# A toolkit for building component based agents

**Nikolaos Skarmeas** and **Keith L. Clark**
Department of Computing
Imperial College, London, UK
**email:** {ns4,klc}@doc.ic.ac.uk

## Abstract

A toolkit for configuring agent based applications is presented. Each agent comprises several components (some generic, some agent specific). All the components run as separate internal processes and communicate via an internal active message board component This indirect method of communication allows us to more easility mix and match components. The code for all the components that can be assembled into an agent are held on a code server. Standard or often used configurations of components can be symbolically described and stored with a suitable name in an agent library. Finally, an agent management platform allows a user to specify that a particular named type of agent should be launched on a particular machine. The platform will pull down the description for the agent library, pull down the code for the components for the code server, and launch the agent as an integrated set of component processes on the machine.

## Introduction

With the advances in Object Oriented technology, the construction of off-the shelf components that can be used and re-used for the construction of large software systems has become fashionable. This leads to a new style of software construction called *component oriented* (Meijler & Nierstrasz 1998), (Buschmann et al 1996). In parallel, especially for distributed applications, another style of programming is often used, called *agent oriented.* In this style of programming the entities of the application are viewed as agents which are capable of accomplishing complex tasks.

A number of software architectures have been proposed for building agents. A common approach is the separation of the agent functionalities into two main categories: the domain independent and the domain dependent one. The domain independent part deals with the communication oriented activities of the agent and other features such as knowledge base management. The domain dependent part of the agent deals with the (possibly local) problem solving activities of the agent. It usually consists of a number of quasi-independent modules, different agents comprising different collections of modules. One way of designing and implementing the domain dependent part is to hardwire into the components the inter-component, hence the intra-agent communication. However, this has the disadvantage of creating inflexible architectures, difficult to modify.

This paper describes an open agent architecture and a platform for constructing and managing agents, that take into account the above issues. Agents consist of a collection of components, which can be integrated using a generic agent architecture. The components are precompiled code segments which are stored as byte code in a *code server.* The components making up the initial configuration of an agent are retrieved from the code server when the agent is launched. They can also be retrieved if a new functionality is to be sunsequently added to the agent, or if an old functionality is to be updated. In addition to the above two servers we have an extra manangement platform. This can be sent a request to launch a new agent on a particular machine.

When building components the April/April++ (McCabe & Clark 1994), (Clark et al. 1996), (Clark & Skarmeas 1997). language is normally used. However, other languages can be used, for example, Java or C. This agent toolkit has been used during the development of two agent based applications (Skarmeas & Clark 1996),(Boman et al. 1998).

## ALFA: The agent architecture

An agent comprises a set of interrelated components whose functionality contributes to the overall agent behaviour. Some of the components are common to all the agents and some are domain dependent, different for each agent. All the components run as separate April processes. This architecture, which is more fully described in (Skarmeas 1997), is depicted in figure 1.

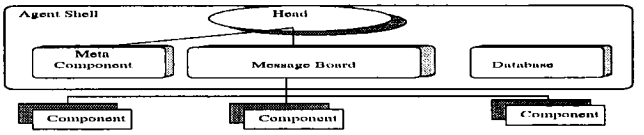


Figure 1: The agent architecture

The **agent head** deals with incoming messages from other agents. It is also a security wall to the outside

world. Incoming messages arrive first at it, and are then put on the the message board in order to be forwarded to the appropriate agent components, based on their content. Outside agents do not have direct access to the message board, which is the internal agent backbone.

The behaviour that the agent exhibits is implemented by a number of behavioural (domain dependent) components. The **behavioural components** will generally differ from agent to agent. Ideally, these behavioural components are changeable over the lifetime of the agent. This allows the agent to be reconfigured and gives it the ability to adapt to new requirements that its environment imposes. A behavioural component can be composite, indeed it can itself be another agent. So agents can have a recursive structure.

The **knowledge base** component keeps information shared by all the other components. It is the global memory for all the components of the agent (each of which may also have private memory) and can be used to store information such as the beliefs, intentions and plans of the agent as well as meta-level information about other agents and the capabilities of the behavioural components. This knowledge base can be accessed and updated by all the other agent components.

The agent components interact with each other in two ways. They store and retrieve information from the shared knowledge base. They also interact via messages. The message interaction is supported by the active **message board** (Skarmeas ). Any agent component can place messages intended for one or more other components on the message board. Components can join and withdraw without disrupting the functionality of the rest of the agent. When a new component is added to the agent it registers itself with the message board using a symbolic name and then sends advertisement messages to the board giving it a set of active message patterns. Because of the advertisements the component may be sent messages in the futute. The direction of messages is either based on the symbolic name of the component or it is content based using the active patterns. Once the message has been forwarded the receiver can reply directly to the component that placed the message on the board.

## Code server

An agent can be constructed in the spirit of a component oriented approach by implementing its domain dependent components independently and dynamically attaching them to its message board. The components implement functionalities that are recurring to several agents. Therefore, a mechanism is required, where we can store the code that implements such components. This code could be retrieved later and executed.

In our framework, components are represented as April closures that are stored in a specially implemented process called the *code server*. This process maintains a database table where the closures are stored. Clients can store new component to the code server and can also contact it in order to retrieve the code they need.

## The agent library

The agent is constructed by attaching a set of predefined components to its message board (which are stored in the code server). An agent description basically consists of the list of agent components. A new agent can be launched by getting hold of such a description and feeding it to a specially defined macro call, `launch_agent`, which will create a new agent. It will automatically launch the domain independent components of the agent (head etc.) and the domain dependent agent components by going through the list of component descriptions. For each component the code server will be contacted.

We have developed a server and we call it *agent library*. It is an April server which maintains a table of agent descriptions. It can be dynamically added new description, deleted old ones, and provide descriptions based on some symbolic name.

## The management platform

All the server and libraries described up to now offer a collection of tools that when used in combination can provide quite a powerful and open platform for building agents. The glue for putting all these together is an agent managemnet platform that is placed on top make use of all these servers and offer a high level, easy to use interface to external clients to create and manage agents. An external client, instead of having to know all the details of how to retrieve agent descriptions, fetch the component code and launch the agent can simply send the message: `(launch_agent,telecom_agent)` to the Management Platform. This will take care of all the steps for constructing and launching the agent.

## References


Boman et al. 1998. Energy Saving and Added Customer Value in Intelligent Buildings. *PAAM'98.*

Buschmann et al. 1996. *A System of Patterns: Pattern Oriented Software Architecture.* Wiley and Sons.

Clark, K. L., and Skarmeas, N. 1997. A Harness Language for Coooperative Information Systems. In Mike Papazoglou., ed., *Cooperative Information Systems.* Academic Press.

Clark et al. 1996. Agents as Clonable Objects with Knowledge Base State. *ICMAS'96.*

McCabe, F., and Clark, K. L. 1994. April – Agent PRocess Interaction Language. *Intelligent Agents.*

Meijler, T. D., and Nierstrasz, O. 1998. Beyond Objects: Components. In Papazoglou, M., ed., *Cooperative Information Systems.* Academinc Press.

Skarmeas, N. April++, ALFA documentation. http://users.otenet.gr/~ nik.

Skarmeas, N., and Clark, K. L. 1996. Intelligent Agents for Telecoms Applications. *IATA'96.*

Skarmeas, N. 1997. *Agents as Objects with Knowledge Based State.* Ph.D. Dissertation, Imperial College, Dept. of Computing, London.