SERVICE-WARE FRAMEWORK FOR DEVELOPING 3G MOBILE SERVICES [*]

Ralf Sesseler, Sahin Albayrak
DAI-Lab, TU Berlin, Sekr. FR 6-7, Franklinstr. 28/29, D-10587 Berlin, Germany
sesseler@cs.tu-berlin.de, sahin@dai-lab.de

Abstract. JIAC IV is a generic agent toolkit which is intended as a service-ware framework for the realisation of applications in the areas of telecommunications, mobile services, and electronic commerce. We motivate the need for open, scalable, and flexible systems and hence the advantages of agent technology in these areas. Then, we describe the basic architectural principles of CASA, which builds the structural and functional base of agents in JIAC IV. The component framework of CASA makes agents highly scalable and allows reconfiguration at run-time. The control architecture of CASA combines reactive, deliberative, and interactive capabilities to control agent behaviour in a flexible manner.

1. Introduction

The future of the telecommunications world will be dominated by increasing customer demands for new services and the integration of electronic and mobile commerce. One of the favourite slogans today is “information anywhere, anytime” claiming the ubiquitous availability of information and services via electronic networks like the Internet. On the other hand, deregulation leads to open markets and increasing competition. Therefore, the telecommunications companies are evolving from simple network providers to suppliers of service infrastructures supporting flexible and dynamic provisioning and combination of services.

We distinguish basic telecommunications applications from telematics services. By the notion of a telecommunications application, we generally denote any kind of information system that is necessary to run, to manage, and to administrate computer networks. On the contrary, the notion of a telematics service emphasises the aspect of trading between vendor and customer based on such networks. Hence, the telecommunications applications have to build the foundation for telematics services.

From this distinction, three main roles emerge in the area of telecommunications, each of them having specific requirements of their own. First, the suppliers of the telematics services need to develop and introduce new services rapidly and maintain existing services on a robust and secure service environment. The customers of these services demand convenient, personalised access and mobility support. Both benefit from the possibility to combine

[*] This work was funded by T-Nova Deutsche Telekom Innovationsgesellschaft mbH.

services on demand, leading from basic to high value services. Finally, the providers of the telecommunications infrastructure have to deal efficiently with increasingly complex and open networks with heterogeneous structures and technologies. Also, they have to take into account the requirements of the other two roles, because they provide the basic layer.

Agent technology (Wooldridge & Jennings 1995, Jennings & Wooldridge 1998) seems to be a promising way to realise such telecommunications infrastructures because of its inherent openness and distribution as well as because of the flexible and interactive capabilities of agents. The agent toolkit JIAC IV (Java Intelligent Agent Component-ware, Version IV) has been developed as a means to build and deploy telecommunications applications and telematics services as multi-agent systems to benefit from these advantages. The special requirements of 3G mobile services like offline capabilities and device-independent access are met by concepts like agent mobility and the multi-access point.

In the following, we provide a short overview of the JIAC IV agent toolkit as a whole. Then we concentrate on CASA (Component Architecture for Service Agents), which provides the structure and control mechanisms of single agents in JIAC IV. We describe how the component framework enforces openness and scalability by modularisation and how this modularisation facilitates the reuse of components at design-time as well as the reconfiguration of agents at run-time. The needed autonomy and flexibility of agents is ensured by a control scheme, which integrates reactive, deliberative, and interactive behaviour. Finally, we place our work in the context of related work and draw some conclusions.

2. Overview of JIAC IV

JIAC IV Agent Toolkit					
Architecture (CASA)		Development		Run-Time	
component architecture	*control architecture*	*methodology*	*tools*	*infrastructure*	*administration*
components roles messages	knowledge languages reactive deliberative interactive	analysis design implementation evaluation	configuration language compilers debugger	market places migration white pages yellow pages security	configuration fault management end user access

Figure 1. The JIAC IV Agent Toolkit

JIAC IV is intended as a comprehensive service-ware framework for developing and deploying agent systems covering design methodology and tools, agent languages and architecture, a FIPA compliant infrastructure, management and security functionality, and a generic scheme for user access. For a more comprehensive overview, we refer to (Fricke et al. 2001).

The development process is guided by an agent-oriented software engineering model, which is tailored to the specifics of JIAC IV. It comprises tools for agent specification including compilers for the different agent languages and tools to analyse and debug the running system.

On the single-agent level, CASA provides a scalable component framework and a flexible, knowledge-based behaviour control scheme, which we will describe further in the following sections.

The agent infrastructure has to facilitate dynamic interactions between agents. Based on the specifications of FIPA (FIPA 2000), it comprises the communication infrastructure as well as

services to administer the agents of a society (Agent Management Service, AMS) and the services they supply (Directory Facilitator, DF).

Commercial applications have special requirements on the reliability and trustiness of the system. Therefore, the JIAC IV toolkit provides several management and security functionalities, which can be easily integrated and adapted as needed thanks to the overall scalability. Management services include configuration, fault management, and logging of system processes, which may be used for analysis as well as for accounting. Security issues are addressed by authorisation, authentification, and privacy mechanisms.

To provide a convenient way for the user to interact with the agent system, JIAC IV contains a generic scheme to translate agent functionality into human accessible services. Thereby, different kinds of user devices and interface formats are supported.

3. Component Framework

Agents have to be adaptable to different purposes, tasks, and domains not only by varying knowledge but also by specific capabilities to process this knowledge accordingly. Therefore, CASA agents have a modular internal structure consisting of an open set of components that can be adjusted to different requirements at design-time as well as at run-time. Since components are reusable, new agents can be created using existing components for generic functionalities, which reduces development effort to application-specific implementations.

CASA provides a framework to manage the components of an agent and their interactions. Components are integrated into an agent by a common interface to the core agent, which realises the internal infrastructure (Figure 2). By this interface, the core agent controls the component and its configuration and gives it access to the message passing mechanism for component interactions. Among one another, components are identified by the roles they take within the agent describing their interactive capabilities to abstract from different implementations of the same functionality.

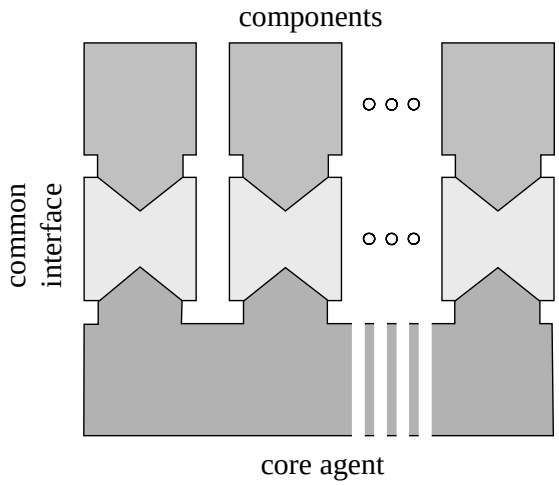

Figure 2. Component Framework

3.1 Component Interactions

Components interact by passing messages relative to their roles. Received messages are stored in a buffer to allow asynchronous processing. Thus, interactions are decoupled both

from time and from the component realising a role, which is needed to allow changes of the component set on run-time without affecting the functioning of the agent.

3.1.1 Messages

A message $m = [r_s, r_r, m_r, t, c]$ consists of the addresses (roles) of the sender r_s and the receiver r_r, a message type t, and the informational content c. The message type determines the type of the content and the intended kind of processing. In addition, a message can include another message m_r it refers to.

There are three general schemes of component interaction:

- *Inform*: In the simplest case, the sender passes some information to the receiver with a single message.
- *Order*: If a message expresses an order for some action, the recipient first replies whether it accepts the order. For an accepted order, it finally reports the result when the action is finished. The initiator of the order can send a message to retract it, which the receiver may either accept or reject.
- *Register*: This is an order to be informed. The sender registers itself to be informed by the receiver about a certain kind of events. The action consists of messages that inform about registered events until the order is retracted.

3.1.2 Roles

The interacting components know only the roles of their counterparts. Thus, dependencies do only exist between roles, but not between specific components. The roles serve as an interface description for the interactions of components and for their integration into the agent. A single component can have several roles, but each role must be unique to an agent to identify exactly one component.

Each role belongs to one or more groups of roles. Such a group subsumes roles with a common functionality. A group has the same kind of interface description as a role, which their roles inherit[1]. On top of the hierarchy is a most common role subsuming all groups.

The specification of a role covers two parts. First, it declares properties, by which a component implementing the role can be configured and its state retrieved. A property has a name, a type, a range of possible values, and a default value.

For interactions, a role declares the message types it can process and which roles and groups of roles have permission to send them. Thus, control structures consisting of several dependent roles can be defined. The message types, for which a component needs recipients, are not part of the role specification because they can differ for diverse implementations of a role. Instead, they belong to the documentation of the component to enable the creator of an agent to insure a set of components, in which all components have their demands for interactions fulfilled.

3.2 Components of the Core Agent

The core agent itself consists of three components. The Agent Kernel manages the components and the agent as a whole. The delivery of messages between components is done

---

[1] The inheritance relations are similar to object-orientation. Multiple inheritance means to merge the specifications, which also can be overwritten by the derived role.

by the Message Server. The Control Cycle organises all processing of the agent including that of messages. These components have direct access to all other components via the common interface (Figure 2).

3.2.1 Agent Kernel

The main task of the Agent Kernel is to manage the component structure of an agent. Also, it represents the agent as a whole and its properties and life-cycle state.

The common interface of the components allows the Agent Kernel to configure them by changing their properties. Properties can be declared for roles as well as for components. The properties of the agent are realised as properties of the Agent Kernel. A special property is the life-cycle state that determines its status of activity. All components have the same life-cycle state as the agent, except while they are added or removed or when they are defective.

Via the Agent Kernel, components of an agent can be added, removed, and exchanged. The Agent Kernel only allows adding a component, if no role of it is already occupied by an existing component. To add a component, its interface is connected to the components of the core agent for mutual access until it is removed again. Exchanging a component means adding a component while removing all previously existing components with corresponding roles.

The Agent Kernel creates an agent out of a specification containing a list of components and their initial properties. This is done by creating instances of the components, configuring them by the properties, and adding them to the agent.

3.2.2 Message Server

The Message Server provides the infrastructure for component interaction. It manages an address list that associates roles and existing components of the agent (Figure 3).

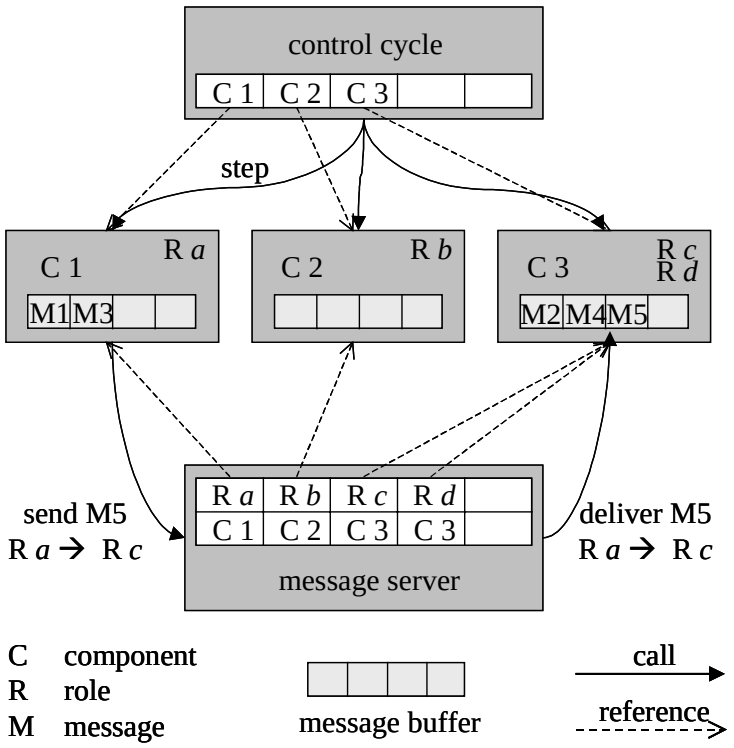

Figure 3. Interaction by Message Passing and Execution

To send a message, a component creates it and passes it to the Message Server. The Message Server validates the message by verifying that the sending component takes the role declared as sender and that this role has the permission to send messages of the given type to the receiver as specified in the role definition for the receiver address.

If the message is not valid or no component exists in the address list for the receiving role, the sender is informed accordingly. Otherwise, the Message Server delivers the message by adding it to the end of the queue of the receiver, which is part of the common interface of the components.

In the example of Figure 3, component C1 sends the message M5 to component C3 addressed as a message from role R*a* to role R*c*.

3.2.3 Control Cycle

The Control Cycle provides and manages the processing resources for the components. For controlled interruptions, all processes of the components have to work step by step. Thus, components are ensured to be in a stable state when they are removed, if they could finish the running step.

Components declare by the common interface if they have steps to execute. Then, the Control Cycle assigns processing resources as available. Steps can be executed in a sequential, parallel, or mixed mode, but only one step per component is executed at one time.

For message processing, the Control Cycle removes the first message from the buffer queue and passes it to the component for processing (Figure 3).

3.3 Reconfiguration at Run-Time

A main design motivation for the component framework is the reconfiguration of agents at run-time. Especially for agents that provide services, tasks have to be changed, updated, or adapted without the agent or some of its services being not available in the meantime. Thereby, not only the knowledge like facts and operators, but also the component functionality may need to be revised.

Changes of knowledge are a special case of component reconfiguration of the components containing that type of knowledge. The configuration of any component at run-time can be done via the Agent Kernel using tools and components for configuration by changing the declared properties of roles and components.

To change the functional part of an agent, components are added, removed, or exchanged. This is also done by the Agent Kernel. Like creating, adding a component simply means to configure it and to connect it to the core agent. On removal, the component gains no more new processing resources from the Control Cycle, but it can finish the current step. Then it is disconnected from the components of the core agent. Components can register at the Agent Kernel to be informed about changes of presence of components for some or all roles.

Exchanging a component means to replace a role without affecting the current working of the agent, especially of components interacting with that role by messages. The new component replaces the old one directly at the Message Server, so that at every time during the exchange messages can be sent to that role. In addition, the messages addressed to the shared role waiting in the buffer of the old component are moved to that of the new one ensuring no message can get lost.

The exchange of a component takes place by simultaneously adding the new and removing the old component. During the exchange, neither component has access to processing resources. The new component takes over the configuration and run-time state of the old one given by their properties. Thus, it can continue the work of the former without interruption or loss of information.

Since any component can take several roles that have to be replaced at the same time and possibly by different components, component exchange is done for a set of new components. All existing components taking at least one of the roles of the new components are removed to ensure uniqueness of roles.

By the described mechanism, component exchange is completely transparent to the remaining components even with respect to ongoing interactions. All dependencies between components are restricted to their roles ensuring an open and scalable structure of the agent.

4. Control Architecture

The components of the core agent do not determine the mechanisms to control the behaviour of an agent, but they provide an open framework to design functional architectures. Using the CASA component framework, such a control structure is defined by a set of roles. The interdependencies between these roles are stated by the message types, a role accepts to receive from a selected set of other roles, or at a more abstract level as the functionality, a role has to provide to other roles.

Before proposing a default architecture for service agents in CASA, we present a basic scheme for reactive, deliberative, and interactive behaviour. This scheme uses a knowledge-based approach in the tradition of artificial intelligence (Russel & Norwig 1995) and the BDI-theory of agency (Cohen & Levesque 1990, Rao & Georgeff 1995) in a pragmatic way without claiming to provide a base for rationality or intelligence in a human sense. Instead, the formal representation of declarative knowledge serves mainly as a common ground for interactions between agents providing a flexible and open scheme to express meaningful contents in a standardised and expressive way. On the other hand, the formal representation of procedural knowledge enables more autonomous, flexible, and explicit control of the behaviour of an agent. Combining both aspects, the formal descriptions of services enable reliable and dynamic interactions between agents.

Thus, the term knowledge as used in the following stands for data expressed in a formalised way that allows automatic processing according to the intended meaning of the programmer or user. It also serves as an abstract level to describe control architectures as well as the behaviour of concrete agents.

4.1 Control Scheme

Figure 4 gives an overview of the control scheme for the default architecture of CASA. An agent represents assumptions about the state of its environment as factual knowledge. Its goals are states to reach and its intentions actions to take. Rules describe the reactive dispositions, while operators describe the options for deliberative actions and interactions. All knowledge is expressed and structured by terminologies and representation schemes contained in ontologies for different domains.

An agent updates its facts and goals permanently, in order to reflect the current environmental and motivational state, and reacts to new situations. To reach its goals, it selects actions by deriving intentions using appropriate operators. The intentions are coordinated and the

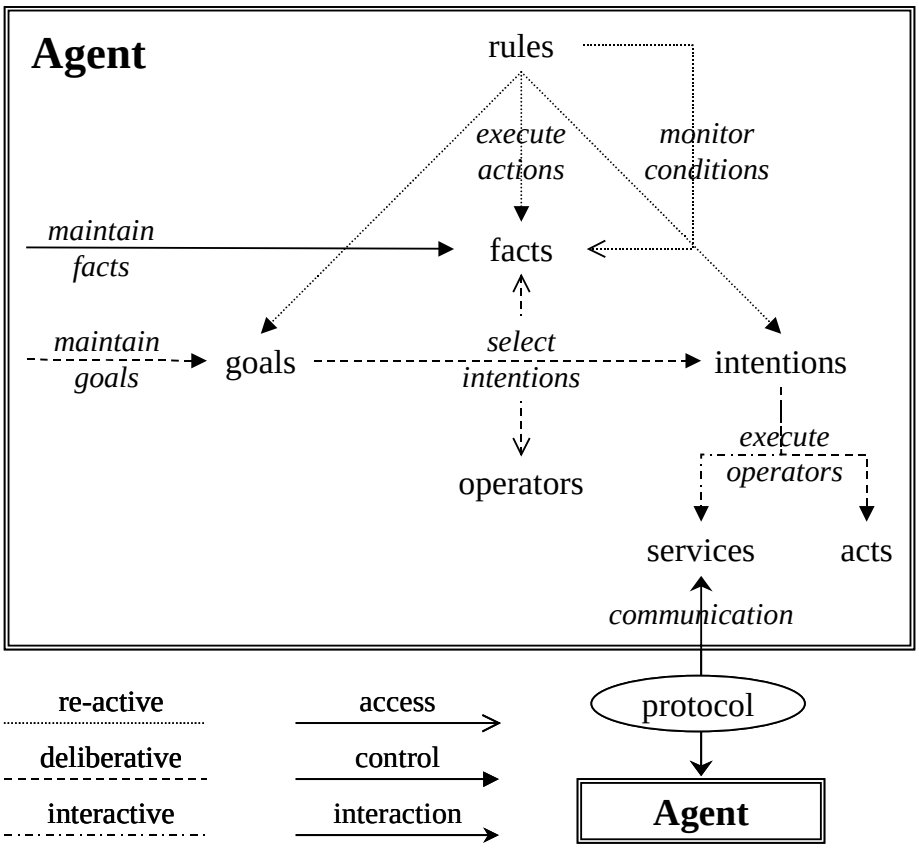

Figure 4. Behaviour Control Scheme

contained operators executed. To interact via services, agents communicate guided by protocols.

4.1.1 Types of Knowledge

Declarative knowledge describes the world as a set of states that are changing with time. Facts are assumptions about the current state of a relevant section of the world. A single fact ascribes a property to an object. The total factual knowledge of an agent is a conjunction of facts and the conclusions it can derive from them. There are no assumptions about the inference capabilities of an agent except that they have to be correct.

Goals guide the future behaviour of an agent. A state goal is a proposition about a partial world state to hold at present or in the near future. A conversation goal concerns to take a role in a protocol for a service interaction.

Rules build the reactive knowledge of an agent. They express dispositions for behaviour as reactions to changes in factual knowledge. The precondition of a rule describes a situation, in which a reaction is needed. The action part states the change of facts, goals, or intentions to perform in reaction.

The capabilities of an agent for deliberative acts are described by operators. Besides of an execution part, an operator consists of the conditions that must hold before or during the execution and the resulting effect after successfully finishing execution.

Several kinds of execution parts for operators are needed to flexibly control deliberative and interactive behaviour. Primitive acts each have a realisation of their own. A service describes potential interactions including usable protocols (see Section 4 for details). The realisation of

protocols is done by protocol operators that reflect the protocol structure for one role. They can contain communicative acts to send or receive speech acts. Protocol operators are used to reach conversation goals. Operators for compound actions like plans or scripts are used to describe courses of actions including protocols. Further operator types like abstract operators can extend the expressiveness of the behaviour control.

An intention is an instantiation of an operator intended for execution to reach a goal.

For having a uniform terminology to express propositions about states of the world, ontologies provide a vocabulary and representation schemes for specific domains. Objects are thereby grouped into categories declaring possible attributes and their types. Ontologies have to be public to serve as a shared terminology in communication.

4.1.2 Control Functions

The behaviour control mechanisms manipulate knowledge according to these types in order for the agent to fulfil its tasks.

To maintain a coherent and up-to-date representation of the current state of the world, the factual knowledge has to be updated constantly by adding new facts and removing inconsistencies. Also, new goals arise and old ones are reached or no longer intended. There are no restrictions to sources of new facts and goals.

For reactive behaviour, the facts are permanently monitored for occurrences of the situations stated by the rules. In such a case, the action of the corresponding rule is executed as a reaction.

Decisions have to be made about which goals to pursue and how to reach them by appropriate actions. The operators that are thereby selected result in new intentions. The intentions are coordinated to prevent conflicts and to benefit from redundancies. Finally, the intentions are executed.

If an intention concerns a service operator, its execution means to use that service by requesting it from a provider. If the request is accepted, both partners start the conversation by selecting and executing a protocol operator each for its role in the protocol for the service. Part of the protocol operators are the communicative acts to perform during the conversation.

4.2 Default Architecture

The CASA default architecture defines a set of component roles mapping the presented control scheme into a functional control architecture. This architecture is still open for different implementations of the defined roles, but it is reasonable to implement default components for application-independent roles. Thus, an agent designer can compose a new agent reusing generic components and only needs to develop new components for application-specific tasks. Also, he can use only particular parts of the default architecture or even another control structure, as long as its interactive behaviour conforms to the requirements for service interactions (see Section 4).

The default architecture consists of four parts: the core agent, the control unit, the knowledge base, and the periphery (Figure 5). The components of the core agent build the component as described in Section 2.2. Control unit and knowledge base together realise the control scheme. The knowledge base is a storage divided into roles for the different types of knowledge, while the control functions are assigned to the roles of the control unit. The periphery contains additional types of roles for auxiliary and application-specific tasks.

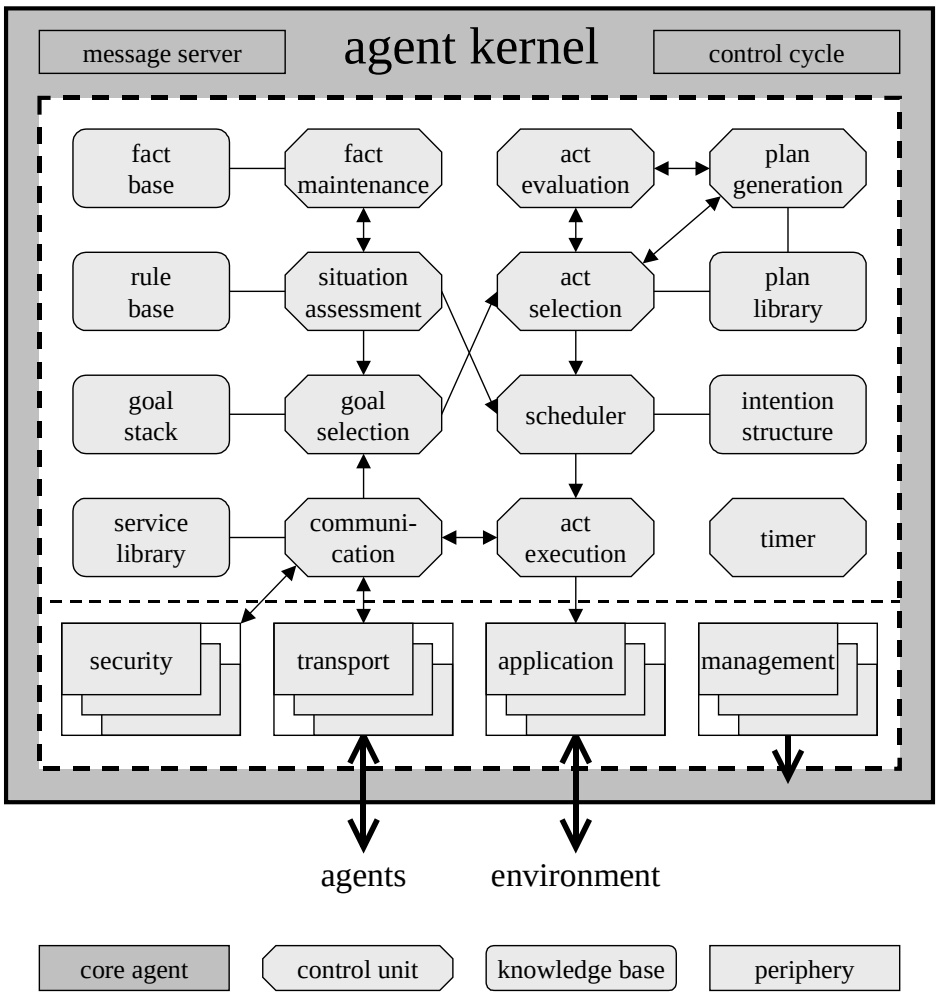

Figure 5. Roles of the Default Architecture

4.2.1 Knowledge Base

The six roles of the knowledge base are the fact base, the goal stack, the intention structure, the rule base, the plan library, and the service library. Each of them is responsible to store, maintain, and provide knowledge of the according type.

The fact base is a set F of facts containing the assumptions of an agent about the current state of the world. It has to be consistent at the ground level, i.e. no two facts contradict each other. The goal stack is an ordered list G of the current goals. Its order reflects the priority of the goals and hence the probability to be pursued next. The intention structure is a set I of intentions together with their interdependencies. The intentions are coordinated according to conflicts, redundancies, and execution order.

Rule base, plan library, and service library are mere containers for the procedural knowledge. The rule base is a set R of rules, the plan library a set O of operators, and the service library a set S of service operators. The plan library contains all operators the agent can use to reach its goals including protocol operators and service operators for service usage. The service operators in the service library describe the services the agent provides.

Together, the knowledge base KB = [F, G, I, R, O, S] constitutes all of the knowledge an agent has. All roles of the knowledge base provide the same kind of functionality by

messages for managing their contents by accessing and changing knowledge and monitoring changes of knowledge.

4.2.2 Control Unit

The control unit operates on the knowledge stored in the knowledge base. Its roles can be grouped by the following four tasks: representation, reaction, deliberation, and interaction.

The representational tasks concern the environment and as a special aspect time. The fact maintenance manages the factual knowledge to reflect the current state of the world. It has to keep the facts up-to-date and coherent, but thereby it depends on the information other components gain about the environment. The fact maintenance also realises the reasoning capabilities of the agent by handling not only isolated facts, but also complex propositions that express changes of the world or whose truth has to be evaluated relative to the current facts. Since time changes continuously, the current time is not handled simply as a frequently changing fact. Instead, the timer role serves as an internal clock offering notifications for absolute, relative, and periodical time events.

For reactions, the situation assessment monitors changes of factual knowledge for the situations described by the rules contained in the rule base. If such a situation occurs, the situation assessment changes the knowledge of the agent according to the action part of the rule resulting in a behaviour adapted to the new situation.

Deliberative behaviour emerges from six roles leading from goals to acts: goal selection, act selection, plan generation, act evaluation, scheduler, and act execution. The goal selection organises the motivational state of an agent. All roles can add new state goals, while new conversation goals arise exclusively from the communication. The goal selection updates, evaluates, and coordinates the goals in the goal stack and selects those to pursue next. These goals are passed to the act selection to find actions to reach them. Therefore, suitable operators have to be found and the best alternative has to be selected as a new intention. Since both tasks may depend on the application domain, they can be delegated to own roles for plan generation and act evaluation. These roles are separated from the act selection in favour of modularisation to allow different domain-specific implementations. The act selection thereby only realises the basic capabilities. The plan generation combines operators to plans in a goal driven manner. The act evaluation compares different acts to decide between alternatives.

The scheduler coordinates the intentions of an agent resolving conflicts and utilising redundancies. It determines the order of execution and selects the next intentions to execute, which are passed to the act execution. The act execution interprets the operators contained in the intentions. Depending on the operator type, it is either executed by the act execution itself or passed to the appropriate role. The results from executing operators are reported back.

Service operators are executed by the communication. This role organises the interactive behaviour of an agent. It initiates the usage of services and handles service requests. In addition, the communication creates speech acts to send and processes received speech acts. It is also responsible to ensure a secure communication according to the security requirements of a service.

4.2.3 Periphery

Since the structure of the periphery is not strictly determined by the CASA control scheme, it only defines groups of roles each with a common functionality: the application group, the transport group, the security group, and the management group.

Application-specific tasks of an agent are realised by the roles of the application group. This includes interactions with the environment like sensory input and behavioural output, but not communications. These roles can affect the agent behaviour by stating new facts and raising new goals. For deliberative acts, each primitive operator denotes an application role that implements its execution.

The communication role is just responsible for organising communication at a higher level, while the transport of speech acts between agents is left to the roles of the transportation group. Each role of this group enables communication via a single communication channel like TCP/IP, SSL, or IIOP. At run-time, it constitutes an address by which the agent can send and receive speech acts. Received speech acts are passed to the communication role, which also is the source for speech acts to send.

To address security issues in communications between agents, the communication role has to rely on the roles of the security group. Security requirements for services may include among other authentification of the communication partner by certificates, privacy of communication via encryption, authorisation for service usage, and trust relationships between agents (FIPA 1998, Thirunavukkarasu et al. 1995). Each security aspect can be covered by a role of its own. Which of these security aspects are supported depends on the component implementing the communication role that has to interacting with the corresponding security roles.

The roles of the management group control the agent and its components at run-time. Their instances gain direct access to the Agent Kernel and its functionality. There are two directions of management: introspection and manipulation. The introspection collects and analyses run-time information about the agent. The manipulation can modify properties and the component set. Management tasks include among other reconfiguration, fault detection and correction, and performance measurement and improvement.

4.3 Agent Interaction

The control scheme and the default architecture of CASA integrate the interactive capabilities of agents into the overall behaviour control. All interactions between agents are based on services, which are actions one agent performs in behalf of another. A service is described as an operator for planning from the view of the customer of the service. Thus, the customer can use the services of other agents to decide about its behaviour as any other action, while only the execution is delegated to another agent, which is called the provider of the service. This allows for flexible and dynamic selection and combination of services for an agent to pursue its goals.

Communication between agents is based on shared ontologies, an agent communication language, and protocols to ensure interoperability. Ontologies are shared by being public and thereby provide a common ground for the contents of communication. Speech acts are formulated using FIPA-ACL (FIPA 2000).

Since all interactions are according to a general scheme of service usage, CASA defines a generic meta-protocol, which handles the common aspects of service interactions. It is initiated by the customer when requesting the service. The protocol comprises three steps. First, a generic negotiation phase is used to establish the possibility for interaction. The

provider ends this step by accepting or refusing the initial request. In the second phase, the customer may select one of several providers by negotiating service-specific parameters using an embedded protocol. Then, another embedded protocol may be used for service provision. Finally, the provider ends the meta-protocol by communicating the result of the service usage. While the generic meta-protocol is an integral part of CASA, the embedded protocols for negotiation and service provision are open for specific communications.

5. Related Work

Only few agent systems attempt to provide complete toolkits covering all aspects of agent development and deployment. ZEUS (Azarmi & Thompson 2000) and AgentBuilder (Reticular Systems 1999) both emphasise the relevance of supporting the development process by a methodology and a set of tools for agent specification and system analysis. Also, each offers a sound control architecture for knowledge-based agent behaviour including appropriate specification languages. ZEUS utilises a traditional planning approach and integrates interactive capabilities. AgentBuilder uses a more reactive control structure based on Agent-0 (Shoham 1993), but supports only simple message passing for agent interaction. An infrastructure for multi-agent systems is provided by ZEUS via dedicated Utility Agents, but is missed in AgentBuilder. Both systems do not address security issues or a generic user access scheme.

Many agent architectures concentrate on the internal mechanisms for knowledge-based behaviour control. Well-known examples are PRS/dMARS (Georgeff & Ingrand 1989, d'Inverno et al. 1997) or InteRRaP (Jung & Fischer 1998). These systems are usually based on theories of action like BDI (Cohen & Levesque 1990) and are inspired by the tradition of artificial intelligence. To put the focus on the control architecture leads to powerful systems, but most of these systems lack to support the development process and to provide a reliable infrastructure.

Other systems mainly provide an agent infrastructure, but rely on only minimal agent models. Good examples are JATLite (JATLite) and the FIPA-compliant JADE (Bellifemine et al. 2000). These systems support only the multi-agent aspect, but do not integrate the single-agent view. Instead, agents are implemented by conventional programming techniques and languages like Java or by using a control architecture, which is not part of the system itself. Either way, the multi-agent functionality is not an integral part of the single-agent behaviour mechanisms and vice versa, which means that these will have to be adapted for each application anew.

6. Conclusion and Outlook

JIAC IV is a complete agent toolkit that is already capable of providing the infrastructure for real-world telecommunications and electronic commerce applications.

By its openness and scalability, the toolkit makes it easier to develop new applications and to adapt running applications according to dynamic changes of customer demands and by the integration of new services. This is achieved at the single-agent level by the component framework of CASA and at the multi-agent level by the agent infrastructure. Furthermore, application development and deployment is supported by a process model and a set of tools.

The control architecture allows to create agents that are flexible enough to fulfil the tasks delegated to them by the user reliable and autonomous. The interactive capabilities enable agent societies with dynamic selection and combination of services.

The end user of applications realised with JIAC IV is provided with a uniform and convenient way of accessing the services, which are realised by agents. The supply and integration of security and accounting functionalities are prerequisites for commercial applications and are needed for user acceptance.

Since agent systems tend to by very complex systems, there are still some improvements to make on the concepts and the implementation of JIAC IV. A critical point is the competence needed by agents to make decisions according to the intentions of the user it is acting for. The mechanisms of the control architecture of CASA are still generic. We aim to integrate aspects like personalisation and learning that allow to adapt the decision making to the demands and preferences of the individual user.

References

Azarmi, N., and N. Thompson, (2000), "ZEUS: A Toolkit for Building Multi-Agent Systems," *Proceedings of fifth annual Embracing Complexity Conference*, Paris.

Bellifemine, F., A. Poggi, and G. Rimassa, (2000), "Developing multi-agent systems with JADE," *Seventh International Workshop on Agent Theories, Architectures, and Languages* (ATAL-2000), Boston, MA.

Cohen, Philip R., and Hector J. Levesque, (1990), "Intention is Choice with Commitment," *Artificial Intelligence*, 42(3).

d'Inverno, Mark, David Kinny, Michael Luck, and Michael Wooldridge, (1997), "A Formal Specification of dMARS," *Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages* (ATAL'97).

FIPA, (1998), *FIPA '98 Specification*, www.fipa.org.

FIPA, (2000), *FIPA 2000 Specification*, www.fipa.org.

Fricke, Stefan, Karsten Bsufka, Jan Keiser, Torge Schmidt, Ralf Sesseler, and Sahin Albayrak, (2001), "Agent-based Telematic Services and Telecom Applications," *Communications of the ACM*, April 2001.

Georgeff, Michael P., and Francois F. Ingrand, (1989), "Decision-Making in an Embedded Reasoning System," *Proceedings of the Eleventh International Jopint Conference on Artificial Intelligence* (IJCAI-89).

JATLite, *JATLite*, java.stanford.edu.

Jennings, Nicholas R., and Michael J. Wooldridge (Eds), (1998), *Agent Technology: Foundations, Applications, and Markets*, Springer-Verlag.

Jung, Christoph G., and Klaus Fischer, (1998), "Methodological Comparison of Agent Models," *DFKI Research Report* RR-98-01

Rao, Anand S., and Michael P. Georgeff, (1995), "BDI Agents: From Theory to Practice," *Proceedings of the First International Conference on Multi-Agent Systems* (ICMAS-95), San Francisco, USA.

Reticular Systems, (1999), *Agent Builder - An Integrated Toolkit for Constructing Intelligent Software Agents*, www.agentbuilder.com

Russel, Stuart, and Peter Norwig, (1995), *Artificial Intelligence: A Modern Approach*, Prentice Hall.

Shoham, Yoav, (1993), "Agent-Oriented Programming," *Artificial Intelligence*, Vol 60, pages 51-92.

Thirunavukkarasu, Chelliah, Tim Finin, and James Mayfield, (1995), "Secret Agents – A Security Architecture for the KQML Agent Communication Language," *CIKM'95 Intelligent Information Agents Workshop*, Baltimore.

Wooldridge, Michael, and Nicholas R. Jennings, (1995), "Intelligent Agents: Theory and Practice, " *Knowledge Engineering Review*, October 1995.