy of Middle-agents for the Internet. In*: Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS'2000),* 2000.