Agent-based Telematic Services and Telecommunication Applications[1]

*Stefan Fricke, Karsten Bsufka, Jan Keiser,
Torge Schmidt, Ralf Sesseler, Sahin Albayrak*

Abstract

The future telecommunications world will induce significant changes to the involved actors as well as to the quality and quantity of offered services. This article points out the determining factors, trends, and demands for infrastructure functionality of future service architecures and then motivates why an agent-based approach is adequate as underlying technology. We present the toolkit JIAC as a potential candidate for realizing such future services and applications as agent-based systems. It provides a scalable agent architecture, which integrates telecommunication-specific management functionality, cooperative service usage, and strong security mechanisms. A runtime environment adds further service infrastructure, authentication, and authorization. The toolkit is completed by a set of tools for rapid development and deployment of services as well as for managing purposes.

Introduction

The telecommunications market is expanding rapidly and players in that market are facing increasingly stiff competition. The key to commercial success in the telecommunications market will be the provisioning of adequate services, the focus shifting from a purely technological one to one of convenience and usefulness. An important prerequisite is the effective management of basic telecommunications infrastructure supporting the rapid deployment of new services.

These future services will determine the market shares to be gained. Not only must their time to market be reduced, but also other requirements need to be fulfilled, e.g. dynamic service development and configuration. Due to a demand for permanent availability the motto to be followed is “information anywhere, anytime”. Service maintenance must not interfere with continuous service usage; future services must allow for personalization, meet security demands, and provide management functionality. Furthermore, asynchronous service usage has to be supported as well as demand-driven service combination and integration. Not the least, services must allow for access independent of specific technologies and terminal equipment.

Next to the technical requirements, new business models must be developed reflecting the fact that various actors in new roles, e.g. content provider or application service provider, will need to co-operate and coordinate in order to provide these future services. Companies will provide integrated solutions with own and third-party services being bundled on their platforms. These platforms will realize required infrastructure functionality and enable various means of access by facing influencing factors and developments of the future

[1] This work was funded by T-Nova Deutsche Telekom Innovationsgesellschaft GmbH.

telecommunications market such as consumer devices (mobile phones, screen phones, PDAs), networks (e.g. GPRS, UMTS), languages and software technologies (Java, Jini), consumer demands and trends like convenience of use, mobility, and ubiquitous computing. Each role participating in the future telecommunications world will have specific requirements to such service platforms. These demands differ in the extent of infrastructure being needed for service usage and provisioning, according to different necessities regarding aspects such as security, personalization, asynchronous usage, mobility, device-independency, and supporting tools.

The issues outlined above open up new perspectives for services and applications, especially in the domains of service and network-management, electronic business, mobility supporting services, and “Intelligent Home” [Albayrak 1998].

Agent Toolkits

In order to actually realize such services an adequate technological approach is required. Available technologies such as middleware, content description languages, directory services, and security frameworks cover only parts of the aforementioned requirements. A better solution would be the provisioning of one comprehensive framework. Such an approach is embodied by the service development toolkit JIAC[2] , which will be discussed in more detail below.

The intrinsic features of agents – interactivity, autonomy, reactivity, and intelligence – let agent-based systems appear to be a promising approach for the realization of open distributed systems as future telecommunications applications will be. Agents provide services in a more flexible way: They may dynamically co-operate in order to share resources intelligently or to delegate subtasks for complexity reasons. In order to support the development process of agent-based systems, methodologies and languages for analysis, design, and programming are needed[3]. In our work, we focus on the concept of an integrating toolkit for the particular tasks involved in agent-based development. Such a toolkit consists of four parts: agent architecture, programming language, runtime environment, and development tools.

The agent architecture determines how agents works: It provides generic functionality for information processing, goal-oriented behavior, communication, and local coordination. Based on the architecture’s representation schemes, the agent description language (ADL) shall allow for a declarative specification of the agent’s specific concepts and capabilities. The compiled ADL-code combined with the agent architecture make up the agent. The purpose of the runtime environment is to provide secure execution of agents as well as additional services like registration, agent migration, and other related telecommunication-specific management functionalities. Tools are settled on top of these concepts: Graphical editors ease the effort of programming, which is further supported by debuggers and monitors, and administration tools allow for setting up, controlling, and maintaining agent applications.

[2] Java Intelligent Agent Componentware. This work presents the current state of our ongoing research, which yields significant changes and improvements compared to earlier versions of JIAC, eg. as described in [Albayrak, Wieczorek 1998].

[3] For further information on agent technology, we refer to the accompanying article from Jennings.

Structure of this Document

In the next section, we describe the JIAC agent architecture, which combines a scalable component framework and a control architecture. Central to JIAC is the concept of services: Agents interact by providing and requesting services guided by protocols. Sections 3 and 4 describe the runtime-environment of JIAC: Electronic market places form the execution environment and deliver further infrastructure such as telecommunication-specific management functionality and security. It’s an outcome of our scalable approach that these properties can either be provided as components, therefore being an integral part of the agents or can be offered externally as services. Section 5 outlines the JIAC agent description language JADL, which combines ontology-based knowledge representation with declarative programming concepts. Tools for agent development, debugging, monitoring, and administration support the development process and system maintenance. Finally, in Section 6 we summarize the outcomes of our work and compare it to related work.

The Agent Architecture

In this section, we describe the agent architecture of JIAC, which has been designed to meet the requirements of telecommunications applications by offering

- openness and scalability by the use of a flexible component framework, allowing for the realization of different agent types by reusable components and the exchange of components at run-time,
- interaction protocols allowing for negotiations and flexible service usage, and
- integration of telecommunication-specific functionalities by the use of dedicated components, as explained in Section 3 and 4.

Component Framework

An agent consists of a set of components that are managed by the *component framework*. Components are identified by the roles they take within an agent and interact by message passing. Messages are delivered to the component currently associated with the role of the receiver. By this way direct dependencies between components are prevented, allowing for their exchange even during runtime. The component framework is also responsible for managing the processing resources of an agent.

Default Architecture

The default architecture realizes a generic control scheme for reactive, deliberative, and interactive behavior following a Belief-Desire-Intention model [Bratman 1987]. It is divided into the *component framework*, the *control unit* controlling the behavior according to the knowledge represented in the *knowledge base*, and the *periphery* for additional and auxiliary functionalities. However, with the component framework it is as well possible to implement other control architectures, e.g. purely reactive ones.

The components of the knowledge base are responsible to store and maintain knowledge. It comprises a fact base, a goal stack, an intention structure, a rule base, a plan library, and a service library.

The control unit operates on this knowledge. Factual knowledge reflecting the current state of the world is managed by the *fact maintenance*. The *situation assessment* reacts to new situations according to rules. For deliberation, intentions, controlled by goals, are selected, coordinated, and executed. The interactive behavior expresses in sending and receiving speech acts guided by protocols in order to provide or to use services.

The periphery contains four types of component roles, comprising application-specific functionalities, speech act transport, as well as management and security tasks.

Interactions and Service Usage

Agents interact by speech acts compliant to the specification of FIPA ACL [FIPA-98]. A common terminology is ensured by shared ontologies.

All interactions between agents are guided by a generic service scheme. Thus, a service describes an act an agent performs on behalf of another agent. Services are specified by conditions, effects, and protocols. Additional parameters specify payment modalities, security requirements, and human user access.

Each service usage is handled by a common meta-protocol. The customer initiates this protocol by requesting the service. If the provider agrees to co-operate - possibly after a negotiation process concerning the service parameters - embedded protocols handle service-specific communications. Finally, the provider transmits the result to the customer.

Management Framework

The telecommunication-specific management framework of the JIAC platform is oriented towards the ITU-T Recommendation X.701. In this standard management activities are divided into specific functional areas, being fault, configuration, accounting, and performance, referred to as FCAP. Additionally, we have realized the features according to the proposed standards of [FIPA-98], partly as management periphery components, partly as services being offered by dedicated management agents.

Management on Marketplaces

JIAC agents reside on marketplaces, being referred to as agent platforms in FIPA. A central role on each marketplace plays the manager agent (MA), which constitutes the role of the FIPA agent platform management agent by providing basic functionality for the operating agents. These management activities comprise the functionalities of the Directory Facilitator, Agent Management System, and Agent Communication Channel. An application may consist of several market places being loosely coupled by the service mediation support of the MAs.

JIAC offers a strong migration concept where code and state of an agent is transmitted. Agent mobility is controlled and supervised by the MAs of the source and destination marketplaces. The process of migration is realized by a dedicated migration protocol that inhibits authentication, authorization, and safe transport of the agent. A localization service allows for tracking mobile agents. On the basis of this functionality, the MA provides a message forwarding mechanism.

Management Functionality inside Agents

Below the marketplace level, rather elementary FCAP management functionalities are located inside an agent as peripheral components.

A wide range of payment models is possible on electronic marketplaces, e.g. remittance, credit card, or cyber coins. For supporting the billing of services, JIAC provides a configurable metering model. Each service session is automatically monitored by an accounting process that collects relevant runtime data. Permanent as well as event-driven logging is possible for all JIAC concepts, being it knowledge, plan activation, migration, or communication.

JIAC supports property-based configuration of an agent's structure. It is possible to exchange or add components during runtime, thus allowing for a flexible version management of services. One can also change the behaviour of a running component by manipulating its property attributes. A fault component provides an error reporting service and allows for manipulation of the knowledge base for error recovery.

Security Framework

The security infrastructure for JIAC offers security mechanisms for agents and marketplaces on different architectural levels, i.e. Java Virtual Machine, Transport Layer (ISO/OSI Layer 4), agent, and agent communities.

The mechanisms of the Java 2 security architecture are used for securing the Java Virtual Machine. These mechanisms include the use of security managers for marketplaces, class loaders for migration, and access control with permissions and policies. JIAC has no predefined transport layer component. At agent design time, the agent designer adds all needed or wanted communication components to an agent. In order to enable secure communication on the transport layer, JIAC offers an SSL component.

Security Functionality inside Agents

On the agent level several security features are realized as agent components.

JIAC offers a protocol, similar to SSL, which can be used for encrypting and signing speech acts on a higher communication layer than TCP/IP. This protocol can be used, if secure communication on the application layer is required. The security protocol precedes the meta protocol for service usage (see Section 2).

With JIAC's service control mechanisms, it is possible to employ and manage access control. This is handled by attaching service control lists to services. By this way, it is possible to restrict service access to a single agent or to an organizational unit specified within a X.509 certificate. Each agent may have a component for the management and handling of this kind of certificates. The certificate management component is a central security component, it is needed by the other security components.

JIAC offers the possibility to manage trust relationships between agent places. Trust relationships are entries in a market place trust list and define, whether migration from and to other marketplaces is allowed.

Security on Marketplaces

Agent marketplaces may contain additional agents, which offer security services. There are three types of security related agents: a certificate authority agent (CA), a security agent, and a security service agent.

The CA is employed by a human user to issue certificates on clients' requests. It should always reside on an own secure agent place. The only task of the CA is to push information to security agents.

On any marketplace there can be a security agent, which has different tasks to fulfill. First, it functions as a key distribution center for an agent place or a domain of agent places. Second, it manages trust relationships between the agent places it belongs to, and other known agent places.

The security service agent is able of creating time stamps, checking signatures, and acts as a trusted third party in a contract conclusion protocol for e-commerce applications.

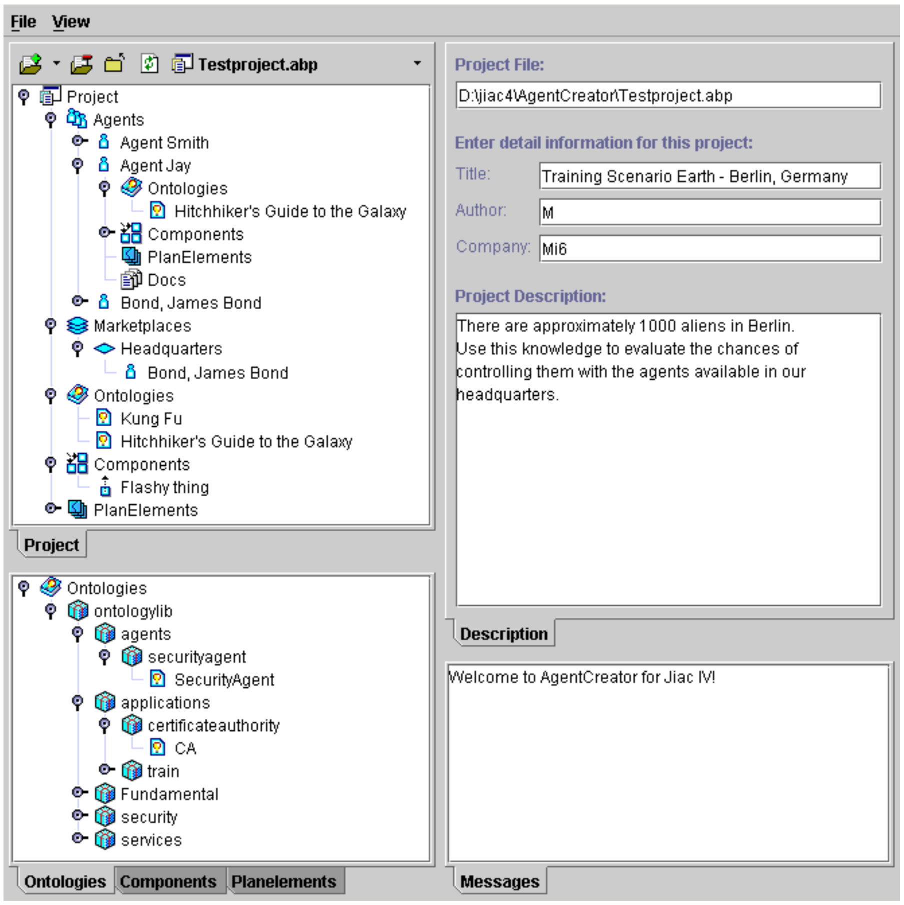
Development Tools

The toolkit as described so far consists of an agent architecture with certain security and management features and a runtime environment based on a JVM with management and security agents forming marketplaces. Given these elementary components at hand, one is able to create applications that are driven by service-provisioning agents.

Because of the powerful information processing features of the JIAC default architecture, only the implementation of domain specific knowledge and capabilities is left to the programmer, thereby relieving him of much work. For this purpose, the JIAC Agent Description Language (JADL) has been designed. Due to its declarative concepts, the use of JADL is a process of specification rather than programming. This development process is further facilitated by a set of tools like knowledge compilers, configuration editors, and debugging instruments.

The first step in programming a multi-agent system is to define the domain vocabulary as a prerequisite for mutual understanding. To this end, JADL offers language constructs for specifying ontological knowledge, such as categories, attributes, and inheritance. Compilers translate ontological representations into Java code that can directly be processed by the agents' components. In the next step, the initial knowledge, reaction rules, and capabilities comprising plans and services of the agents are to be specified. This is done declaratively by using a predicate logic style of programming, relying on the previously declared ontologies. Subsequently the procedural functionality of the plans and services must be programmed in Java as periphery components.

Agent Development Environment



Project File:
D:\jiac4\AgentCreator\Testproject.abp

Enter detail information for this project:

Title: Training Scenario Earth - Berlin, Germany

Author: M

Company: Mi6

Project Description:

There are approximately 1000 aliens in Berlin.
Use this knowledge to evaluate the chances of
controlling them with the agents available in our
headquarters.

Welcome to AgentCreator for Jiac IV!

Figure 1: Agent Development Environment

The Agent Development Environment (see Figure 1) acts as an integration tool. It manages the compilation of ontologies and knowledge, the configuration of the agents' components, and the initial arrangement of agents on marketplaces. Furthermore, it supports project management and documentation.

An agent is generated by specification and configuration of a set of components. The Agent Development Environment treats marketplaces as own entities to reflect the structure of the agent system appropriately. This structure consists of the various marketplaces and the agents being assigned to them.

Agent Debugger

The Debugger provides run-time information and control of an agent. Properties of the components can be examined and reconfigured during runtime without the risk of service degradation. A single-step mode allows for controlled execution.

The Debugger logs internal processes and messages of the agent. For this purpose, different levels of detail for logging activities can be set for each component of an agent. It is also possible to introspect the informational state of the agent by looking into the contents of the knowledge bases, and by monitoring the effects of executed plans and services. For controlling the agent's behavior directly, the Debugger allows for setting new goals.

Marketplace Monitor

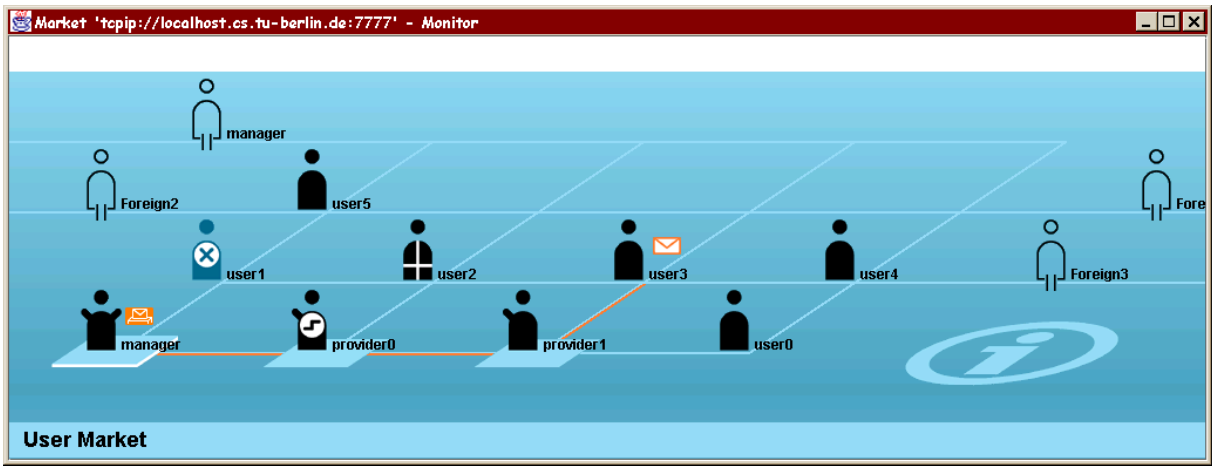


Figure 2: Marketplace Monitor

The activities of the agents on a marketplace are visualized by the Marketplace Monitor (see Figure 2). It shows the agents currently populating the marketplace and their ongoing communications. The symbol of each agent indicates its life-cycle state and whether it’s the manager agent, a stationary agent, a mobile agent, or a remote agent being involved in communications from another marketplace.

Agent Navigator

The Navigator serves as a generic user interface to access the services of an agent system by a human user. Each Navigator is a manager agent constituting a user marketplace.

Via the Navigator, the user can browse the Directory Facilitator by categories and keywords to find services as needed. It handles the initiation and termination of service usage and allows for simultanous service access.

A service is available to human users, if an agent provides a graphical user interface (GUI) for service access. Such agents are called Alter Ego because they act in behalf of a user. When the user selects a service, the Navigator uses a generic service for human-agent interaction at the Alter Ego. First, the Alter Ego transfers the GUI to the Navigator that displays it. Thereafter, the protocol of this generic service allows information exchange between the GUI and the Alter Ego during the service usage. The Alter Ego thereby acts as a gateway by translating GUI interactions into agent communications and vice versa.

Remote Administration

The administration of the agent system is done remotely by a set of administration services. The access to these services takes place via the Navigator, but is restricted to the system administrators by service control lists.

The administration comprises management functionalities for marketplaces, agents, and services. The administrator can search for available entities of these types, gain run-time information about them, and may configure them. At each marketplace, agents can be created and terminated or sent to another marketplace. For each agent, components and knowledge, including services, can be added, removed, and exchanged.

Summary

Services in the telecommunications domain demand short time-to-market spans, a high level of configurability, support for a multitude of devices, network independency, personalization and adaptivity, as well as support for a broad range of functionalities and applications.

In this paper we presented the basic concepts of the toolkit JIAC, which relys on a scalable and open component-based agent architecture for service provisioning. Management and security functionalities contribute to the requirements of the telecommunications domain, whereas a set of tools support an integrated and easy development of agent-based applications in this area.

Several implemented prototypes demonstrate the applicability of the toolkit. For example, in a realized traffic telematics scenario, a mobile user is supported before and on his trip with routing and re-routing services, up-to-date traffic information, car status like fuel level and personalized gas station information, and break-down and emergency support.

Related work

BDI-style agent architectures are based on the work of [Bratman 1987]. Like other BDI architectures (e.g. [Rao, Georgeff 1995]), JIAC doesn't rely on belief-desire-intention logic but rather employs a pragmatic solution of information storage and processing. On the other hand, its modular structure is as well comparable to component-based architectures (e.g. [Hayes-Roth 1995]). The use of toolkits in order to support the development process of agent-based systems is widely-used (e.g. the ZEUS toolkit[Azarmi, Thompson 2000]).

Further contributions to our work stem from standardization gremiums, mainly FIPA. From FIPA we adopted not only the communication language ACL, but also the specifications of agent and platform management [FIPA-98]. JADE [Bellifemine, et al. 2000] is another Java-based FIPA-compliant agent development framework which provides powerful debugging tools. However, no support for telecommunication-specific management functionality is given.

Although security issues are commonly regarded as very important in the context of mobile agents, architectures with enhanced security features are sparse. [MOLE], another mobile agent platform, deals mainly with life-cycle aspects, localization, and transactions. Access control lists are a well-known feature of Aglets [Karjoth, et al. 1997]. In the Ajanta framework [Karnik, Tripathi 1999] security is managed close to our approach by use of credentials and authentication protocols; additionally, it provides a proxy-based mechanism for protected resource access.

References

[Albayrak 1998] S. Albayrak: Introduction to Agent Oriented Technology for Telecommunications. In: S. Albayrak (ed.): Intelligent Agents for Telecommunications Applications, IOS Press, p. 1 – 18, 1998.

[Albayrak, Wieczorek 1998] S. Albayrak, D. Wieczorek: JIAC – An Open and Scalable Agent Architecture for Telecommunication Applications. In: S. Albayrak (ed.): Intelligent Agents for Telecommunications Applications, IOS Press, p. 1 – 18, 1998.

[Azarmi, Thompson 2000] N. Azarmi, N. Thompson: ZEUS: A Toolkit for Building Multi-Agent Systems. Proceedings of fifth annual Embracing Complexity conference, Paris, April 2000.

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

[Bratman 1987] M. E. Bratman: Intentions, Plans, and Practical Reason. Cambridge, MA: Harvard University Press, 1987.

[FIPA-98] Foundation for Intelligent Physical Agents: FIPA 98 Specification. http://www.fipa.org. October 1998.

[Hayes-Roth 1995] B. Hayes-Roth: An architecture for adaptive intelligent systems. Artificial Intelligence 72, 1995, p. 329 - 365.

[Karjoth, et al. 1997] G. Karjoth, D. Lange, M. Oshima: A Security Model for Aglets. IEEE Internet Computing, p. 68-77, July-August 1997.

[Karnik, Tripathi 1999] N. Karnik A. Tripathi: Security in the Ajanta Mobile Agent System, Technical Report, Department of Computer Science, University of Minnesota, May 1999.

[MOLE] http://mole.informatik.uni-stuttgart.de/.

[Rao, Georgeff 1995] A. S. Rao, M. P. Georgeff: BDI-agents: From theory to practice. In: First International Conference on Multi-Agent Systems (ICMAS-95), San Francisco, CA, 1995.