1

Agent Technology for Communications Infrastructure: An Introduction

S. J. Poslad, R. A. Bourne, A. L. G. Hayzelden, P. Buckle

1.1 Introduction

This introductory chapter outlines the increasing need for software support in the design and implementation of the rapidly expanding communications industry and argues that the agent-based computing paradigm is the most promising approach with which to address these issues. Communication systems are expanding both in physical terms, with the exponential growth in internet usage and the mobile phone market, and in the range and complexity of services offered by providers. Recent developments in wireless application programmes (WAP) which are bringing these two technologies closer together only suggest that larger and more sophisticated software systems will be needed to cater for the increasing demand. Approaches that facilitate the design and implementation of these new communication systems are therefore required. Agent-based computing, and in particular research into multi-agent systems, is leading the way in the development of techniques to address the variety of needs associated with these new technologies: languages and protocols which enable autonomous systems to communicate, frameworks for negotiation between self-interested parties, platforms which allow mobile processes to operate at remote sites, and many more.

To provide the interested reader with both the background motivation and an introduction to the software agent philosophy, this chapter outlines the reasons underlying the need for more radical approaches to communication systems design (section 1.2) and the agent-based computing paradigm as a design methodology (section 1.3). For readers more familiar with these issues, section 1.4 reflects the authors' views on the current state-of-the-art in agent technology with pointers to recent research results in its application to the communications industry. In section 1.5, we overview the research papers brought

together in this volume which, though not comprehensive, provide a snapshot of both the variety of problems which can be addressed using software agents and the novel ways in which researchers are attacking them.

1.2 Communications Infrastructure for Next-Generation Services

1.2.1 Mass production for a mass consumption service infrastructure

The start of the electronic communication and information service industry was dominated by a relatively small number of powerful service providers and service delivery channels. These faced little or no competition, were heavily regulated and licensed, and offered a relatively fixed product and service range. The product and service requirements were supplier-driven, determined and fixed by mass production, and they supported limited customisation by the end-user. A poor match between the user requirements, perceived by the suppliers before production and the actual user requirements impaired service uptake. Customers often needed to modify their business models and requirements substantially to fit them to the information and communication products on offer (McDermid, 1998).

Under pressure from different market forces such as increasing privatisation, deregulation, diversification and competition, the customer beholds increasing choice, complexity and heterogeneity at a variety of infrastructure levels including types of message transport (e.g. wired-voice, wireless-voice, wired IP), portal, service and customer interfaces. The communications and information industry along with other types of industry is evolving from a predominantly mass production model towards a model that also incorporates a mass customisation service infrastructure.

1.2.2 Consumer-driven customisable service infrastructure

Customisation can be defined as the ability of a product or service to be modified and maintained to meet particular customer requirements or profiles. Customisation can occur at multiple levels and can be customer, third-party (e.g. broker) or provider driven. Here customer (or consumer), broker, and provider, are just temporal roles played depending on an entity's position in the supply demand chain for a particular service. Mass customisation can be considered as a form of mass production, differing from it only in terms of granularity and abstraction. For example, in the late 1990s, customers became able to switch to an additional service provider for cheaper international voice calls by dialling additional access numbers at the start of a call; at this level, customisation is customer-driven. A new communication service can be synthesised from combinations of services from different vendors, resulting in a cost reduction for both long-distance and other calls.

However, the complexity of integration of multiple service components may overwhelm the consumer. For example, service providers may appear or disappear overnight; lucrative service opportunities may be temporal in nature; and integration may require specialist knowledge that is not available to the customer. This seeds a growth in third-party intervention, such as human and electronic service brokerage, offering independent advice to users to help them intelligently search and select service

combinations. A broker may be able to simplify, configure and coalesce access to multiple services for the user. For example, TV set-top boxes, telephones and PCs can all behave as a service portal to integrate information from the internet, voice conversations and broadcast video.

For optimal customisation, service providers need to produce interfaces and delivery channels with suitable designs and abstractions to support reconfiguration and cooperation. For example, although customers may simultaneously hold several contracts on different tariffs between multiple service providers, there is no easy way to produce an integrated service bill – this may require existing services and channels to be re-engineered or wrapped.

Providers or third parties who wish to customise or slant services and products for selected customers (e.g. through unicast or multicast rather than broadcast), will require detailed and up-to-date access to personal and group customer profiles. In contrast, for customers to customise services requires timely access to service and product configuration information. Due to the potential temporal nature of both providers and their products and services and the opportunity to re-evaluate contracts when desired, automated processing of these requirements becomes expedient.

1.2.3 User requirements for a highly customisable service infrastructure

We now discuss several specific user requirements that support a highly customisable service infrastructure, extending the generic requirements of distributed systems such as openness and scalability. These comprise:

- Accessible, configurable and secure service profiles
- Accessible, configurable and secure customer profiles (also called personal profiles)
- Sophisticated advertising and matching of service provision to service requests
- Cooperation, coordination, control and coherence of groups of autonomous service providers and users.

Service profiles are a high-level publicly configurable form of the service interface that is distinct from the internal developer interfaces. The former are abstract and modular enough to be customisable by service users and their proxies (others designated by users to act on their behalf because they provide the necessary know-how). Service profiles define which elements of the server can be modified. Some sort of grand design is clearly necessary to constrain the service as a whole, and to prevent chaotic and improper use of the service. However, a complete grand design is usually not a viable option, especially for extensive, long-lived software, since its future environment may not be known in advance, and fixed designs may alienate users. Instead, designs should support *habitability*, a specialisation of customisation – this is the characteristic of a service that enables users or their proxies to be comfortable and confident in maintaining or modifying that service (Gabriel, 1996). Although service interfaces may be available, they may not be habitable; for example, they may not be configurable by the user, users may not understand how to use them, or they may be too low-level. Habitable services provide interfaces at the correct level of abstraction and modularisation.

Customer profiles that characterise customer preferences need to be defined to enable services to be customised. Specific vendor-driven initiatives such as *directory-enabled networking* (DEN)[1] seem attractive here, but they often lack the appropriate abstractions and open design to support all parts of this requirement. For example, DEN has focused on modelling, storing and retrieving schemas, which relate customer profiles to network service profiles in directory services supporting LDAP (Light Directory Access Protocol) and Microsoft's Active Directory model. DEN primarily focuses on modelling the information network providers′ need to exchange in order to provide interoperable network services such as constructing and supporting end-to-end quality of service across multiple providers, rather than on modelling how users can configure network provision. However, DEN does not define security nor does it define a protocol or policies for exchanging this information between users and service providers.

Another user-specific requirement is support for sophisticated advertising and matching of usage and provision. For example, if a service request cannot be satisfied by a single provider then flexibility is required to progress beyond a simple denial of service – in this case a service request may be satisfied by aggregating possible parts of various services from more than one provider. This requires a holistic view of service provision matching which may involve a more complex negotiation protocol to exchange and agree the specifics. We also need to be able to share a common understanding of information across multiple domains. For example, while frame-relay network providers may define quality of service using parameters such as FECNs and BECNs (Forward and Backward Congestion Notification bits), customers may better understand terms such as peak and average throughput, delay and delay jitter, discard/loss rate and overall availability.

Finally, we need groups of services to be controllable, i.e. to be coordinated and cooperative so that the set of services as a whole is coherent. This becomes increasingly necessary when participating services are autonomous, or do not belong to static organisations with a fixed command and control structure. For example, when previously autonomous services are aggregated, cooperation often involves dynamic peer-to-peer interaction to resolve problems such as feature interaction or resource contention over bandwidth.

1.2.4 Support for next-generation services

The issues raised above have implications for the development and maintenance of current and future e-commerce and e-business applications and the underlying network infrastructure. All such services will require rapid and flexible implementation of software support. There is clearly a need to ensure that this can take place using robust but adaptable design solutions. We argue herein that the agent-oriented programming paradigm facilitates this process and is therefore the most likely design methodology to fulfil the requirements of the next generation of services.

The research featured in the coming chapters amply demonstrates the flexibility of agent-oriented design when applied to new services such as Virtual Home Environments (VHE) and customisable datacom services, as well as enabling the rapid development of

[1] DEN. Available: http://www.dmtf.org/spec/denh.html [18 May 2000].

simulation platforms from which to test the impact of new services. Furthermore, in the area of network management and infrastructure, the use of software agents has led to the development of radical new solutions and capabilities.

Before the reader dives into these exciting new software approaches, there is a need to reflect on the underlying themes of agent-oriented programming and to highlight the reasons that this approach is well-suited to such a diverse problem domain.

1.3 Agent Architectures

At their simplest, software agents are just independently executing programs – often implemented as multiple threads or UNIX-like processes – which are capable of acting autonomously in the presence of expected and unexpected events. Agents can be of differing abilities, but typically possess the required expertise to fulfil their design objectives. To be described as 'intelligent', software agents should also possess the ability of acting autonomously, that is, without direct human input at run-time, and flexibly, that is, being able to balance their reactive behaviour, in response to changes in their environment, with their proactive or goal-directed behaviour; in the context of systems of multiple autonomously acting software agents, they additionally require the ability to communicate with other agents, that is, to be social (Wooldridge, 1999).

The central concept which distinguishes software agents from simple programs is their interaction with their environment; typically, we say that an agent is embedded or situated within its environment (Jennings and Wooldridge, 1998). While it is true that any distributed system could be implemented as a single centralised system, such an approach ignores the fact that, in some types of environment, information and control are naturally distributed throughout the system; this is particularly true of communication systems. Distributing agents to deal with information and control locally, while enabling communication at higher levels of abstraction, can make systems easier to understand and hence make them easier to design and develop.

In this sense, agents can be embedded throughout a system performing functions with appropriate levels of expertise. For example, in chapter 11, it is shown that using a layered hierarchy of agents to manage a network (similar to an organisational structure), enables both long-and short-term decision-making to occur independently, providing robust and rapid responses to short-term fluctuations in network demand, with longer-term considerations such as network survivability being controlled at the higher management plane.

Agents can act as intelligent decision-makers, active resource monitors, wrappers to encapsulate legacy software, mediators, or as simple control functions. The common link between this vast array of possibilities is that each agent autonomously strives to meet its own design objectives. In some ways, agent-based computing can be seen as a natural extension of object-oriented programming whereby objects are empowered with their own thread of control and their own goals or objectives, so that they autonomously control their own behaviour in order to reach their goals. An excellent introduction to intelligent agents and multi-agent systems can be found in Weiss (1999).

1.3.1 Mobile agent systems

The term mobile agent encompasses two distinct concepts: mobility and agency. While some researchers regard the mobility of an agent as its defining feature, with the support for agent behaviour secondary, others regard an agent with mobility to be simply a special type of agent.

Mobility can provide a useful mechanism to enable modification of both services and customer profiles at separate locations, potentially protecting privacy of both parties; subsequently, the updated service or profile can be transferred to the other party for integration. However, additional abstractions are required, such as agency, to support advertising and matching of service usage to service provision, and to support cooperation, coordination, control and coherence of providers and customers. Generally, the mobile agent frameworks or tools mean that there is a large platform overhead for many problem domains – the associated overhead of providing a mobile agent solution may outweigh its benefits in some domains.

1.3.2 Multi-agent systems

The ability of an agent to be social and to interact with other agents means that many systems can be viewed as *multi-agent systems* (MAS). Our definition of software agents as autonomous processes means that they encapsulate their own state and behaviour. For agents to interact, they must also possess the ability to communicate with other agents via some common *agent communication language* (ACL). Similarly, for agents to cooperate with one another or to coordinate activities, requires some inherent desire to be social, whether this is due to self-interested, utility-maximising criteria among competitive agents, or because the system as a whole is designed to achieve some ultimate goal, through essentially cooperative agents pursuing the common good. Any such MAS must include infrastructure to support transport and directory services among agents, ACL message encoders and parsers, dialogue management, and brokers and facilitators.

Multi-agent systems can adopt a range of sophisticated approaches for advertising services, customer preferences and subsequently matching service usage to service provision through interaction between service, user and facilitator agents. Agents can deploy a variety of strategies for the cooperation, coordination, control and coherence of groups of services and users. For example, service and user agents can send messages within the context of interaction sequences such as client server interactions, auction mechanisms, the contract net protocol, and different types of brokerage to support complex negotiation strategies. Multiple providers and users who share common service and customer ontologies have a basis to gain a much deeper agreement in the provision and use of services.

1.4 Agent Technologies

Having made a case for exploiting agent and multi-agent system designs to develop next-generation services and infrastructure, we now turn to the state-of-the-art support for agent technology. This section describes the current status of agent standards, communication

languages and toolkits which support development of both open and dedicated multi-agent systems. The FIPA agent standards and the agent toolkit FIPA-OS are then looked at in more detail.

1.4.1 Agent standards

Early adopters of new technology tend to be wary when there are no commonly agreed, well-defined interfaces that are also backed by industry-supported standards. The standardisation process shifts the emphasis from longer-term research issues to the practicalities of realising commercial systems. For this reason, there has been a growing force behind developing agent standards. Such standards provide added confidence to potential adopters of agent technology. However, full compliance to complex agent standards can lead to large overheads in terms of time-to-market and adherence to unnecessary features. Moreover, if standards bodies are not careful, users will not take on board this concept – speed, efficiency and robustness are still top priorities in the communications industry.

Agent standards bodies have reflected the distinction between stationary and mobile agents: the Foundation for Intelligent Physical Agents (FIPA)[2] and the Object Management Group's Mobile Agent System Interoperability Facility (MASIF).[3]

FIPA is a non-profit standards organisation established in 1996 to promote the development of specifications of generic agent technologies that maximise interoperability within and across agent-based applications. Part of its function is to produce a specification for an agent-enabling software framework. Contributors are free to produce their own implementations of this software framework as long as its construction and operation comply with the published FIPA specification. In this way, the individual software frameworks are interoperable. The FIPA agent standard aims to bring the commercial world a step closer to true software components, the benefits of which will include increased reuse, together with ease of upgrade. FIPA allows for focused collaboration (of both industrial and academic organisations) in addressing the key challenges facing commercial agent developers as they turn agent technology into products.

OMG's MASIF differs from FIPA in that it regards the mobility of an agent from one location to another to be its defining characteristic. MASIF defines interaction in terms of Remote Procedure Calls (RPC) or Remote Method Invocation (RMI); it does not support or standardise interoperability between non-mobile agents on different agent platforms. Further, MASIF restricts the interoperability of agents to agents developed on a single type of platform (e.g. agents deployed on one Grasshopper platform can migrate to another Grasshopper platform) whereas the focus of FIPA is to directly support the interoperability of agents deployed on heterogeneous agent frameworks.

Stationary and mobile agent standards are, however, starting to overlap and may ultimately converge. There is strong interest within FIPA, particularly by wireless network operators and service providers, to extend its approach to support mobile agents.

---

[2] FIPA. Available: http://www.fipa.org [18 May 2000].

[3] MASIF. Available: http://www.fokus.gmd.de/research/cc/ecco/masif/index.html [18 May 2000].

Meanwhile, the Object Management Group is exploring the development of standards for stationary agents in addition to mobile agents, and FIPA is inputting into this process.

1.4.2 Agent communication languages

Communication enables agents to coordinate their actions and behaviour, resulting in systems that are more coherent; through coordination, agents can better achieve their design objectives (their own or the system's goals). Whether agents inhabit a dedicated system, or an open one, some means of communication among agents needs to be supported. Long before the drive towards agent standards was initiated, attempts at standardising agent communication languages was well under way.

Communication among agents has most successfully been modelled using *speech act theory* (Searle, 1969). Speech act theory uses the term *performative* to identify the intention behind a spoken communication; examples include verbs such as request, tell, report, commit or reply. These performatives are used to constrain the semantics of each act of communication among agents, so simplifying how agents should react to messages they receive.

Knowledge Query Meta-Language, or KQML (Finin and Fritzson, 1994), was one of the first initiatives to specify how to support the social interaction characteristics of agents using a protocol based on speech acts. However, KQML is not a true de facto standard since there is currently no consensus in the community on a single specification (or set of specifications), nor has it been ratified by common agreement between members of an organisation or forum of some standing in the community. As a result, variations of KQML exist and different agent systems may speak different dialects which prevents full interoperability between them.

Any act of communication involves three aspects: the method of message passing, the format, or syntax, of the information being transferred, and the meaning, or semantics, of the information (and message). The syntax of a generic message in KQML is given in Figure 1.1.

```
(KQML-performative
      :sender   <word>
      :receiver <word>
      :language <word>
      :ontology <word>
      :content   <expression>
      ...)
```

Figure 1.1 The syntax of a generic KQML message

One of the main problems with KQML, notwithstanding the lack of an agreed specification, is its lack of a well-defined semantics. The use of performatives alone is insufficient to guarantee that messages will be interpreted and acted on correctly by other agents.

This lack of semantics is one of the driving forces behind the FIPA specification of its own agent communication language (ACL) which provides an attempt at a universal message-oriented communication language (FIPA, 1997). The FIPA ACL describes a standard way to package messages, in such a way that it is clear to other compliant agents what the purpose of the communication is. Although there are several hundred verbs in English corresponding to performatives, the FIPA ACL defines what is considered to be a minimal set for agent communication (it consists of approximately 20 performatives). This method provides for a flexible approach for communication between software entities exhibiting benefits including:

- Dynamic introduction and removal of services
- Customised services can be introduced without a requirement to recompile the code of the clients at run-time
- Allowance for more decentralised peer-to-peer realisation of software
- A universal message-based language approach providing a consistent speech-act-based interface throughout software (flat hierarchy of interfaces)
- Asynchronous message-based interaction between entities.

Another attempt at providing a common language among agents and legacy software, is the Knowledge Interchange Format or KIF (Huhns and Stephens, 1999). This is based on first-order logic which is capable of describing almost any concept of interest or utility to people and other intelligent agents. Agents which can translate to and from KIF have the potential to exchange information with any other KIF-speaking agent or software system.

Even if agents speak the same language, they require some common understanding of the meaning of the message content. This is provided by a specialised knowledge component called an *ontology* which specifies the objects, concepts and relationships in a given domain (Guarino, 1995). Agents which share a common language and an ontology can communicate effectively both syntactically and semantically.

1.4.3 Agent toolkits

For agent technology to fulfil its promise, agent toolkits need to become widely available and must be robust enough to be used in e-business sectors such as telecommunications. These tools must be able to be applied to address the customer requirements and ideally they need to adhere to standards which define multi-party agent interoperability.

Currently, a profusion of agent toolkits (see Table 1.1) are being made available for use by third-party developers, application developers and end-users under licensing arrangements which range from open source to commercial. Agent tools and systems have mainly been developed within research laboratories funded by governments; at this time, they are being deployed to develop and test concept demonstrators and very specialised one-off applications for a new generation of communications services, rather than to underpin core business solutions or mass-production end-user applications. This new generation of communication services include: Virtual Home Environments, Virtual Private Networks, video conferencing, etc.

For agent toolkits to be taken up by application developers, moving out of the laboratory and into more mainstream development, they need to:

- Provide suitable abstractions, interfaces
- Adhere to appropriate and proven industry standards
- Generate software that can be embedded within a non-agent software infrastructure.

Agent toolkits are defined as *sets of components* from which to build agent systems and *sets of tools* to help operate agent systems. The agent system is somewhat different from the agent toolkit itself in that when it executes, it represents one or more particular configuration(s) of the toolkit.

The first generation of multi-agent systems, which derived from Distributed Artificial Intelligence (DAI) in the 1980s, consisted of components which were tightly integrated, often used a proprietary, shared memory-data model to communicate, and were implemented using specialised languages, often with proprietary extensions. The concept of agent toolkits emerged in the next generation of agent systems in the 1990s (see Table 1.1), and used standard message-passing protocols to communicate knowledge or to migrate the agent itself to a new location. These were implemented using mainstream languages such as C++ or, more commonly, Java. Common functions such as agent-level communication, message transport and directory service were encapsulated into components with well-defined interfaces.

Table 1.1 Publicly available agent toolkits

Product Name (Company) URL	Mobile / Stationary	Language	Standards	Availability/ Licensing	Example applications
AgentBuilder (Reticular Systems) http://www.agentbuilder.com	Stationary Mobile	Java		Evaluation, Academic	E-commerce, auctions, job-finder, private e-mail, etc.
Aglets (IBM Japan) http://www.trl.ibm.co.jp/aglets	Mobile	Java	MASIF	Evaluation	Remote Monitoring (e.g., Web site), meeting scheduling, auctions, etc.
APRIL (Fujitsu, USA) http://www.nar.fujitsulabs.com		APRIL	FIPA	Open Source	Schedule Mgt, etc.
Comtec Agent platform (Japan) http://www.fipa.org/glointe.htm	Stationary	Java	FIPA	Open Source	Enterprise Information Mgt.
Concordia (Mitsubishi) http://www.meitca.com/HSL/ Projects/Concordia	Mobile	Java		Evaluation	Information retrieval, etc.
DMARS (AAII) http://www.aaii.oz.au/proj/ dMARS-prod-brief.html	Stationary	C/C++		Superseded by Deimos	Air traffic control, TNM, simulation, fault diagnosis, etc.

Table 1.1 (continued)

FIPA-OS (Nortel Networks) http://www.nortelnetworks.com/fipa-os	Stationary	Java	FIPA	Open Source	Personalised services, VPN, VHE, meeting scheduler, etc.
Grasshopper (ikv++) http://www.ikv.de/products/grasshopper/grasshopper.html	Mobile	Java	MASIF/ FIPA	Evaluation	E-commerce, information retrieval, etc.
Jackal (UMBC) http://www.alphaworks.ibm.com/tech	Stationary	Java	KQML	Evaluation	Enterprise information mgt, manufacturing planning, etc.
JADE (CSELT) http://sharon.cselt.it/projects/jade	Stationary	Java	FIPA	Open Source	Travel assistant, audio-visual entertainment, meeting scheduler, etc.
JAFMAS (Uni. Cincinnati) http://www.ececs.uc.edu/~abaker/JAFMAS	Stationary	Java	KQML	Unrestricted licence	Supply chain mgt, etc.
JATLite (Stanford Uni.)	Stationary	Java	KQML	Open Source	Design decision tracking, constraint mgt, enterprise control.
Jess (Sandia Nat. Labs) http://herzberg.ca.sandia.gov/jess		Java		Academic	Various.
MOLE (Uni-stuttgart) http://mole.informatik.uni-stuttgart.de/	Mobile	Java		Evaluation, Academic	information retrieval, groupware, active documents, etc.
Open Agent Architecture (SRI) http://www.ai.sri.com/~oaa/main.html	Stationary	Java, C, VB, Lisp, Prolog, etc.	KQML	Academic	Automated office, robot control, collaborative fridge, etc.
Voyager (ObjectSpace) http://www.objectspace.com/	Mobile	Java		Academic	Various.
ZEUS (BT UK) http://www.labs.bt.com/projects/agents/zeus/	Stationary	Java	FIPA KQML	Open Source	Supply chain mgt, service provisioning, network resource mgt.

Table 1.1 classifies agent toolkits according to whether they support stationary agents (intelligent messenger) or mobile agents (intelligent message), or both. In addition, it highlights the language in which they are implemented, which communication standards they support at the agent level, the type of availability and licensing, and the types of applications or services they have been used to develop.

The availability of toolkits can range from use of the binaries for a fixed evaluation period, to unlimited use for academic (i.e. non-commercial) purposes, to unlimited use of both the binaries and source code for academic use, to a full open source-code licence for academic and commercial use. JATLite was one of the first stationary agent toolkits made publicly available; it later became available as open source. FIPA-OS was the first toolkit available under full open source (at least the first one that supports the FIPA agent

standards); it was released by Nortel networks in October 1999. Since that time, others such as BT's ZEUS platform, CSELT's JADE and Comtec's agent platform have followed suit.

Sun's Java is the current hot implementation language for agent toolkits, with one or two exceptions such as APRIL (a language in its own right) and dMARS (implemented in C++). The latter is no longer available, but has been superseded by Deimos which is implemented in Java. Some developers, particularly those in the DAI community, consider an object-oriented language imperative, but languages like Java tend to be fairly static and unsuitable for the kind of dynamic rule and fact manipulation required to support a notion of intelligence. Although new Java objects can be loaded dynamically, this is at a fairly coarse level of granularity and does not have suitable abstractions and formalisms to support sophisticated rule-based, reasoning systems. To support this latter requirement, Jess (Java expert system shell), a rule engine and scripting environment, was developed by Ernest Friedman-Hill, at Sandia National Laboratories in Livermore, CA. Using Jess, you can build applications that have the capacity to 'reason' using knowledge supplied in the form of declarative rules. JESS, which is implemented in Java, is increasingly being used in agent toolkits such as FIPA-OS, JADE and JATLite, and to support complex reasoning. For this reason, we have included JESS in Table 1.1, although strictly speaking it is not an agent toolkit.

There are many multi-agent systems currently being researched, developed and used. The remainder of this section discusses a subset of these, focusing on those which use an agent communication language for communication between multiple agents and which support the FIPA specification, hence promoting openness in terms of extensibility and licensing.

Most agent platforms, including FIPA and non-FIPA platforms, naturally offer openness at the agent level; whether or not the platform itself is fixed or closed, service and user agents can be dynamically added to the platform and can interoperate. This requires a common means of representing (encoding), understanding (ontology) and exchanging (protocol) service information. FIPA platforms define a standard protocol based on speech acts and several standard ontologies: a core one for registering, querying and deregistering services (part of the FIPA management ontology) and various domain-specific ones; there is no mandatory specific service encoding. Non-FIPA platforms require the use of platform-specific service ontology, encoding, and protocol combinations.

Many agent platforms support some type of bootstrap process which includes agent synthesis for resident agents; often an agent shell or agent factory API is defined. The shell contains hooks into the platform to use lower-level services such as a transport service. To synthesise agents, it is not strictly necessary to use the platform API. Providing a standard protocol such as a TCP/IP can be understood by the platform for the exchange of messages, the non-resident agent can be synthesised externally and registered with the platform. Of course, resident and non-resident agents may be managed by the platform differently.

Several non-FIPA platforms such as OAA, JATLite and JIAC offer some degree of openness for reconfiguration, including replacement of the platform services. For example, JIAC describes several interchangeable components including service broker, message transport, knowledge base and security.

There are now several MAS platforms that report support for the FIPA agent standards: JADE, Grasshopper, ZEUS and FIPA-OS. Of these, only FIPA-OS, JADE and ZEUS are

freely available under a general public licence and are released with source code under an open source licence.

There are several important differences between FIPA-OS, JADE and ZEUS. FIPA-OS (Poslad et al. 2000) and JADE were designed at the onset to focus on supporting agent communication using the FIPA agent standards, whereas ZEUS initially supported only KQML as the agent communication language and has since been rewritten to support the FIPA ACL. FIPA-OS was designed to operate in a heterogeneous open service environment – it does this by supporting multiple transports such as IIOP using a variety of CORBA APIs, RMI and TCP and by supporting multiple encodings for the content. The interoperability of the publicly released version of FIPA-OS is being evaluated in a heterogeneous FIPA environment in the FACTS project. The current ZEUS platform supports Part 2 of the FIPA ACL, but it does not yet support Part 1 of the FIPA 1997 specification (agent management), nor has its interoperability with non-ZEUS FIPA platforms in a heterogeneous environment yet been evaluated. FIPA-OS initially focuses on providing abstractions and interfaces for developers who wish to extend, enhance and integrate an agent platform with existing software infrastructures. ZEUS provides many higher-level abstractions for developing agents – it hides the API and instead provides an integrated development environment to allow developers to configure the agent platform.

1.4.4 The FIPA agent standards

There are several versions of the specifications for FIPA standards in existence. Currently the most widely implemented version is FIPA 97 Version 2.0. In practice the two core parts are: Part 1, which defines the agent platform or agent reference model, and Part 2, which specifies the agent communication language. The FIPA 97 agent reference model (Figure 1.2) provides the normative framework within which FIPA agents exist and operate. Combined with the agent life cycle, it establishes the logical and temporal contexts for the creation, operation and retirement of agents.

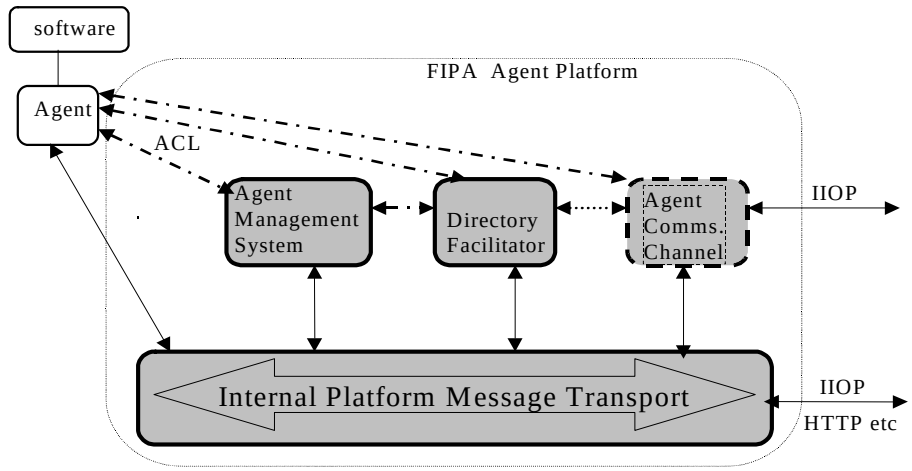

Figure 1.2 The FIPA 97 agent reference model

The agent reference model consists of a Directory Facilitator (DF), Agent Management System (AMS) and Agent Communication Channel (ACC). These are the specific capability sets for agents to support agent management. The DF provides 'yellow pages' services to other agents. The AMS and ACC support interagent communication. The ACC supports interoperability both within and across different platforms. The Internal Platform Message Transport (IPMT) provides a message routing service for agents on a particular platform, which must be reliable and orderly. The ACC, AMS, IPMT and DF form what are also termed the Agent Platform (AP); these are mandatory, normative components of the model. The ACC, AMS and DF are capability sets – they may be performed by a single agent, or three different agents (this is left to the agent platform developer).

To be minimally FIPA-compliant requires compliance to FIPA 97 V2, Part 1, Agent Management, and FIPA 97 V2, Part 2, Agent Communication. There are differing degrees of compliance to, and interpretation of, FIPA compliance. For example, if the FIPA platform has no requirement to support access to non-agent services by agents, then adherence to Part 3 is not required.

The FIPA standards in some areas introduce conceptual problems for designers and implementers. For example, the FIPA Agent Communication Language (FIPA ACL) focuses on an internal mental agency of beliefs, desires and intentions, and closure is not enforced (agents are not compelled to answer); both these concepts hinder multi-agent coordination. The FIPA normative specifications are not intended to be a complete blueprint or specification for building a multi-agent system. For example, FIPA standards do not prescribe how to manage the existential aspects of agents in a discrete world; nor do they define error handling, although some aspects of error reporting are covered. Useful information for developers is also given in an informative part of the FIPA 97 Developers' Guide, Part 13.

The FIPA specifications themselves may evolve to solve current shortcomings and to meet future needs. For example, at present, agents are managed via a message-passing interface at the ACL level, i.e. agents are managed by interaction with the three core FIPA platform agents: the DF agent, the AMS agent and the ACC agent. Currently, an agent often needs to send a forward request to the ACC agent in order to send the payload message. FIPA is currently considering offering one or more of these core services via a non-agent interface such as a method invocation interface rather than by encapsulating it as a service whose sole interface is via the specialised service agent. This may reduce the degree of message passing required during bootstrapping and during the session, thus enhancing scalability. For these reasons, 'FIPA-compliant' agent platforms may also need to evolve to comply with new or replacement standards.

1.4.5 FIPA-OS: an agent platform based on the FIPA standards

FIPA-OS is designed to support the FIPA agent standards. The FIPA reference model discussed earlier defines the core components of the FIPA-OS distribution. Currently, there is no formal or clear mechanism to determine the compliance of a FIPA architecture implementation. Moreover, there are different versions of the architecture, which are not

completely backward-compatible. For these reasons, it is only possible to describe which features of a FIPA specification version are incorporated within a given implementation.

At the time of writing, FIPA-OS includes implementations of the core components of the FIPA 97 reference model. In addition, the FIPA-OS distribution includes support for:

- Different types of agent shells for producing agents which can communicate with each other using the FIPA-OS facilities
- Multi-layered support for agent communication
- Message and conversation management
- Dynamic platform configuration to support multiple IPTMs, multiple types of persistence (enabling integration with legacy persistence software)
- Abstract interfaces and software design patterns.

The FIPA-OS architecture can be envisaged as a non-strict layered model, supported by an underlying component model (Figure 1.3). In a non-strict layered model, entities in non-adjacent layers can access each other directly. The developer is able to extend the architecture not only by appending value-added layers, such as specialist service agents or facilitator agents, but in addition, lower or mid-layers can be replaced, modified or deleted.

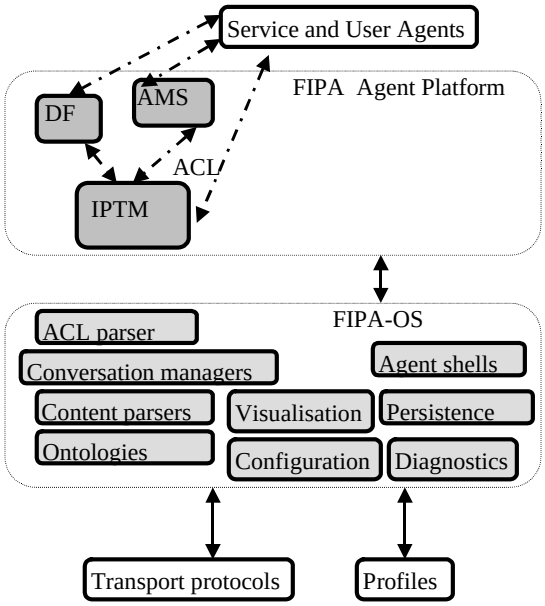

Figure 1.3 A schematic diagram of the FIPA-OS architecture

1.4.5.1 Agent shell

There are a variety of ways in which new agents can be added to FIPA-OS platforms. Agents can be built using different types of agent shell. Agent developers can also create agents without using FIPA-OS services; provided these 'non-resident' agents support a transport protocol supported by FIPA-OS, they can use FIPA-OS to interact with other

agents on the platform. Non-resident agents can use or invoke the transport API of the platform directly or use their own non-platform transport.

1.4.5.2 Multi-tiered ACL communication

Understanding an ACL message requires processing the message with regard to its temporal position within a particular interaction sequence between two or more agents. This involves understanding the type of communication called a communication act (which is specified in the message, e.g. a request, statement of fact, or query), understanding the structure of the content and finally understanding the semantics of the request.

As ACL communication is so rich, it is often represented as a multi-tiered layer in its own right. FIPA-OS supports ACL communication using four main sets of components: conversation, ACL message, content (syntax) and ontology (content semantics). Not all of these components need to be used by each agent, and combinations of different types of components can be supported at each layer (see below). This flexibility is needed because, in a heterogeneous world, different agents may encode and transport the content differently.

1.4.5.3 Conversation management

Multiple messages are sent as part of an interaction protocol, for example, the base FIPA multi-agent interaction protocol called FIPA-request supports client server request reply-type dialogues. Unlike typical RPC invocation, requests are indirect. Agent A can request another agent B to perform some action on its behalf; agent B normally sends back an agree or a refuse message as a reply to agent A.

Without conversation or dialogue management, messages are sent and received independently of any context within any interaction pattern or conversation in which they occur. It is more difficult here to detect whether failures such as an absent reply or inappropriate reply occurred.

As an aid to conversation management, FIPA provides two hooks in the ACL itself but the semantics of those hooks are yet to be defined. Firstly, the FIPA ACL defines two fields ‘reply-with’ and ‘in-reply-to’; however, the semantics of how these fields are to be used by the sender and receiver, and how the values of these two fields are related, are as yet undefined. Secondly, the FIPA ACL defines a Conversation-ID field; FIPA-OS defines semantics to use the Conversation-ID field to coordinate conversations.

1.4.5.4 Customisation

There is built-in support to plug in different components at key interfaces or hot spots including: transport, message content encoding and ACL message storage. These are plugged in during platform initialisation. The plug-in nature of the platform supports a dynamic component-oriented middleware model, and promotes a combination of thin client agents and thin platform services. Agents are not required to implement platform services themselves: they access these services in the platform (supporting thin agents), and platforms do not need to load support for all types of each service at start-up (leading to thinner platforms).

The types of these plug-and-play components are currently configured in terms of profiles for the platform as a whole and for each individual agent. These profiles are encoded in XML/RDF and stored in resource files.

1.5 Summary and Following Chapters

This chapter has provided the reader with an introduction to agent technology with a focus on how agents have been applied to service delivery platforms. We have discussed various aspects of agent technology, from the discussion of what is an agent, through the various architectures that designers can use to design and build agent systems, to the latest on how agent standards are evolving. The later sections have led the reader through various detailed applications of agent technology. Chapters 2 to 10 of this book extend these ideas by considering the issues related to service and application support for the enabling of e-business. Chapters 2 and 3 provide the scope of application of agent technology in networked environments and provide an insight into how agents in software terms can be developed for e-business applications. Chapter 4 describes the use of learning agents to capture feedback control for multimedia communications management.

E-business can be described as the process of conducting trading or facilitating traditional commerce activities via the use of the Internet. At the time of writing, using the Internet as a platform to deliver novel services and for conducting business processes has become one of the most discussed topics in the popular press. In this vein, chapter 5 is concerned with how agents may facilitate a variety of relevant e-business functions, such as advertising, matchmaking and brokering; the chapter presents a general survey of key enabling techniques that are needed to build institutions of agent-mediated trading within the Internet. Chapters 6 and 7 focus on the security issues that are intrinsic to networked systems. Chapter 6 relates to the security problems and possible solutions for using agent technology for resource allocation such as deciding on bandwidth utilisation measures. Chapter 7 tackles the concerns about secure transactions for buying and selling products over the Internet with a special focus on how agent technology can provide one such solution. Chapters 8 and 9 offer differing approaches to how multi-agent systems can provide a solution to the problem of delivering information to the mobile user. Chapter 10 continues in a similar application stream and explains the use of algorithms and agent technology for planning and supporting individual travel services.

Chapters 11 onwards constitute the second Part of the book and cover the application of agent technology to network infrastructure. With the rapid development of new and larger communications networks comes an increased need for integration of diverse and distributed underlying hardware and middleware platforms. This need has led to a variety of interesting agent-based solutions, some of which are represented here. Chapters 11, 12, 13 and 14 describe agent architectures that have been used for network management and control in both wired and wireless networks. Chapters 15, 16 and 17 show how distributed monitoring of real-time network parameters by agents has been used for prediction, survivability and optimisation. Chapters 18 and 19 apply market-based control techniques to allow economic agents to manage call routing and load control in both ATM and intelligent networks. Chapter 20 uses a collaborative multi-agent architecture to track and resolve faults in global connections. Chapter 21 shows how agents have been used to deal with distributed search for resources. Chapter 22 enables agents to evolve routing algorithms using genetic programming. While this selection can only provide a snapshot of on-going research in both academia and industry, it is clear that researchers in the network domain have found the conceptual framework of agents and multi-agent systems particularly applicable. As the communications industry continues to expand, we envisage

that agent-based techniques will be at the forefront of the research and development required to engineer the network infrastructure for the twenty-first century.

1.6 References

Bellifemine, F., Rimassa, G., and Poggi, A. (1999) JADE - A FIPA-compliant Agent Framework. *Proceedings of PAAM 1999, London.*

Finin, T. and Fritzson, R. (1994) KQML: A Language and Protocol for Knowledge and Information Exchange, in *Proceedings of 19th International Distributed Artificial Intelligence Workshop*, pp. 127–136, Seattle, USA.

FIPA (1997) FIPA 1997 Version 2.0 specifications. Available: http://www.fipa.org/spec/fipa97.html [18 May 2000].

Gabriel, R. (1996) *Patterns of Software: Tales from the Software Community*. Oxford University Press, Oxford.

Guarino, N. (1995) Formal Ontology, Conceptual Analysis and Knowledge Representation. *International Journal of Human-Computer Studies*, **5/6**, 625–640.

Huhns, M.N. and Stephens, L.M. (1999) Multiagent Systems and Societies of Agents, in G. Weiss (Ed.) *Multiagent Systems*, pp. 79–120, MIT Press, Cambridge, MA.

Jennings, N.R. & Wooldridge, M.R. (1998) *Agent Technology Foundations, Applications and Markets*. Springer-Verlag.

McDermid, J. (1998) The cost of COTS. *IEEE Computer*, **31:6**, pp. 46 52.

Poslad, S., Buckle, P. and Hadingham, R.G. (2000) The FIPA-OS agent platform: Open Source for Open Standards, *Proceedings of PAAM 2000*, Manchester, UK, pp. 355 368.

Searle, J.R. (1969) *Speech Acts*. Cambridge University Press Cambridge, UK.

Weiss, G. (Ed.) (1999) *Multiagent Systems*. MIT Press, Cambridge, MA.

Wooldridge, M.R. (1999) Intelligent Agents, in G. Weiss (Ed.) *Multiagent Systems*, pp. 27–77, MIT Press, Cambridge, MA.