Building Agents for Service Provisioning out of Components*

Ralf Sesseler

DAI-Lab, TU Berlin
Sekr. FR 6-7, Franklinstr. 28/29
D-10587 Berlin, Germany
+49 (0)30 314-21736

sesseler@cs.tu-berlin.de

ABSTRACT

The CASA architecture describes a platform for the provisioning of services by agents by supporting three levels of agent design. At the base level is a framework to build an agent out of reusable components. Using this framework, a predefined component structure realises control mechanisms for reactive, deliberative, and interactive behaviour. The agent communication used for interactions is guided by protocols and formal service descriptions. By these concepts, CASA provides an open, scalable agent architecture for service provisioning.

Keywords

agent architectures, component systems, agent-based service provisioning, multi-agent collaboration, communication protocols

1. INTRODUCTION

Computer networks like the Internet promise many advantages as electronic platforms for service provisioning. Service supply is very flexible and effective, while service access is independent from time and location of usage. The open, distributed, and heterogeneous character of the networks offers many new possibilities, but also raises new technological demands. Current platforms are often proprietary stand-alone systems that miss to utilise the whole potential of the networks and their dynamics, because they lack interoperability.

An alternative is given by agent technology [8, 12], which is concerned with flexible interactions in open, distributed systems. In the following, we describe CASA (Component Architecture for Service Agents), a multi-agent architecture that facilitates the design of interoperable service platforms. The aim of CASA is automated and flexible provisioning and usage of services in domains like electronic commerce and telematics by dynamic supply, selection, and combination of services. This is done by combining the multi-agent approach and a service concept that maps service provisioning and usage to the agent level.

The CASA architecture consists of three design levels: agent creation, single-agent behaviour, and multi-agent interactions.

For scalability, an agent is built by selecting an arbitrary set of reusable components as needed to fulfil its tasks. The component set can be changed even at run-time by adding, removing, or replacing components. The core agent manages the component set and supports decoupled interactions between components by directed message passing.

Even if it is not mandatory, we propose a default architecture of components to realise reactive, deliberative, and interactive behaviour. This architecture is divided into a knowledge-based kernel consisting of components to store and to process knowledge and a periphery with components that provide interfaces to the environment or additional functionality. The use of explicit knowledge representations allows thereby a more flexible behaviour control and to handle the formal descriptions of services.

Agents interact by services that specify actions an agent offers to perform on behalf of other agents. For service usage, agents communicate via speech acts according to protocols. An agent infrastructure supports dynamic changes of available services and agents by providing services to search for services and agents.

The following three sections each describe one of the aforementioned agent design levels: the component framework to create agents, the default architecture for agent behaviour control, and the service conception for multi-agent interactions.

2. 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

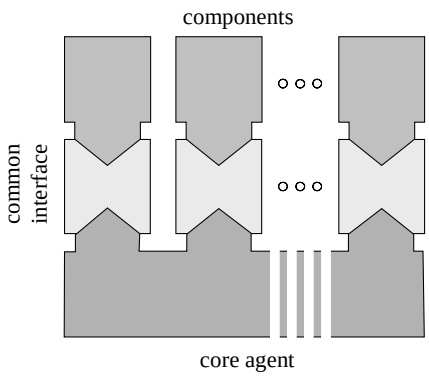

Figure 1. Component Framework

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

reusable, new agents can be created using existing components for generic functionalities, reducing 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 1). 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.

2.1 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.

2.1.1 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.

2.1.2 Roles

The interacting components know only the roles of their counterparts. Thus, dependencies can 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. If a property only provides run-time information, it is read-only.

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.

2.2 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 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 1).

2.2.1 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.

2.2.2 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 2).

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

[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.

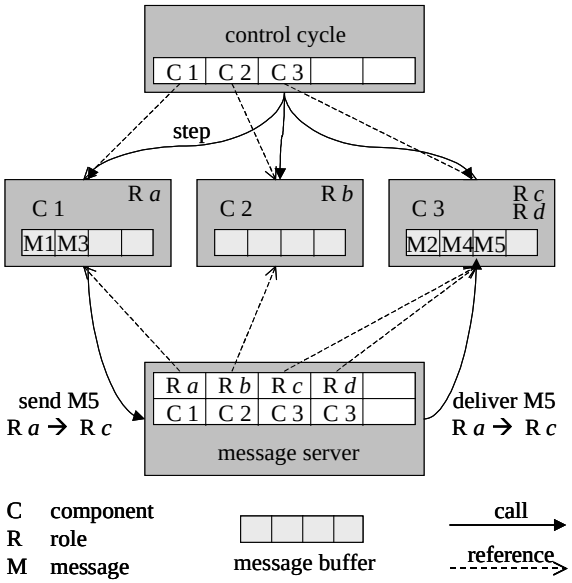

Figure 2. Interaction by Message Passing and Execution

to the end of the queue of the receiver, which is part of the common interface of the components.

In the example in Figure 2, component C1 sends the message M5 to component C3 addressed as a message from role R*a* to role R*c*.

2.2.3 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 2).

2.3 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.

3. BEHAVIOUR CONTROL

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 [10] and the BDI-theory of agency [2, 9] 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.

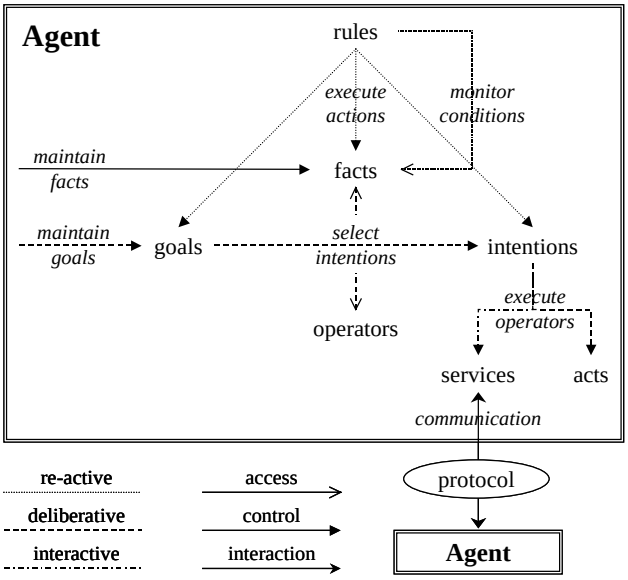

Figure 3. Behaviour Control Scheme

3.1 Control Scheme

Figure 3 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 contained operators executed. To interact via services, agents communicate guided by protocols.

3.1.1 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.

3.1.2 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.

3.2 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 4). The components of the core agent build the component framework

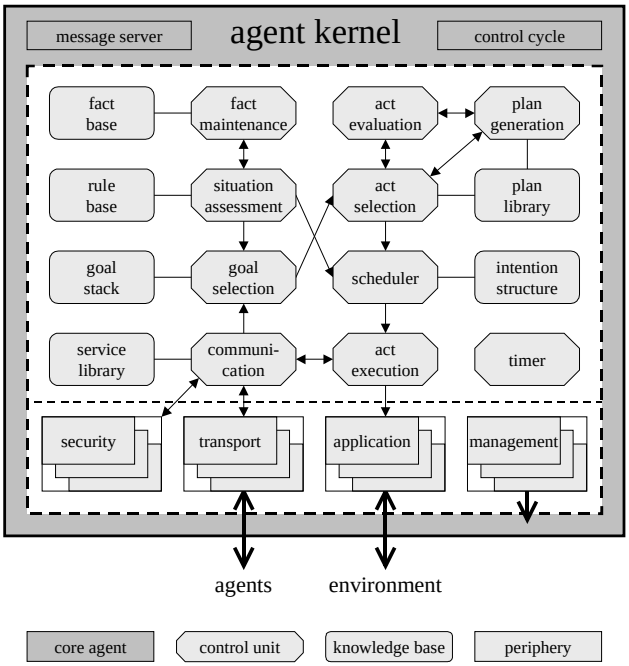

Figure 4. Roles of the Default Architecture

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.

3.2.1 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:

- *Knowledge access*: Knowledge can be retrieved and searched to determine the current knowledge.
- *Changes of knowledge*: Knowledge can be added and removed to adjust the current knowledge.
- *Monitoring of changes*: Roles can register to be informed about some or all changes of knowledge.

3.2.2 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.

3.2.3 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 [6, 11]. 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. INTERACTIONS BY SERVICES

A society of agents is characterised by the interactions taking place at a communicative level. To allow interactions between different agents, there is a need for interoperability, which is ensured in technical systems by standardisation. A standard prescribes a compatible design for conformant systems.

Most agent systems employ standards for interactions at different levels. Communication languages like KQML [3] and FIPA-ACL [4] provide a format to exchange messages between agents. The representational content of these messages is formulated by own languages with uniform syntax and semantics using terminologies expressed in shared ontologies [7]. Thus, communicating agents are enabled to interpret received messages as intended by the sender. In addition, protocols regulate sequences of communicative acts for specific purposes to reduce the space of possible conversations.

On the top of this, CASA adds the concept of services. A service provides a generic scheme to describe interactions between two agents, the provider and the customer of a service. A predefined protocol frames the service usage, allowing embedded service-specific protocols. Furthermore, the default architecture supports the service conception by operators for services and protocols and by the role of the communication.

Besides of the agents, a multi-agent system consists of an infrastructure managing the system as a whole. This infrastructure also facilitates interactions between agents by supplying services to search for other agents and their services and communication addresses.

4.1 Services

The main idea of a service in CASA is that of an act one agent performs for another one. Therefore, it is described by an operator stating conditions and effects of this act from the point of view of the customer. Thus, the customer can handle a service as any other act to control its behaviour and only the execution is delegated to the provider via communication.

On the other hand, formalising all interactions by services provides an explicit framework for generic requirements and parameters of interactions like accounting, billing, payment, and security. Services can thereby support competitive as well as cooperative or mixed societies.

The execution part of a service operator contains the service description. It consists of a unique identifier and generic parameters common to all services. These parameters include the content language and ontologies used to express the conditions and effects of the operator. It also comprises a list of protocols that can be used as embedded protocol for service usage or negotiation respectively.

To specify interactions by service operators has many advantages. Conditions and effects of using a service are described explicitly in a familiar manner. A dynamic selection of services can be done in the same way as for other actions, and the combination of services means nothing else than generating a plan with several service operators. Thus, service operators allow for flexible and automated interactions between agents.

Some particular aspects have to be considered when using service operators. For each service, several providers and protocols may be available, which have to be selected at planning or execution time. Especially for service operators inside of plans or scripts, commitments from the providers to supply it to the customer should be gathered in advance to ensure all steps of the plan or script to be executable. Also, the evaluation of a service intention may depend on the selected provider and protocol as well as on other service parameters, which might even be negotiable. Thus, components that select, evaluate, or execute service operators have to cope with these different possibilities of performing them.

4.2 Protocols

The communication language of CASA is compliant to the specification for FIPA-ACL [4]. In CASA, each speech act (except error handling) belongs to a conversation and is part of a protocol. Each service usage is a single conversation with a global unique identifier. The protocol is either the generic meta-protocol common to all services or a service-specific protocol, be it for negotiation or the service supply itself. Since every interaction is part of a service usage, all protocols have exactly two roles, the customer and the provider. The customer is always just one agent, while the role of the provider may be taken by several agents for a dynamic selection of the provider by negotiation. A service

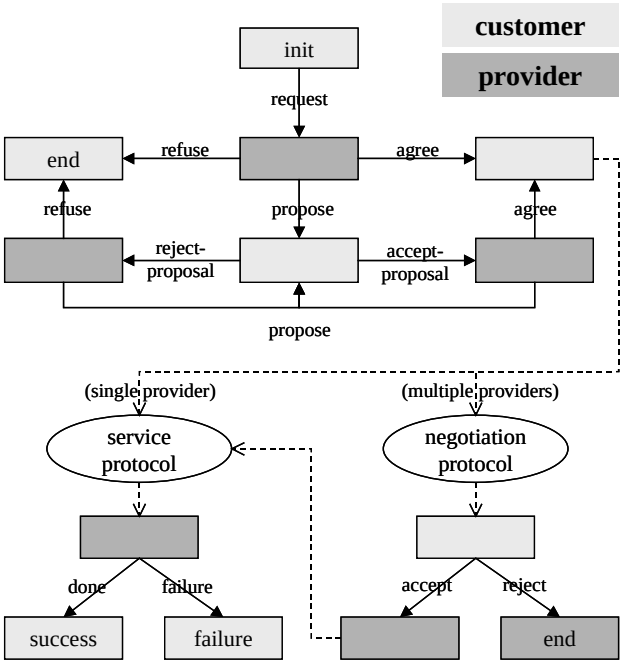

Figure 5. Meta-Protocol for Service Interactions

allows multiple providers, if it declares at least one protocol for negotiation.

4.2.1 The Meta-Protocol

The meta-protocol (Figure 5) frames every service usage as a generic ordering scheme for services. It is always initiated by the customer when executing a service operator. The initiating speech act is a request containing the unique identifier for the service and values for generic and specific parameters including the embedded protocols to be used. This is the only kind of speech act an agent has to process outside of an ongoing conversation, because it establishes a new conversation. An optional negotiation phase follows, where a provider can propose other values for the generic parameters, which the customer can accept or reject. If a proposal is rejected, the provider can make another proposal for the same parameter, otherwise only for other parameters. Finally, the provider has to agree or refuse the request. An agreement is only possible, if for every negotiated parameter a proposed value was accepted. A refused request finishes the conversation.

On agreement, the service-specific part begins. If multiple providers agreed, the negotiation protocol is performed to select the best provider by negotiating service-specific parameters. When it ends, the customer has to accept one provider and reject all other. Then, for the accepted provider as well as for a single provider, the service protocol starts. When the service protocol is finished, the provider sends the result of the service usage, which may be a "done"[2] or a "failure" speech act. In the first case, the execution of the service operator ends with a success, otherwise with a failure.

[2] "done" is a communicative act type of its own with a similar meaning to inform(done(action)) in FIPA-ACL to avoid the need for a content language expressive enough for modal logic.

In addition to the scheme of Figure 5, the meta-protocol allows communicative act types for error handling which may be used at any time by both roles. The first type is "cancel", which cancels the conversation. The second type is "not-understood", which refers to a speech act that can not be processed by the receiver, be it part of a conversation or not. Also, there might be a security protocol framing the meta-protocol handling the security requirements for a conversation.

The meta-protocol is handled by the component implementing the communication role. It provides a uniform scheme for interactions by ordering and negotiating services. Specific communications for a service are covered by embedded protocols.

4.2.2 Embedded Protocols

Inside of the generic meta-protocol, there are two types of embedded protocols, one for negotiation and one for service supply. For each service usage, these protocols may be chosen by negotiation from the ones prescribed by the service operator. Since the protocols are generic parameters, this takes place at the negotiation phase of the meta-protocol. Thereby, agents should only propose or accept protocols for which they have a corresponding protocol operator for their role.

To start an embedded protocol, each agent sets up a new conversation goal to take its role in the protocol, which leads to the execution of an appropriate protocol operator. If the meta-protocol already covers all communications needed in a conversation, the embedded protocol specifies no communications. This is still a kind of protocol in CASA, because it is also realised by an protocol operator determining the decision or result. In this case, only the role sending the terminating speech act of the protocol, which is already part of the meta-protocol, needs to execute an protocol operator.

A protocol is a scheme for conversations determining which role may perform which communicative acts depending on the previous communications. A protocol operator has to reflect the structure of a protocol for one role by executing appropriate communicative operators to send or await speech acts accordingly. In addition, it takes the service-parameters as input and has a result as output.

The description of a protocol consists of a unique name and the type declaring it as a negotiation or service protocol and as containing communicative acts or not. Also, it specifies a content language and the ontologies used in communicative acts. Finally, it contains the service-specific parameters to be processed by a protocol operator. A protocol operator description adds to a normal operator the protocol description and the role.

4.3 Infrastructure

Agents have to cope with open societies, where agents join and leave and the supply of services changes dynamically. Hence, there is a need for an infrastructure, allowing these changes to take place and providing information about them to the agents.

In CASA, this infrastructure is realised by so-called marketplaces, which provide a platform for the agents to reside on. Each marketplace is constituted by a Manager Agent administering the agents on a marketplace and offering infrastructure services to them. The infrastructure services are not restricted to one market place, but are interconnected by mutual acquainted Manager Agents. In addition, agents can migrate between marketplaces by using an according service of the Manager Agent.

The infrastructure services of CASA are compliant to the FIPA agent management specification [5] consisting of Agent Communication Channel (ACC), Agent Management System (AMS), and Directory Facilitator (DF):

- The ACC is a default communication means between agent platforms with a standardised interface. It supports sending messages to other agents addressed only by a global unique identifier (GUID).
- The AMS manages an agent platform allowing to add and remove agents and to control their lifecycle state. It assigns a GUID to agents it creates and maps GUIDs to local communication addresses.
- The DF contains information about available services. Agents announce the services they offer by registering them at the DF.

By these services, an agent can dynamically access the resources of an agent society. It can query the DF for available services fulfilling its needs, e.g. if it does not have own capabilities to reach a goal. Also, the DF offers information about agents providing a particular service. For communication, an agent can use the ACC or query the AMS for communication addresses of a service provider. Thus, an agent can access up-to-date information concerning the agents and services of the agent society it is a part of.

5. CONCLUSIONS

The CASA architecture provides a pragmatic approach for flexible, automated interactions in open, dynamic environments. Its component framework facilitates the design and creation of a variety of different agents and agent types. Also, it makes agents scalable and allows dynamic reconfigurations at run-time. The default architecture supports a flexible control of agent behaviour by combining reactive, deliberative, and interactive capabilities. The service concept frames interactions by an open, but reliable scheme. Together with a FIPA-compliant communication language, shared ontologies, and the marketplace infrastructure, it ensures a high level of interoperability within a multi-agent system including dynamic usage and combination of services.

A suitable application domain of the CASA architecture is the realisation of platforms for service provisioning in electronic networks. Services are thereby provided to the human user by particular agents that access the capabilities of the agent society to fulfil their tasks. The flexibility and interoperability of the multi-agent-system utilises the openness and dynamics of the network in favour of a likewise open and dynamic service environment. The platform itself is open and scalable at agent level as well as at society level to cope with the increasing dynamics of service supply and customer demands. Since the agents interact by services, requirements of user-services like pricing and security can be mapped easily to the agent-level.

Based on the CASA architecture, the agent system JIAC IV (Java Intelligent Agent Component-ware, Version IV) was implemented. The implementation consists of the component framework, components realising the roles of the default architecture, a language for knowledge representation, and the marketplace infrastructure. In addition, there are services, components, agents, and tools for advanced management functionalities and for agent security [1]. The implementation and evaluation of agents and agent systems is supported by a set of tools.

ACKNOWLEDGEMENTS

The JIAC IV system was developed and implemented by three interrelated projects AARFTA, MIATA, and SIATA at DAI-Lab, funded by T-Nova. Thanks to all project members, particularly to Siegfried Ballmann, and to the leader of the DAI-Lab Sahin Albayrak.

REFERENCES

[1] Karsten Bsufka, Stefan Holst, Torge Schmidt. Realization of an Agent-Based Certificate Authority and Key Distribution Center. in: Sahin Albayrak (Ed.), Intelligent Agents for Telecommunications, LNAI 1699, Springer, 1999.

[2] Philip R. Cohen, Hector J. Levesque. Intention is Choice with Commitment. Artificial Intelligence, 42(3), 1990.

[3] Tim Finin, Yannis Labrou, James Mayfield. KQML as an Agent Communication Language. in: Jeff Bradshaw (Ed.), Software Agents, MIT-Press, Cambridge, 1995

[4] FIPA 97 Specification, Version 2.0, Part 2: Agent Communication Language. ww.fipa.org, 1997.

[5] FIPA 98 Specification, Part 1: Agent Management. ww.fipa.org, 1998.

[6] FIPA 98 Specification, Part 10, Version 1.0: Agent Security Management. www.fipa.org, 1998.

[7] Thomas R. Gruber. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. Technical Report KSL 93-04, Knowledge Systems Laboratory, Stanford University, 1993.

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

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

[10] Stuart Russel, Peter Norwig. Artificial Intelligence: A Modern Approach. Prentice Hall, 1995.

[11] Chelliah Thirunavukkarasu, Tim Finin, James Mayfield. Secret Agents – A Security Architecture for the KQML Agent Communication Language. CIKM'95 Intelligent Information Agents Workshop, Baltimore, 1995.

[12] Michael Wooldridge, Nicholas R. Jennings. Intelligent Agents: Theory and Practice. Knowledge Engineering Review, October 1995.