Methodology for Agent-Oriented Software Design

Principal Investigator dr. Frank S. de Boer (frankb@cs.uu.nl)

Summary

The main research goal of this project is the investigation and development of a methodology for the description and design of software architectures based on the agent-oriented approach. Agents provide a high abstraction level for developing software and thereby potentially simplify the design of complex information systems. The specific features of agent-oriented programming, which involve the autonomy of agents and their high-level communication and interaction, require the development of new concepts and techniques which are needed to structure the design process. Our aim is to provide these new techniques and analyse the conceptual basis of the agent-oriented approach.

The main research questions associated with the overall research goal involve:

- The development of component-based software techniques for the integration of components like database software applications into agent systems,
- The development of generic architectures for agents (roles),
- The design of patterns of information exchange between agents, and
- The construction of useful structures for sets of agents into groups,
- The implementation of tools and a programming environment for the design of agents,
- The implementation of a substantial test case.

Funding

This project has funding for one postdoc (3 years) or AIO (4 years) and one scientific programmer (for 3 years).

Research Proposal

The concept of an agent constitutes a new and promising paradigm for the realisation of a new generation of computational systems which assist, actively support, and in some cases even model (certain aspects of) the user. For example, applications based on agent-technology can be found on the Web (e-commerce, search agents, etc) and in applications supporting business processes where the information flow is high (personal assistants). Agent-oriented programming offers a new perspective on software design for managing the inherent complexity of software systems. This alternative provided by the agent-oriented approach is based on a set of concepts which provides both for an intuitive as well as a structured software design approach. The approach naturally incorporates different abstraction levels for analysing
a software system and a high-level perspective on system design. It thereby facilitates the analysis and, more specifically, the architectural design phase in the software development process.

Similar to the concept of objects, the concept of an agent serves as a heuristic guideline in the development of software systems. In the literature, there is a rich and diverse set of proposals for a definition of an agent which show a clear family resemblance. Moreover, there is a strong emerging consensus on the set of concepts that is computationally useful to associate with the notion of an agent. These concepts include the notions of belief, goal, plan, role, and group. Associated with the latter concept are issues like social abilities which involve communication and the coordination of activities. These concepts relate naturally to typical software engineering issues concerning the interaction of software components and the responsibilities and services required of such components. In particular, these concepts are well-suited to the design of the type of applications mentioned above. Moreover, it can be argued that these concepts provide a more intuitive basis for software design of these types of applications than, for example, object-oriented design. The reason is that it is more difficult to comprehend system functionality in terms of objects than in terms of actions, functions or processes as is affirmed by several authors in the field of software engineering and object-oriented design.

A lot of effort has been devoted into making good engineering sense out of these concepts. This effort has resulted in a number of architectures and programming frameworks for agents, but, as yet, there still has been relatively little systematic research which addressed the issue of practical methods for the design of agent-based systems. This is due to the inherent complexity of building agent systems in by now established programming paradigms like object-oriented programming and a lack of consensus on a clear and dedicated programming framework for realising agent systems. This inherent complexity of using traditional approaches for the engineering of agent systems leads to ad hoc solutions and in practice to building these systems each time from scratch. Our overall research goal has been a programming methodology to support the agent-oriented approach to overcome these shortcomings of current agent engineering practice.

The Agent Framework 3APL

We think it is essential that agent terminology is grounded in a computational clear and precise way by providing a dedicated programming environment. Object-oriented languages are primarily designed for modelling a world of objects and their relations and as such these languages and their corresponding methods (like UML) do not provide appropriate high-level abstraction mechanisms necessary for the design and modelling of multi-agent systems. For this purpose, an abstract agent programming framework called 3APL (pronounced "triple-a-p-l") has been developed which provides a set of programming constructs which give a precise meaning to the basic concepts in the agent-oriented approach, including the notions of belief, goal and plan. 3APL combines both logic and imperative programming. The incorporation of logic programming into 3APL provides for the information querying capabilities of an agent and the incorporation of imperative programming provides for the structuring of the goals and skills of agents. 3APL Agents are software entities consisting of beliefs, goals and plans, which are capable of communicating messages concerning their beliefs and goals. This work on 3APL has been reported in various international conferences and journals (for example, best student paper of ATAL98). This framework is closely related
to a number of other frameworks in the field like AgentSpeak(L), ConGolog, and AGENT0. But 3APL has the benefit over these other approaches of providing a more general as well as a more simple and easier to understand programming framework. Moreover, 3APL is a very powerful combination of the well-known programming paradigms of imperative and logic programming which enables the use of well-understood techniques from these areas as an extra benefit.

3APL offers a framework that fixes the basic structure of the software system based on the agent-oriented perspective. The main components of an agent system, like its database or basic operations, however, are plug-in features in 3APL which facilitate the use and integration of techniques from databases, knowledge engineering, user interfaces, etc. and thereby supports reuse. Thus viewed, 3APL offers a perspective similar to that of component-based software engineering with the added value of a number of architectural or structuring principles based on the notion of an agent. This commitment to a set of clearly defined conventions and constraints in the agent-oriented approach thus has the benefit associated with the explicit introduction of a number of guiding principles for the design of software systems.

Agent Methodology

The potential of agent-based systems is large and it remains to exploit this new technology. As mentioned already, the technology to date, however, is still often used ad hoc. The success of agent-based systems in the future, in contrast, in our view depends on a thorough analysis of its conceptual basis and the construction of structuring principles and methods for the design of agent systems. Despite the many proposals in the literature to support the construction and design of agents by means of specific agent architectures or programming languages, there are still few methodologies for the development of multi-agent systems. Some promising efforts in providing a methodology for agent-oriented software engineering derived from object-oriented methodologies have been proposed. These proposals provide a first step towards an approach to software design based on agent-related concepts. Nevertheless, the additional complexity of inter-agent communication and coordination requires a major effort which involves the development of substantially new concepts which is not fully accomplished in the proposals so far. These new concepts are related to the high-level nature of the information present in multi-agent systems. The high-level nature of the information present in multi-agent systems which involve, for example, mechanisms for update and belief revision, in general introduces a completely new dimension to existing design methodologies.

The aim of the agent-oriented approach is to facilitate analysis, design and implementation, at a level of abstraction that is more adequate for complex information systems, by introducing the single, main concept of agent. We provide an outline of the basic ideas that we think are useful to this end and a preliminary sketch of an agent-oriented design methodology. Of course, it is the purpose of this proposal to refine, evaluate and test, and extend this sketch of a methodology. The sketch provided here serves to illustrate the main ideas and to introduce the main research issues. The basis for a methodology for agent-oriented design can be derived from the most important concepts in the agent-oriented approach, namely that of agent and group. A third design issue - that of the interaction between agents and groups - is inferred immediately from these notions. One of the basic concepts in our proposal for a methodology is that of a role. Roles are associated with agents and are composed of skills, knowledge and goals. In our design methodology a number of steps concern the further refinement of these
skills, knowledge and goals of roles.

We propose that an agent-oriented methodology should consist of the construction of five main models: agent/role models, skill models, knowledge models, an organisational model, and an interaction model. Each of the steps in the methodology below result in the construction of one (or more) of the corresponding models. The methodology for their elaboration and refinement can be expressed in five steps.

1. The first step consists of the identification of the roles in the application domain in terms of the corresponding goals or tasks. Consequently, a role is identified by the main tasks or responsibilities required by the system and forms the basis for a specification of agent types analogously to the concept of class in object-oriented programming (OO). There is a close link here with similar concepts in OO. The research issues involve defining notions like inheritance, agent classes, aggregation, etc in the agent context.

2. Secondly, identify the main skills associated with agent roles - skills consist of the responsibilities and basic services required to be able to perform a role. Special attention will be addressed to the issue of agent-wrapping: the integration of non-agent software components in this step, and the design of appropriate interfaces.

3. A third step consists of the modelling of the knowledge about the application domain associated with identified roles or skills and should result in the design of an adequate ontology. Techniques from knowledge engineering can be used here, and again the issue of reuse and integration of existing ontologies will be addressed.

4. Fourthly, an organisational structure (architecture) of the multi-agent system should be designed. The coordination between agents to achieve a task and the required communication schemes need to be analysed in this step, and group structures need to be identified. Groups allow sets of agents to be treated as a single conceptual entity. The main research themes related to this step concern the design of generic group structures, communication schemes and the overall infrastructure of the multi-agent system.

5. Finally, the dynamics of the multi-agent system should be analysed in terms of the flow of information, resulting in, for example, synchronisation requirements derived from the roles associated with individual agents. In this step dependency-relations between roles and agents should be modelled. An important research issue here concerns the investigation of the flow of information of the different agent roles in relation to the (multi-)agent architecture(s).

The methodology outlined above is grounded in the agent programming framework 3APL which facilitates both the internal design of agents as well as the implementation of the multi-agent system. 3APL grounds the methodology in providing clear interpretations of the basic notions of belief (step 3 above), and goals and plans of an agent (built by composing basic skills as analysed in step 2). Finally, the communication primitives of 3APL provide for high-level information exchange capabilities of agents which are suitable for refining and implementing the communication schemes of step 4. Apart from the analysis steps 1 and 5, 3APL provides thus a concrete platform which substantially reduces the effort needed to refine three of the five types of models which are constructed during the design of a software system.
We are currently implementing a programming environment for 3APL. This implementation provides a test-bed for the evaluation of a methodology for the design of agent systems and the construction of an agent application.

We now summarise the various themes which we believe should play a major role in an agent-oriented methodology and elaborate on some of the associated issues which require further investigation. The themes correspond with issues which need to be addressed in the first four steps of the methodology.

**Agent Roles**

In elaborating and refining the methodology for agent-oriented design, we aim at a further development of the concept of a class from object-oriented programming, which we call a role. In contrast to a class, however, roles offer a more goal or action-oriented perspective. A role should specify at an appropriate abstraction level the set of possible behaviours of an agent in terms of the services associated with it and its high-level interaction capabilities. Roles can be organisational - in case they are required by system implementation - or functional - in case they are directly related to the application. The concept of a role is expected to provide a basis for the description and development of generic agent architectures. Typical examples include agent classes like brokers, mediators, and facilitators used for delivering information to the agents with the appropriate processing skills, and contracting agents to perform services associated with their roles. As an abstract specification formalism for roles, we envisage a graphical notation, like for example UML, with a declarative interpretation.

**Agent Wrapping**

Another important design issue concerns the integration of existing software, like existing database applications, and software techniques into agent-based systems. Agent-based systems constructed in 3APL are not introduced as a new and self-contained way of building software. In contrast, 3APL provides a programming environment that explicitly aims at the exploitation and reuse of existing software. This aim, however, poses new challenges and design choices as to how to integrate a wide range of heterogeneous system components into the basic agent design and finally how to refine this design into the basic structure offered by 3APL. The choices involve the integration of existing information sources like databases and the Web, the sharing, manipulation and accessing of information between agents, and issues like security and privacy.

**Knowledge Modelling** First of all, the focus in the agent-oriented approach is on the processing of information and therefore it is of vital importance that the beliefs of an agent are adequately modelled when designing agent-based systems. More specifically, the combination of issues like persistence, and the reliability of other agents introduce a very different and new picture.

A characteristic of 3APL is that it does not fix the type of information that is processed or exchanged between agents and as a consequence does not impose many restrictions (as some existing agent architectures do) but allows for a wide range of information agents. As such, it provides a generic mechanism for turning a database into an agent, that is, a generic mechanism for agent wrapping of databases but at the same time raising the important issue of
designing appropriate ontologies and associated problem solving methods.

**Coordination and Groups**

Agent-based systems provide for different levels of abstraction which are built into the very notion of a multi-agent system. This provides for a potentially powerful and new design approach. It also poses the problem of structuring and articulating the different levels in such a way that they are most useful for the design of the type of information systems considered here. Apart from the design level of the individual agent, a no less important level is that of the multi-agent system itself. This level concerns the interoperability of agents. The notion of a group here complements that of a role. Similar issues like finding a balance between private and public information reappear and new issues like group-structure, and hierarchy relations between agents need to be investigated. In particular, the strategies or protocols for information exchange and negotiation between agents, within and outside a group, provide new challenges. A major challenge therefore consists in the development of design patterns for the distribution of databases among agents.

**Programming Environment and Application**

At a more practical level the research described above should lead to the development of tools to support the design of agent-based systems in 3APL by means of a programming environment and the actual design and implementation of a substantial application. Related to this we would like to develop libraries consisting of basic agent skills and generic agent types or architectures, like, for example, a broker agent. A particularly interesting application, both from a practical as well as from a more scientific perspective, are so-called personal assistants. These type of agents involve activities like maintaining an agenda of their user, actively filtering and suggesting relevant information sources, and negotiating with other agents.

**Work Programme**

<table>
<thead>
<tr>
<th>Activities Postdoc</th>
<th>Activities Programmer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>first year:</strong></td>
<td></td>
</tr>
<tr>
<td>1 Design of a language with a graphical notation</td>
<td>Develop programming environment</td>
</tr>
<tr>
<td>for roles/goals/knowledge/skills/</td>
<td></td>
</tr>
<tr>
<td>2 Design of generic agent architectures</td>
<td>Pilot Project</td>
</tr>
<tr>
<td><strong>second year:</strong></td>
<td></td>
</tr>
<tr>
<td>3 Design of a personal assistant</td>
<td>Development of libraries of agent types</td>
</tr>
<tr>
<td>4 Design patterns for agent wrapping</td>
<td>Implement personal assistant</td>
</tr>
<tr>
<td><strong>third year:</strong></td>
<td></td>
</tr>
<tr>
<td>5 Design of communication protocols</td>
<td>Libraries for agent components and communication protocols</td>
</tr>
</tbody>
</table>
The postdoc/aio will focus on the overall methodology first and extend, add detail and evaluate the ideas outlined in the proposal. In particular, the postdoc/aio will design a language with a graphical notation and a clear semantics that can be used to formally develop agent-based systems. This formalism and the methodology will then be put to use for the design of generic agent architectures. In the second year, this work will be used to design a substantial multi-agent system like, for example, personal assistants supporting organisational tasks (which include, for example, meeting scheduling, agenda management, email filtering, information retrieval, etc). During the project special attention will be directed to the issue of reuse and the integration of legacy software, in this context also known as agent wrapping. After the design of a substantial agent application, agent wrapping will be analysed in more detail and design patterns for agent wrapping will be developed. Finally, in the third year, the main effort will be put into the design of communication protocols.

The scientific programmer will continue the development of an efficient implementation of the programming language and especially focus on the development of a supporting programming environment (tools for program development) first. These activities complement those of the postdoc/aio in creating a platform for the realisation of agent systems based on the proposed methodology. As soon as the programming environment has been finished to the extent that it allows the construction and development of a realistic agent application, a pilot project will be started in close cooperation with the postdoc/aio that is aimed at the development of a number of small, instructive examples, concerning, for example, integrating Internet facilities in agents (search engines, etc.). The overall goal of this pilot project is to test a number of ideas concerning the design of generic agent architectures. The experience gained during this pilot project is also used to provide feedback on the language developed by the postdoc/aio. When this pilot project is finished, the scientific programmer starts with the construction of libraries for generic agent types. Moreover, the scientific programmer will be involved in the implementation of a design of an agent system as a test case for providing feedback both on the methodology as well as on the programming language 3APL (need for additional constructs, modifications, etc). Finally, based on the work of the postdoc/aio on agent wrapping and communication protocols, the scientific programmer will create and develop libraries which can be used to deal with each of these issues in the second and third year.