Poslad S, Pitt J, Mamdani A, Hadingham R, Buckle P. Agent-oriented middleware for integrating customer network services. In: Software Agents for Future Communication Systems, Hayzelden A, Bigham J Eds., Springer-Verlag, ISBN 3-540-65578-6, 1999, pp 221-242.

Agent-oriented middleware for integrating customer network services

Stefan Poslad, Jeremy Pitt, Abe Mamdani

Imperial College of Science, Technology and Medicine, Intelligent Communication Systems, Department of Electrical and Electronic Engineering, Exhibition Road, London, SW7 2BZ, UK.
Email {s.poslad,j.pitt,e.mamdani}@ic.ac.uk; Web: http://www-ics.ee.ic.ac.uk/

Robert Hadingham, Philip Buckle

Nortel Networks, Smart Network Technology, London Road, Harlow, Essex, CM17 9NA.
Email {pbuckle,rgh}@nortelnetworks.com; Web: http://www.nortelnetworks.com

1 Introduction

There are a variety of forces causing previously disparate networks and services to be unified or integrated, including cost-reduction, ease of configuration and ease of maintenance. Communication networks of the future will consist of a wide variety of inter-linked computer networks, giving customers access to a huge range of potentially competing network services. From a customer perspective, the unification and integration of heterogeneous networks means that a service user (or value added service provider) can access any service from anywhere at any time. Examples of how services are coalescing include a diverse range of services from unified messaging services (incorporating FAX, voice-mail, voice and email) to the diversification of services offered by supermarkets.

This vision of ubiquitous service access, requires open and dynamic relationships between service providers and service users. For example, in the Telecom domain Messerschmitt (1996) has identified a shift away from vertical integration, where a single provider offers a dedicated infrastructure embracing networks, services and applications, to horizontal integration. This horizontal integration is characterized by 3 layers: a set of integrated network and transport mechanisms, a set of generic support services (middleware) including operating system services, and a diverse set of applications and services (Figure 1). The separation of applications from transport and common functions encourages application diversity; any service provider can supply these applications.

Poslad S, Pitt J, Mamdani A, Hadingham R, Buckle P. Agent-oriented middleware for integrating customer network services. In: Software Agents for Future Communication Systems, Hayzelden A, Bigham J Eds., Springer-Verlag, ISBN 3-540-65578-6, 1999, pp 221-242.

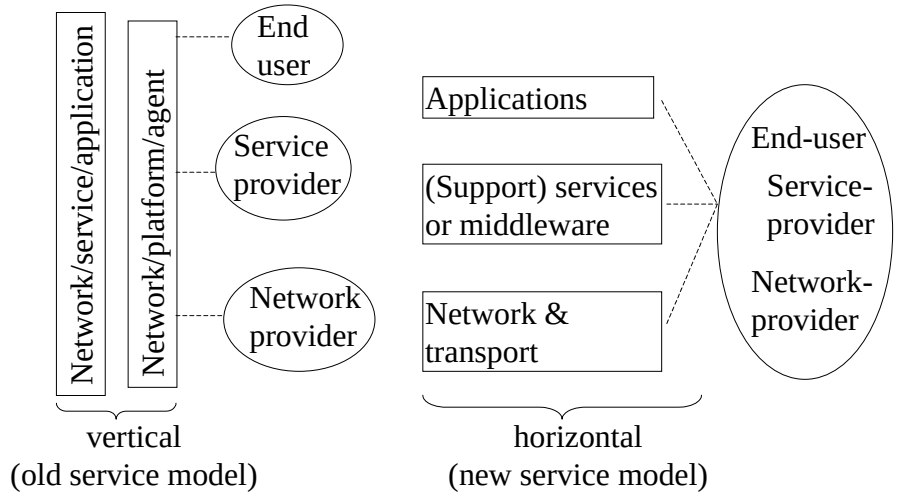

Fig. 1. Network services are moving from vertically integrated to horizontally integrated service model.

Note that there is no exact correspondence between this model and the ISO Open Systems Interconnection Reference Model as Figure 1 is a logical service model not a network protocol model. Note also until the advent and use of agent standards (including de facto standards) agent services may also tend to follow a vertical service model.

In order to support transparent application access and interoperability, horizontal integration is required at both the network layer and the service layer. To access services from any network requires heterogeneous networks to be inter-linked. To combine heterogeneous services requires service interoperability. In addition, it may also be potentially advantageous, to introduce loose coupling, vertically, between levels. For example, Directory Enabled Networks [URL 1] and software agents (O'Brien and Nicol, 1998) can enable lower level entities in the network level to make use of higher-level end-user and application profiles to allocate network bandwidth.

The roles between different users and providers of applications, services and networks will become more dynamic and inter-linked in future communication systems (see Figure 1). Traditionally, networks and services were usually managed by the network and service provider on behalf of the customer and offered as a single virtual network service but customers themselves are becoming more empowered to integrate and manage their own services. For example, a network customer in the guise of a service provider to its own customers, may wish to link inquiries to a sales database so that customers associated with the highest sales are always dealt with first.

The configuration and management of networks, primarily from a network provider perspective, using software agents is covered in detail elsewhere in this book. This chapter focuses on discussing how software agents can work within the existing and within future communication infrastructures to better support the management of networks and services in the end-user and Value-Added Reseller (VAR) domain.

The rest of this chapter is structured as follows. The next (second) section provides a basic introduction to service management issues for users and value added resellers. It describes the benefits of using agent-oriented middleware and reviews some existing agent platforms with reference to the structure of the agent support system. The third section discusses general design issues for agent-oriented middleware. The ability to

support both agent and non-agent software services in a federated open environment is regarded as a key design requirement. The fourth section outlines the approach adopted by the CASBAh (Common Agent Service Brokering Architecture) project. Conclusions are presented in the final section.

2 Background: network and service level integration

2.1 Network level integration

The development and deployment of communication systems is under pressure from a number of different forces, including network proliferation, deregulation, computer-telephony integration, TV-Web integration and Internet technology. We expand on these forces below:

Network proliferation*: future networks will be, as the Internet is today, networks of networks, they will have different physical characteristics (fixed copper, optical fiber, radio, wireless, satellite), will have different transport mechanisms, protocols and quality of service parameters (bandwidth, delay and latency), and customers will be connected simultaneously from different access points.*

Deregulation: *different operators already control large parts of national and global networks comprising mobile phone networks, fixed networks and satellite connections. Diminished regulation implies increased competition, and vendors will have to be responsive to differing customer requirements or face losing market share.*

Computer-telephony-television integration: *the drive here is to integrate content from traditionally non-integrated networks of computing, telephony and television to generate new value-added services such as directory-enabled call-answering and the use of computers to provide supplementary information to support more interactive selection and knowledge-base linked viewing of audio-visual broadcasts.*

Internet technology: *advances in networks, protocols and security have made possible access to a diverse set of applications video-conferencing (perhaps delivered via MBone the Internet Multicast back bone) and electronic commerce (supported by protocols, such as those for SSL the Secure Socket Layer, which offer encryption and authentication). Customers will expect to buy services over the Internet and deliver IT related services over enterprise-wide Intranets.*

To summarize, a variety of forces are driving diverse sets of networks to become interlinked. Technically, there are several different approaches to achieve this. Firstly, heterogeneous networks can be unified. For example, non-IP networks can be enhanced to carry IP packets perhaps by adding a ‘shim’ over another network layer protocol or by the use of tunneling (Rose, 1990). This is called IP everywhere. Secondly, heterogeneous networks can be homogenized by simply replacing each network to be one of the same type, i.e., it was envisaged initially that ATM would pervade the whole network from desktop to WAN (ATM everywhere). Thirdly, heterogeneous networks can be linked at discrete points in the network such as bridges and gateways where different network protocols can be translated (Rose, 1990).

There are several ways in which such a vision of unification and integration of distributed services can come about in the market place. Although it is anticipated that the Internet will continue to dominate as the backbone, some types of network may

become subservient to it perhaps losing their identity (unification), whilst others may retain their identity and become integrated with it. Further, new infrastructures such as satellite entertainment broadcast networks may play an important role as providers of communication and data network services and will need to be integrated with this internet.

An example of the IP-everywhere vision is Webtone [URL 2]. Webtone is an initiative from Nortel Networks to build the Unified Networks of the future - secure, reliable and scalable carrier-class multimedia networks that will allow businesses to reap the benefits of electronic commerce and the new digital economy spawned by the Internet. Webtone aims to make the Internet as easy to access as the voice network and it will be fast, reliable, secure, and as much a part of our everyday lives as dial tone is today.

The amount of data traffic being carried on corporate and public carrier networks has surpassed the amount of voice traffic, and the fastest growing segment of data traffic is being transmitted over TCP/IP. A key enabler for Webtone networks will be the addition of quality of service (QoS) to today's IP networks. The IETF is instrumental in realizing the standards necessary to achieve end-to-end QoS, through initiatives such as Integrated Services [URL 3] and Differentiated Services [URL 4].

2.2 Service integration

In this sub-section, we first focus on why end-users and value-added resellers are becoming proactive in driving service integration. We then consider various technologies for integrating services, finally settling on the use of multi-agent systems as a powerful solution to the complexity of integrating heterogeneous services in an open-market place.

2.2.1 Client-centered service integration in an open market-place

User or client-centered service management prescribes that, users rather than providers integrate and manage their own services within the customer domain. Service users may wish to control this ‘in-house’ for a variety of reasons particularly if they are also themselves value-added service providers further down in the supply chain.

Service users may not wish to commit themselves to a single strategy in order to future-proof their system or to provide flexibility. For example, a different network to the actual application session network may be used for management and control.

Service integration often requires access to information, which users may wish to keep private from any other organization. Users may spot niche opportunities and wish to synthesize value-added services quickly whilst competition is weak. If users own the domain expertise, they may not find it easy to impart it to others to act on their behalf. For example, network users who play a service provider role may wish to link enquiries to the sales database so that customers associated with the highest sales are always dealt with first.

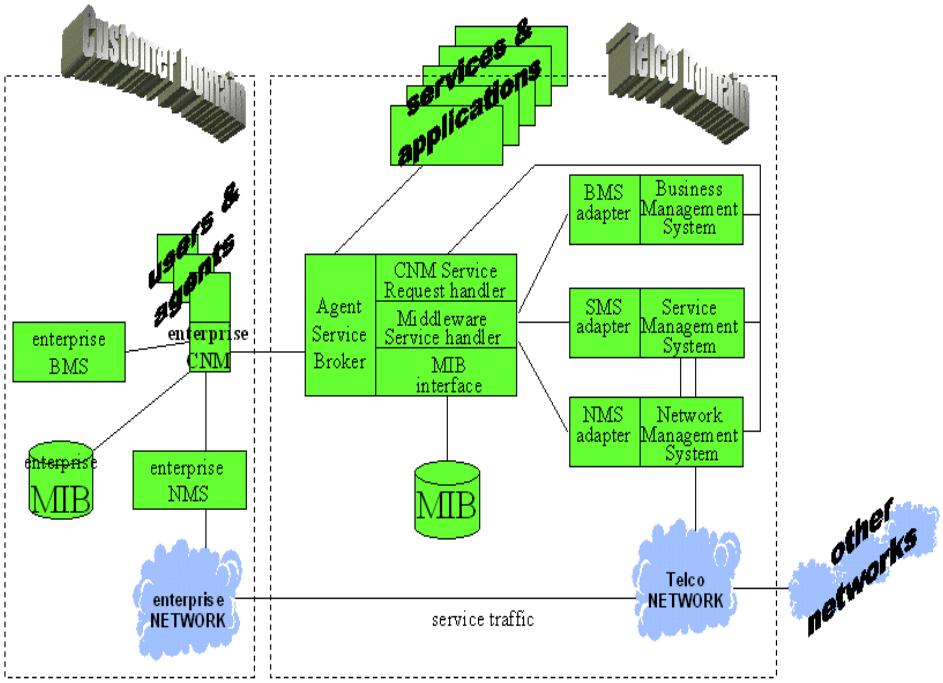

Fig. 2: use of an (agent) service broker within a service user–provider chain

In addition, service rationalization and divergence may occur in a phased evolution, passing from control by the provider to control by the user (figure 2). In the first phase, service interfaces will be coalesced on the user or consumer-side and connected to a service infrastructure such as middleware inside the customer domain. In the second phase, the unified interface will be expanded with the customer being able to use the interface to secure online reconfiguration of existing services and provisioning of new services. Furthermore, these services will not be just those offered by a single provider: predefined service interfaces will enable third party service providers to define, configure and supply services. Access to these services will be mediated by an "intelligent" intermediary deployed by the operator which will also be able to 'push' new services pro-actively to customers. In a third phase consumer systems may adapt and evolve, perhaps automatically, independently of a network or service provider further up the chain.

From evaluating this interaction in the value-added reseller, service user (client-centered) chain, we can identify several main requirements for service resellers and service users:

- to enable value-added resellers to be able to introduce new services quickly (this was one of the driving forces which drove Plain Old telephone Services (POTS) to be re-organized into Intelligent Network (IN) services);
- to integrate existing (legacy?) heterogeneous services;
- to package (broker) a selection of services into a virtual service to ease use.

The above requirements typify an open distributed service environment. Such an environment introduces the following complexity, which will also need to be handled by the service integration technology:

- *the environment is dynamic*: at any instant of time there are potentially many different ways for the environment to change. For example, most obviously,

the occurrence of any fault and the natural fluctuation of service requests. This also includes new services coming on-line, old services being withdrawn, and the behavior of federated service providers (promotions, strategic alliances, etc.);

- *system decision-making is open*: at any instant of time there are potentially many different actions or procedures the system can execute. For example, the negotiation of quality of service parameters and the establishment of permanent virtual circuits can be traded between multiple service and application providers;
- *successful operation is pareto optimal* with respect to the potentially many different objectives that the system is asked to achieve. The system's overall goal is to optimize revenue by delivering services within the negotiated quality of service parameters, without wasting bandwidth, and where appropriate, minimizing latency. This goal has to be achieved in the context of established contracts, existing service usage, and incoming service requests;
- *the actions that best achieve the objectives*: may sometimes be independent of the state of the system and dependent on the state of the environment. For example, service requests are satisfied depending on network state, network load, current operating conditions, etc., in addition to the computational state of the service integration system;
- *the environment can only be sensed* locally: the system can only detect local network conditions and is not aware of operating conditions in other networks and switches. Without near real-time information, the service integration system cannot build a coherent, end-to-end view of the network, its service distribution, and performance. Therefore the system has to operate in the face of uncertain and incomplete information;
- *the rate of computation is bounded*: the rate at which the system can compute its next action is bounded by the rate at which the environment evolves. For example, changes in network load, service availability and service requirements (e.g. burstiness in variable bit rate connections) can occur during the computation of a response to a service request or its management.

2.3 Use of middleware to aid service integration

A primary means of achieving service integration is via the use of middleware. For the purposes of this chapter, we define *middleware* to be any software entity that is interposed between a client (service user) and a server provider, a peer and other peers (one or more users or providers), or an application and a platform. This entity provides some kind of additional services that we would intuitively associate with a human middleman, broker, arbiter, facilitator, and so on, although obviously realized in computational terminology of types and services. Further these services are organized to be readily accessible by any user, yet to be independent of specific end users and services - this organization has arisen through a process of heterogeneous service generalization and rationalization.

Traditionally (pre-1980s), telecommunication systems used specialized vertically integrated applications, platforms and networks. In the 1980s, middleware specialized for telecommunication voice services such as the ‘Intelligent’ Network or IN model, was introduced to decouple applications from the network. It also

organized services so that more complex services could be synthesized from a set of simple service building blocks. The most recently proposed Telecom model, TINA uses an enhanced version of standard object-oriented middleware, CORBA (see below) to support management and configuration services.

There exists a variety of commercial-off-the-shelf object-oriented middleware including CORBA (OMG, 1992)], Java [URL 5], Microsoft's COM/COM+ Common Object Model based software [URL 6] and the Web [URL 7].

The main role of middleware in Common Object Request Broker Architecture or CORBA is as a 'forwarding broker': it relays each request to a server, retrieves the response and relays it back to the client. Although, the CORBA middleware appears to insulate service users from service providers, optimally CORBA clients must have detailed a priori knowledge, albeit in a language independent manner, of their service invocation interfaces before the clients are commissioned. There are several components of Java which can be regarded as performing some sort of middleware role, these include: the Java Virtual Machine is a type of middleware, which provides generic service invocation, insulated from a particular hardware platform; Java Beans (and similarly COM) standardizes the packaging of services to support run-time binding. The Web provides a generic invocation service using a standard address format, URL or Uniform Resource Locator but there is more important middleware functionality implicitly offered by the Web model of distributed computing. URLs are analogous to object references, the common currency of, for example, CORBA systems. But, to access the server, information, or program referenced by the URL, first the client has to know the URL. The way that a client discovers what URLs are available is via a *search engine*. A search engine therefore actually provides location services in the manner of a trader or yellow-page directory service.

Whilst such object-oriented middleware could provide many of the facilities to support the client-centered service integration, their use is non ideal when used within a federated open distributed environment. Object-oriented middleware can require too much detailed knowledge to be acquired before service invocation can occur. Some object-oriented middleware is difficult to extend at run-time, not very re-configurable or adaptive, is unable to broker services intelligently within an open service environment and unable to allow entities at different levels of abstraction and at different levels of the network-service-user model to interoperate effectively.

In addition, the way object-oriented parts interact may naturally be very complex to reconfigure and maintain, depending on how the objects and object interaction is designed. The use of meta-objects, which contain information, to constrain the temporal object interaction sequence, to vet replacements and extensions of parts of the system, and to co-ordinate the actions between different autonomous object aggregates becomes essential. Traditionally, these meta-objects exist in the form of an object-oriented application framework but these are not geared towards dynamic service re-configuration at run-time or towards intelligent re-configuration in response to unusual events.

2.4 Software Agents to aid service integration (enhance network access)

Before we can describe how software agents can aid service integration, we first define what a software agent is. We then consider why we should use agents and show how agents can aid service access integration and network access. We end this

section by analyzing how best to provide support for agents by reviewing some existing agent systems.

A software agent can be regarded as an embedded process that encapsulates some notion of state, communicates with other processes by message passing, and can perceive/affect its environment by means of sensors/actuators (where both environment and sensing can be implemented in software). The agent process will also require an ontology, grounded in the application domain; an AI component, for reasoning about the domain; and a more or less anthropomorphic shell, for any user interaction.

Therefore, agents are software entities that may, collectively, be physically distributed but, individually, are certainly logically distinct and are unlikely to operate in isolation. Furthermore, in order to function at all, each agent needs to be bound to one or more hardware entities (platforms).

Software agents offer many potential advantages over object-oriented middleware for the provision of enhanced service integration. These include: support for much more flexible means of interaction such as farming out parts of problems to others via collaborative problem solving; using inherently richer semantics for discovering the capabilities and requirements of service providers; use of intelligence to reason about alternatives, incomplete knowledge and past experience. Finally, agents can enable tasks to be specified in more user effective ways: they can be designed to understand the user's particular domain terminology, incomplete task specifications and to suggest optimal or sub-optimal service matches. Agents can use knowledge representation and (logical) reasoning to provide a basis for service ontologies, so that an agent can reason about task descriptions based on a logical formalization. This provides a more sophisticated mechanism for service allocation and provision that transcends the syntactic matching used in conventional object brokering (Puder, 1994).

There are many ways in which software agents can support service integration and access. A predominant way in which agents are currently being introduced for this purpose is by introducing service proxies: they replace existing service provider software. For example to provide a travel service, several different service agents such as a Travel broker, Flight service provider, tourist office broker and accommodation broker could be introduced to provide the travel service.

Another technique is to reuse or wrap existing (legacy?) services, for example Foundation for Intelligent Physical Agents, FIPA, 97 architecture specification part 3 [URL 8], rather than use an agent substitute. Here, the agent needs an invocation interface to the non-agent service and to be able to translate agent requests to non-agent service requests and vice versa for the results of the requests.

Collaborative agents can pervade the whole of the service architecture not just the application layer of the service model (Figure 1), for example, they can enable the network to be application and people aware - thus supporting the Directory Enabled Network vision [URL 1]. This is achieved by commissioning various agents such as user agents, network-access agents and service agents, which are able to share information and negotiate. E.g., a home-net parent user agent can attain privileges to change which web-sites can be accessed, whereas a home-net child user agent is unable to. When a parent rather than a child is logged on, he or she may have prioritized use of the home-net bandwidth. Conversely a multiple agent system can

enable the application to be network aware. For example, an application agent who finds a recipient uncontactable by phone, may decide to enlist another service agent's help to email or mail (re-route) information possibly in a modified format.

As such multi-agent systems are complex, they tend to use some sort of agent-oriented middleware to act as an intermediary between agents and to provide a pool of generic services for communal use.

2.4.1 Review of some agent systems from a middleware perspective.

A selection of agent platforms have been compared with respect to types of middleware services which they offer: these include: [message] transport, directory, management, agent communication, brokerage and mobility. Some of the services have been broken down into sub-services, e.g., the agent communication service is broken down into an organizational sub-service and a co-ordination sub-service. The FIPA Agent Platform (AP) is used as a reference model and is discussed later.

Table 1. middleware services offered by some multi-agent platforms. ✓ indicates a middleware service is supported; × indicates it is not; ? indicates that it is not clear whether or not such a service is supported from the published article.

	Agent Systems							
	FIPA AP	Info-Sleuth	JATLite	KAoS	KIMSAC[1]	OAA	OMG AF	ZEUS
Middleware services								
Transport services	✓	✓	✓	✓	✓	✓	✓	✓
Forwarding	✓	✓	✓	✓	✓	✓	✓	✓
Routing	✓	✓	✓	?	?	?	✓	?
Mail-box	×	?	?	?	?	?	✓	?
Directory services	✓	✓	✓	✓	?	✓	✓	✓
White-pages	✓	✓	?	✓		✓	✓	✓
Yellow-pages	✓	?	?	✓		✓	✓	✓
Management services	✓	?	?	?	?	?	✓	✓
Message storage	✓						✓	?
Agent life-cycles	?						✓	?
Agent comms. Service	✓	✓	×	✓	✓	✓	×	✓
Organization	?	?		✓	?	?		✓
Co-ordination	✓	?		?	?	?		✓
Brokerage services	×	×	×	×	×	×	×	×
Mobility Services	×	×	×	×	×	×	✓	?

InfoSleuth (Nodine and Unruh, 1998) provides middleware in terms of an agent shell which includes a white-page directory service, an autonomous composite component called the conversation layer to provide routing, message-forwarding and basic dialog management, and a broker agent component. The agent system is implemented in a Prolog like language.

JATLite (Java Agent Template Lite) system [URL 9] provides Java middleware libraries, called layers, for basic communication service, a combined routing and message forwarding autonomous component or 'active library' and an agent communications library. The libraries by can be substituted with alternatives. For example, the default basic communication library supports only TCP/IP transport not UDP/IP nor CORBA but it can be substituted by an alternative which supports these alternatives. Similarly, the agent communication library supports KQML (Finin, Labrou and Mayfield, 1997) by default but other alternatives can be supported. JATLite can be regarded as providing some parts of a basic agent infrastructure for developers.

KAoS (Knowledgeable Agent-oriented System) system (Bradshaw, Dutfield and Benoit. 1997) is designed to be independent of a particular communication service.

[1] Trial 1 system. There is a 2nd trial underway whose architecture and middleware is different but this has not yet been published.

Several types of communication service "have been investigated" such as OMG's CORBA, IBM's SOM, Microsoft's COM and Java socket model. All KAoS agents are derived from a generic agent template, which provides a basic communication mechanism. Several important Agents may play a persistent role but it is not clear whether this is implemented as middleware. Specialized middleware agents carry out other generic services such as a matchmaker (yellow pages), a domain manager (to keep track of ownership issues, white-page service) and proxy and mediation agents act as external interfaces to the agent platform.

KIMSAC (Charlton, Espinoza, Mamdani et al, 1997), Kiosk-based Integrated Multimedia Service Access for Citizens, EU ACTS project (030) middleware is oriented to project a single virtual service abstraction for the end-user, specifically to unify the interactive presentation of information from a suite of social welfare [service] services. In user trial 1, the middleware to support the single virtual service consisted of user agent, presentation manager and asset manager components. The presentation manager behaves as a user interface for the user agent, the user agent behaves as a personal assistant to the user. The asset manager is not an agent, but software which can aid each service agent to map and unify its information with other information into a generic set of visualization components. All agents communicate using KQML. The middleware is implemented in a variety of languages such as Java (Presentation Manager), Prolog (User Agent) and CLIPS (service agents). In addition, a CORBA based layer called the Agent Service Layer (ASL) is used to integrate non-agent software with the agent software. The agents themselves in the trial 1 system communicated via an underlying virtual memory or blackboard type system rather than using the CORBA layer.

OAA (Open Agent Architecture) system (Martin, Cheyer, Moran, 1997) middleware consists of an agent component called a facilitator, which provides yellow-page directory and persistence and co-ordination services. OAA also provides an agent library linked to each agent, which implements the agent communication service and supports interaction with the facilitator.

The OMG Agent Facility, AF, [URL 10] is a type of common facility, which forms part of CORBA. It seeks to essentially provide object wrappers or containers for objects to support agents. Agents are defined to be "small to medium grained objects, they can move, and they can start or stop their execution autonomously". The AF uses other parts of CORBA such as the ORB to provide the basic communication service. Agents are treated differently to (non-mobile) objects: CORBA does not define any agent services such as communication or brokerage service, this must be built on top of the CORBA. GMD Fokus among others have built AF compliant software, which they call Grasshopper, see chapter 14 (Breugst, Choy, Hagen, Höft, Magedanz, 1999). This can create, suspend, resume, remove agents, receive agents, list agents, list available services, get an agent state and get an agent reference. Hence the OMG AF offers white and yellow directory services and life-cycle management services for agents.

The ZEUS (Nwana, Ndumu, Lee, 1998) 'agent building tool-kit' middleware includes: agent components offering yellow and white page services and a set of template libraries linked to each agent to support agent communication, different co-ordination and organizational strategies and to support reasoning and inference. ZEUS is implemented in Java and uses an underlying TCP/IP communication service. ZEUS

is not just an agent system but also provides an environment, which includes visual tools for building, monitoring and configuring agents.

2.4.2 Summary and Trends for agent middleware

Several trends in developing agent platforms can be highlighted:

Extraction of parts of domain-dependent agent behavior into middleware to support the (Wooldridge and Jennings, 1995) strong notion of agency. For example, the ZEUS definition layer generalizes the agent inference engine. OAA defines meta-agents, which are similar to the CORBA common facilities – they are a kind of application-specific middleware, e.g., a notification agent can understand and optimize the use of different communication terminals such as fax, phone, email and can be contracted to supply these services to the service agents.

Use of de facto or COTS (commercial off the shelf) distributed object software middleware to implement the generic middleware functionality, e.g., use of CORBA by KIMSAC and KAoS.

Design and packaging of agent middleware into coarse-grained components, for use by third party and value-added application agent builders. For example, developers using JATLite may replace one or more of the default [layer] components with their own version, e.g., the default TCP transport can be replaced with a UDP transport. Third parties can also use JATLite as a basis of more complex agent system

The provision of higher-level tools and environments, in contrast to low-level application program interfaces, for building agents, e.g., ZEUS.

Support for mobility using facilities in de facto object-oriented middleware such as the OMG Agent Facility [URL 10] or using the capabilities of Web-browsers. There is some contention as to whether this is software or object (software which does not support agent properties such as social interaction and autonomy) mobility rather than agent mobility.

Current agent platforms are limited because they:

- use propriety internal middleware services such as directory services rather than use open software middleware service;
- are tightly-coupled to one particular service infrastructure rather than being able to adapt to several different service infrastructures;
- broker service in a propriety agent environment rather than an open agent environment (the advent of standards including de facto standards will help alleviate this problem);
- offer weak support to integrate non-agent software services with agent software services.

In addition, many current agent systems may be difficult to reuse by VAR designers because they:

- may be too complex and unstable nor sufficiently documented to be used by third-party developers;
- may not be available to third party developers;
- may not be customizable or extensible except in limited directions.

These issues of openness, integration of agent and non-agent services, reconfiguration by end-users and VARS are being investigated as part of the CASBAh project described later.

3 Design issues for agent-oriented middleware

3.1 The importance of standards

Standards, including de facto standards, enable value-added resellers to synthesize new services and to provide interoperability with existing services. To manage and access services from heterogeneous networks requires interoperability on several different levels: protocols, organizational roles, information types and on the types of information processing supported. Network protocols were the first entity to be standardized in the early 1980s, they later became part of the ITU Telecommunication Network Management (TMN) standards (Sidor, 1998). At the same time, ISO also specified the organizational and co-ordination roles: agent entities collect and forward information to the manager entity, which processes the information. Although, the CORBA interface standardization can also be used to define the structure of the data exchanged, this is service dependent whereas the TNM information is specified independently of how it is processed. Next the information exchanged was standardized, e.g., TNM Management Information Bases or MIBs. Finally, the information processing functions, c.f. CORBA service interfaces were standardized e.g., the IN Service Information Base or SIB. This enables processing functions to be defined for reuse, and exchanged: synthesis can then be used to generate new services quickly.

The agent community standardized the agent communication protocol first, e.g., KQML (Finin, Labrou and Mayfield, 1997), FIPA Agent Communication Language (ACL) (O'Brien and Nicol, 1998); next the syntax of the information exchanged was standardized, e.g., Knowledge Information Format (KIF) [URL 11], c.f. TNM MIB. In parallel with these, the agent community is attempting to standardize the interpretation or meaning of the information, e.g., FIPA management ontology (this relates to how the information is processed).

The highly interactive nature of multi-agent systems points to the need for consensus on agent interfaces in order to support interoperability between different agent systems. The completion and adoption of such a standard is a prerequisite to the widespread commercialization and successful exploitation of intelligent agent technology. At the time of writing FIPA has around 50 member organizations (commercial and academic) committed to achieving the required consensus for interoperability.

The FIPA standards are ratified by common census in contrast to de facto agent standards such as KQML, they provide:

- a commonly agreed means by which agents can communicate with each other so they can exchange information, negotiate for services, or delegate tasks;
- facilities whereby agents can locate each other (i.e. directory facilities) ;
- an environment which is secure and trusted where agents can operate and exchange confidential messages;

- a unique way of identifying other agents (i.e. globally unique names) ;
- a means of accessing non-agent and legacy systems, if necessary;
- a means of interacting with users;
- a means of migrating from one platform to another, if necessary.

The FIPA agent standard will bring the commercial world a step closer to true software components, the benefits of this will include increased re-use, together with ease of upgrade. Early adopters of new technology tend to be wary where there is no commonly agreed standard and which lacks the support of a large consortium of companies; an agent standard will provide added confidence to potential adopters of this technology. Finally, the standardization process shifts the emphasis from longer-term research issues to the practicalities of realizing commercial agent systems. FIPA allows for focused collaboration (of both industrial and academic organizations) in addressing the key challenges facing commercial agent developers as they turn agent technology into products.

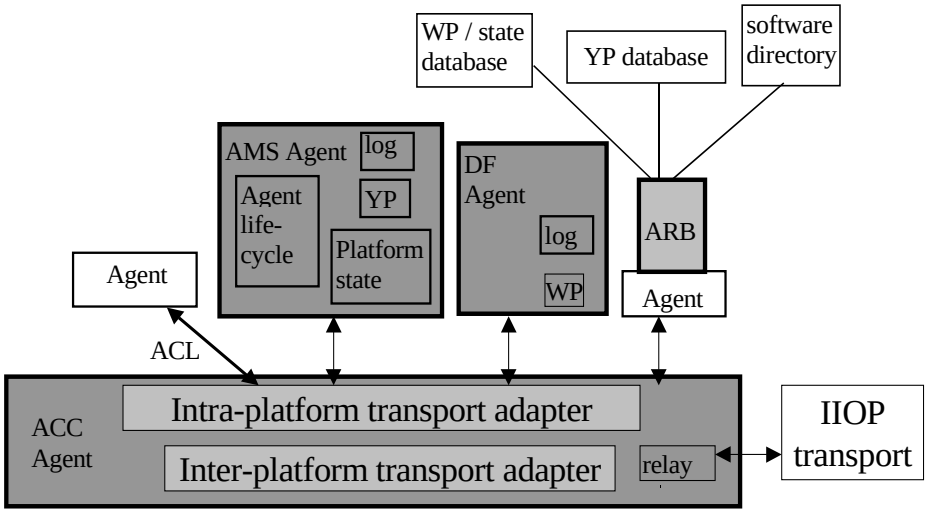

Fig. 3: FIPA agent platform reference model consists of 4 agents: Agent Management System (AMS) Agent; Directory Facilitator (DF) Agent; Agent Communication Channel (ACC) and an agent playing a role as an Agent Resource Broker (ARB) to interface to non-agent software. YP is a yellow-page directory service, WP is a white-page directory service. The shaded areas indicate the boundary of the AMS.

FIPA specifies a reference architecture (Figure 3) for multi-agent systems, it does not specify a platform for the architecture. There are several projects, such as the EU ACTS projects FACTS [URL 12] and MARINER [URL 13] reported to be in the process of building agent systems, which conform to the reference architecture. The FIPA reference architecture defines agent middleware in terms of an agent platform (AP) and an agent communication language (ACL). The agent platform comprises four different middleware agents: agent management system, agent communication channel, directory facilitator and an Agent Resource Broker. The Agent Management System (AMS) is an agent, which manages the life cycle of agents on the platform and provides a “white pages” directory service. The Directory Facilitator (DF) is an agent, which provides a “yellow pages” directory service for the agents. The Agent Communication Channel (ACC) is an agent which uses information provided by the Agent Management System to forward messages between agents within the platform and to agents resident on other platforms [using IIOP]. The ARB Agent is an agent,

which provides the Agent Resource Broker (ARB) service – this can be thought of as a facilitator for non-agent software services. Each agent communicates using a common agent communication service [language], which defines both the context for interpreting the content and the structure of the content.

3.2 Basic architectural designs

There are two orthogonal directions for discussing architectures to support agents: fat vs. thin middleware and static vs. dynamic middleware.

Fat middleware (thin agent) maximizes the abstraction of common functionality of agents, ‘delegating’ these functions to a separate component, the middleware. The FIPA agent-platform (Figure 3) is an example of fat middleware.

Thin middleware (fat agent): maximizes the autonomy of each agent and minimizes the common functionality supplied by the middleware.

In the 'fat middleware' scenario, agents would be bound to work only within that agent platform, the middleware provides the agent environment, whereas for thin middleware, the agent controls its own environment.

N.B. the middleware may appear to be a separate component in the design [logical] model, e.g., OMG CORBA, but then it may be contained (statically linked) in each agent [process] in the implementation.

Both fat and thin middleware fix their functionality at build-time. In contrast, a component oriented architecture defines a repository of components, which perform these common functions and provide design-tools to allow these components to be dynamically combined in different configurations such as fat middleware or thin middleware. Components may even migrate between the agent and the middleware as needed.

There are several central issues here: the nature of the relationship between agents, middleware and components; the level of granularity of use or abstraction of the component and how components are managed. These are in turn affected by how a component is itself defined. Whilst it is commonly agreed that components act as replacement units, there are two major perspectives of the term 'replacement unit'. One perspective views a components as a COTS commodity, the other views a component as a unit of design independent of any concern for the component reuse (Brown and Wallnau, 1998).

The COTS commodity view is related to the level of granularity, agents may be the component or it may itself be composed of components. If the agent is composed of components, the types of components could correspond to the generic services offered by the middleware (Table 1) such as transport, management and brokerage. Different transport services could then be added as needed as could the brokerage service etc. These agent service components would be 'manufactured' by the middleware.

The component as a unit of design perspective differs from the COTS perspective, principally by isolating the application (business) logic and separating it from any implementation. For example the application component may be specified in term of a specification language or script and then interpreted and executed on a particular platform. In addition, to being loosely coupled to any component technology such as Microsoft's ActiveX or Sun's Java Beans, such specifications are in a much more

accessible form for meta design-tools such as inference engines and analyzers to manipulate.

Paradoxically because components can be so easily modified and replaced, this may also be their greatest weakness from a management perspective. Software components that are modified in some small way become increasingly harder to track, for re-use purposes, similar to documents without a version control system. Unfortunately the use of components is generally ad hoc and lacks engineering discipline. Current state-of-the-art component systems lack complete models, formal underpinning, and scalable tools. These issues are only partially being addressed by emerging standards, such as Microsoft's ActiveX and Sun's Java Beans.

3.3 The degree to which the AOM should be agentised

A system which principally stores and forwards messages does not itself need to analyze or process the messages, this can equally be done by software rather than by an agent. There maybe advantages in using the simpler sequential logic of the software, e.g., higher throughput. There are many existing robust message handling systems such as APRIL Communication System [URL 14] and Kannel [URL 15]. Generally, traditional middleware software plays more of the role of a name-server or match-maker rather than a broker: this is true of CORBA in its normal mode of operation even though it is called a ‘Broker. CORBA initially consisted of a very simple name server and a channel or message bus. With the addition of the trader facility, it became a matchmaker. Object-oriented middleware uses a co-ordination policy, which is often very simple e.g., organizational, i.e., the service requestor or client takes command of the co-ordination.

There may be justification for middleware to be able to evolve (emergent behavior) or switch from a simple-software mode operation into a more complex agent mode at run-time. For example, high throughput requirements may dictate prescribed (passive) routing being preferred to content-based (active) routing. If the behavior of the broker does change, it may need to publish such changes to its users.

As a rule, the more complicated the co-ordination, the more ‘agentised’ the broker needs to be. The agent broker could be regarded as an agent, which performs generic agent tasks. Note, if there are no other agents ‘out-there’, just homogeneous software services, there is less need for an agent-oriented middleware, a client-server model would suffice.

There is also a philosophical viewpoint as to whether or not the middleware is an agent. For example, basic middleware may simply store and forward agent messages but not process the actual agent message content. For this case, middleware can be philosophically regarded as not adhering to the so-called weaker notion of agency (Wooldridge and Jennings, 1997) - as there is no or little social interactivity. When the middleware needs to process the content of agent message, for example to do content-based routing, then the communication of the middleware with other agents requires greater social interaction and this would seem to fulfil the social ability requirement for a weak notion (Wooldridge and Jennings, 1995) of agency.

3.4 how to integrate the agents with an existing infrastructure

There are two levels at which heterogeneous software needs to be integrated within an agent service broker:

- external integration: integrating services implemented using a different software paradigm to the agent-oriented middleware
- internal integration: if the AOM is built using an existing underlying heterogeneous software architecture, the AOM must be mapped to it and vice versa.

Integrating heterogeneous software can be complex. The complexity increases if an underlying software layer uses a very different structure and interaction paradigm to a higher software, e.g., agent layer - here an architectural mis-match (Garlan, 1995) can occur. An example of a potential architectural mis-match is the use of an underlying procedural interaction layer , e.g., CORBA, to support a higher-level, message-passing or event interaction style more natural for agents

A procedural interaction style is adopted by de facto object-oriented and procedural middleware such as CORBA and COM. These use a so-called push-model for the interaction, the flow-of-control is asymmetrical and caller driven, requiring input or entry interfaces[2] to be defined in the called code. In addition, the interaction is synchronous by default. Procedural calls naturally lead to a direct invocation style, which introduces tight-coupling, and reduces autonomy between caller and called components. Even if data abstraction techniques are used, the caller must still have detailed knowledge of: the signatures of the set of procedures to be called; the order in which to call them; and a software 'handle' to the encapsulating software structure which contains the procedure. Procedure invocation often fixes the structure of the data or information exchanged prior to the service being built and limits the amount of information which can be exchanged (it is limited by the operating system memory stack size for example). Procedures are inherently for one-to-one interaction.

The message-passing or event interaction style differs from the procedural interaction style in several important ways: it defines both input and output interfaces, the generation of output events is sometimes referred as a push-model for interaction. As components can both call and be called, the control-of-flow is symmetrical. Message-passing implicitly triggers procedures or actions: the message sender does not need to have knowledge of how to invoke any action associated with the message receiver. In contrast to procedural interaction, the caller does require a specific software handle to the called software, although it does require the identity of a mailbox or port on the computer on which the called code resides. A port differs from a software handle in that it is not dependent on the characteristics of the software implementation of the software service but is more generic and associated with the network protocol. Message structure and message types can change at run-time and the size of the message is limited more by the free storage space. Messages can be sent to single, multiple parties or everybody depending on the network protocol and hardware support.

Assuming that two heterogeneous systems can interact, they may need to convert the structure of the information exchanged. There are two main approaches: convert near the message source or near the message receiver. For example the use of proxies or wrappers to make each service appear as a part different service is an example of the former. The use of indirection such as a gateways to convert each interaction into another type is perhaps an example of the latter. There is however, the added problem

[2] some object-oriented libraries do define specific output or exit call points but these are for abnormal control of flow (exception-handling)

of matching two different types of co-ordination. This is discussed in more detail below.

Procedural interaction can simulate asynchronous communication by making the invoked procedure very simple - it just transfers control to another thread and then returns. This is simplest if the receiver does not need to return any results to the sender. If the receiver does need to return results back to the receiver, it must return them by invoking a procedure in the sender at a later date or use some software signaling technique to indicate to the sender that the information is available. In this case, the sender must invoke a second procedure later to gather the results. Thus, simulating asynchronous communication using synchronous communication can involve considerable extra signaling overhead.

Because of the difficulty in using procedural style interaction to support message-oriented interaction, de facto middleware such as CORBA is evolving by adding message-oriented type interaction facilities such as a message service and an event service [URL 10]. These new services are however not yet ratified as part of the CORBA standard neither are they commercially available at this time.

4 Common Agent Service Brokering Architecture (CASBAh) project

The design and development of a platform for agent-oriented middleware is being undertaken as part of a project called the Common Agent Service Brokering Architecture or CASBAh (Pitt, Mamdani, Hadingham et al, 1997) - this is jointly funded by UK EPSRC and Nortel Networks. The aim of CASBAh is to specify and implement an agent–oriented architecture, an enhancement and higher-level abstraction of the traditional object-oriented architecture. It is applicable in any open computing environment which spans organizational boundaries, where there is some kind of networked infrastructure, and where service providers and service consumers are connected to it. Other examples include pre-competitive project management, manufacturing augmented by Electronic Data Interchange, and electricity generation and supply.

CASBAh consists of: firstly, meta-agents such as: *agent service brokers or ASBs*, which can match task requests to service descriptions and obtain favorable rates, etc. (analogous to a human broker). Secondly, it contains *agent-oriented middleware core* services to support agents. Thirdly, it has *non-agent adapters* to interface to *software* such as object-oriented middleware; to users via human computer interfaces; to *non-agent software services* to enable them to be enhanced to offer, request and negotiate over tasks and services. (Figure 4). We expand on each of these below.

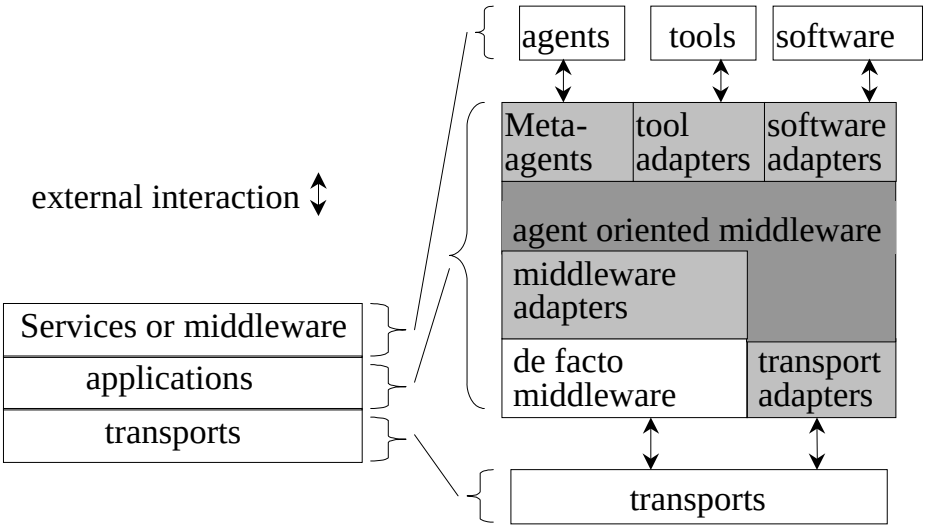

Fig. 4. The CASBAh Agent-oriented middleware, AOM, (the dark-shaded areas refer to the core AOM, the light-shaded areas to the AOM interfaces and the non-shaded areas to non-AOM components).

A key meta-agent component to enable the CASBAh system to operate within a federated service environment is the agent service broker or ASB. A preliminary requirements specification of the functionality of an ASB could include: task brokering within and between management information bases; negotiation mediation, delegation and co-operation techniques for the federation of multiple service providers, auction management, contract enforcement and service location (especially at an optimum rate).

Very little middleware functionality has addressed the problem of federating multiple service providers in an open distributed computing environment. However, in previous work, we developed a system to support advanced computer-supported co-operative work for project management in extended enterprises (Pitt, Anderton, Cunningham, 1996). This showed how the decentralization of resources coupled with standardized processing systems and networks could give better co-ordination, increased flexibility, and economy of scale. The normalization process coupled with the brokerage illustrated how distributed, autonomous, and heterogeneous organizations could collaborate using middleware, although this remains some way from fully supported federation services.

Given the open federated distributed computing environment that we postulate, it is the function of core Agent-Oriented Middleware to provide support services to both internal meta agents and external agents, inter alia:

- *message transport services*: routing, forwarding, mailbox;
- *directory services* to enable agents to find and communicate with each other, either directly or indirectly, in either client-server or peer-peer configurations;
- *management services* such as maintaining type descriptions, service instances, contract negotiation, ensuring service compliance, banking and billing, agent life-cycle management;
- *agent communication services*: context interpretation, knowledge or content interpretation, ontology, dialog management, translation;
- *brokerage services*, including third-party service provision and deployment, mapping task requests to service instances, and push-pull operations between clients and servers (to support the ASB);
- *federation services*, negotiation, delegation and cooperation techniques to forge temporary alliances of otherwise autonomous operators to form a federation of multiple service providers.

In addition, the AOM should also provide standard interface operations and tools to be used in managing, viewing, and verifying compliance with established protocols. Tools to support human interaction maybe particularly important as it is perhaps unlikely, that a sophisticated agent system can be commissioned in one phase. There is strong justification for agent middleware to heavily involve ‘human agents’ during their evolution, hence support for pervasive protocols such as SMTP and HTTP may be useful. It may also be likely that initially, agent brokers will not be sufficiently robust or have enough intelligence or expertise to operate without human intervention or support. Experience shows that closed-loop systems, such as those comprising multiple autonomous software brokers used previously in the London stock exchange can lead to spectacular system crashes. A further use of a human communication channel such as email is that it can act as a channel for system users and VARs to re-configure AOM components.

4.1 Relationship of CASBAh to other work

CASBAh potentially overlaps with many other agent projects which build agent platforms or middleware to support agents and which provide generalized support for agent negotiation. Specific agent based brokerage projects include the Agent based information Brokerage Service, ABROSE [URL 16]. There are several related electronic commerce projects, which specifically entail brokerage such as those under the ESPIRIT ACTS framework, however of these, only ABROSE, specifically uses agent-oriented brokerage. CASBAh differs from ABROSE in two main ways: firstly CASBAh is not specifically coupled to work solely with one existing architecture such as TINA. Secondly, CASBAh is investigating the brokerage of both agent and other software services.

It is out of scope here to describe in detail how the CASBAh architecture relates to the FIPA specifications but briefly, CASBAh firstly focuses on developing prototypes and secondly providing much richer support for brokerage and legacy service reuse within an open service environment. The latter issue is also not being adequately addressed in current software agent projects at this time.

CASBAh is similar to the JIAC (Wieczorek, Albayrak, 1998), Java Intelligent Agent Componentware, project in that it is building an open component architecture to support communication services. However, JIAC currently focuses on support for mobile agents whereas CASBAh focuses on support for multi-agent systems composed of mainly static agents.

4.2 System prototype and scenario

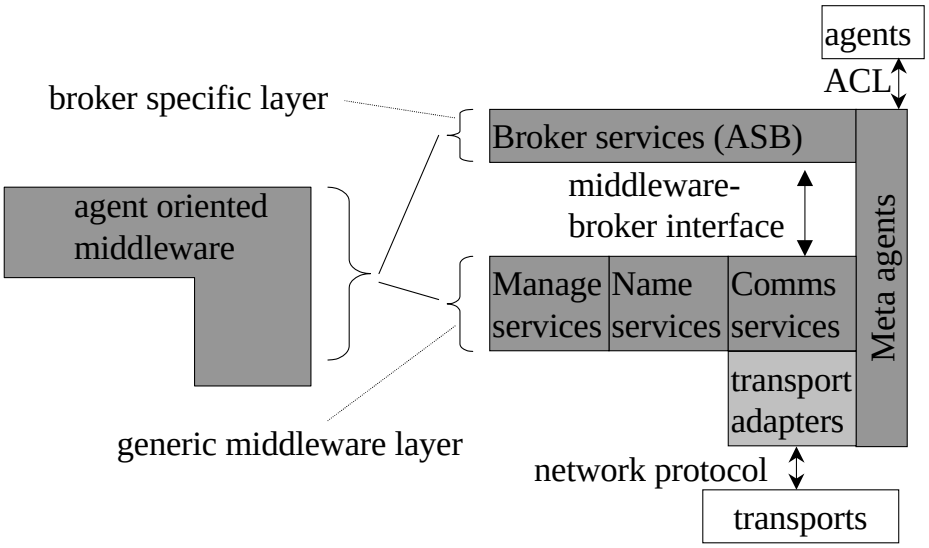

Fig. 5: expanded view of the agent-oriented middleware

A prototype of core parts of the CASBAh AOM has been implemented in Java. The AOM consists of:

- a communication service which supports the FIPA ACL, different kinds of agent communication transport such as SMTP and HTTP and which forwards messages between agents;
- a simple white-page name service.

The CASBAh AOM may or may not directly support services. In the advent that it doesn't directly support a service it may know how to indirectly support it. The AOM may also play a role as a back-up service supplier in certain cases when the primary service supplier fails – this is partly dependent on whether the service client software has the capability to connect to multiple service providers. To illustrate these points, we describe a case study of how the CASBAh AOM can be used to integrate email enabled agents and email software.

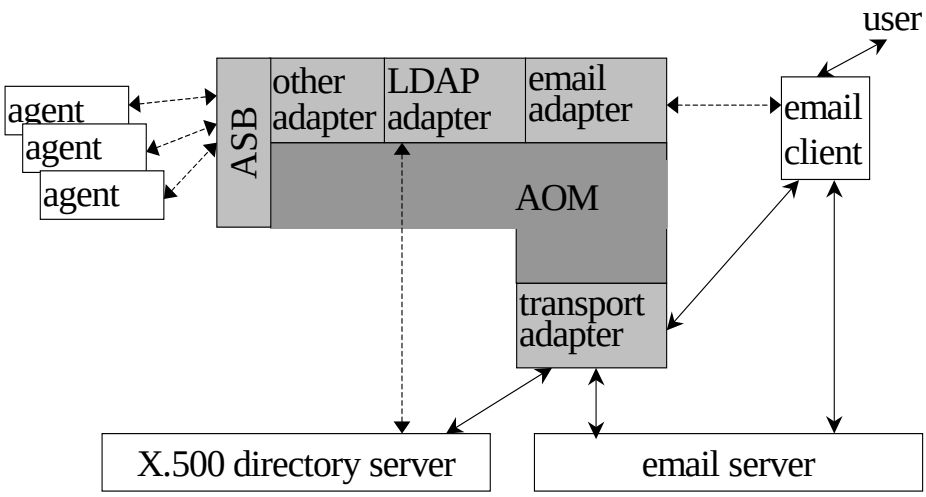

Fig. 6. Use of CASBAh to integrate email enabled agents and email software (dotted-arrow lines indicate virtual links, full-arrow lines indicate actual links).

An agent may wish to use email to contact and interact with end-users. The agent uses the FIPA ACL over a SMTP transport to send a message to the AOM. The AOM does not directly support a full SMTP service, nor does it need to know the email address of the users. Using its LDAP adapter, it issues a query to an enterprise X.500 directory service it knows of, to find the email address of a user, giving their real name as a key. The AOM constructs an appropriate SMTP envelope and contacts an SMTP email server it knows of to send the email message to the user. The user uses his/her usual email client to browse and issue email. The AOM is represented as a proxy-user to the SMTP service. The user replies to the email asking the AOM to forward the reply back to the specific agent.

This kind of interaction also allows support engineers or end-users to issue commands via email in order to reconfigure the AOM.

User's email browsers may also have the capability of connecting to more than one email service. In this case the AOM can be used to provide additional feedback (it may for example consult other services to suggest alternatives if the email fails) or as a secondary email service in case the standard primary email service fails.

5 Conclusions

This article has considered how multi-agent systems offer a timely solution to the complexity of interoperability and heterogeneity of distributed communication services. We have also reviewed a range of agent technologies and toolkits for

supplying solutions, and have focussed on the joint Imperial College / Nortel Networks project CASBAh as our contribution to these developments.

In particular, we would argue that agent-oriented middleware is applicable to any open distributed computing system, comprising a federation of suppliers that spans organizational boundaries and involves some networked infrastructure. Agent-mediation through service brokerage in such contexts offers advantages as an intermediary offering sophisticated brokerage and media-transport services to users who require (negotiated) access to a diversity of multimedia applications.

Arguably, though, the limit to success is the extent to which agent-oriented middleware enables integration, not just of computing and telephony, but of computing, telephony, and television within consumer devices. This may entail a radical paradigmatic shift of our current perception of these devices, but it may be necessary if the current saturation of the home PC market is to be opened up. Selling the device as an integrated television set-top box can increase computer sales. The software running inside will include an agent.

Acknowledgements

We acknowledge support for the CASBAh project from the UK Engineering and Physical Sciences Research Council (EPSRC), under grant GR/L34440. The CASBAh project is being undertaken in collaboration with Nortel Networks and their support is gratefully appreciated.

Abbreviations

AOM	Agent-Oriented Middleware
ASB	Agent Service Broker
CASBAh	Common Agent Service Brokering Architecture
COM	Common Object Model
CORBA	Common Object Request Broker architecture
FIPA	Foundation for Intelligent Physical Agents
HTTP	Hyper Text Transfer Protocol
ITU	International Telecommunications Union
KQML	Knowledge Query Meta Language
MBone	Multicast Backbone
OMG	Object Management Group
POTS	Plain Old Telephone System
SMTP	Simple Mail Transfer Protocol
TINA	Telecommunication Information Networking Architecture
TNM	Telecommunication Network Management
VAR	Value Added reseller
VPN	Virtual Private Network

Bibliography

Asatani, K. (1998) ‘Standardization on multimedia communications: Computer-telephony-Integration-related issues’. IEEE Communication Vol. 36. No. 7. 1998. pp.105-109.

Bradshaw, J.M., Dutfield, S., Benoit, P., et al. (1997) 'KAoS: towards an industrial strength open agent architecture'. In Software agents, Bradshaw, JM (ed.), MIT Press, 1997. pp.375-418.

Brown, A.W., Wallnau, K.C., (1998) 'The current state of CBSE'. IEEE Software. Vol. 15. No. 5. 1998. pp.37-46.

Charlton., P., Espinoza, F, Mamdani, E., Olsson, O., Pitt, J., Somers, F., Waern, A. (1997) 'An Open Agent Architecture supporting Multimedia Services on Public Information Kiosks'. Proc PAAM ’97, 1997. pp.445-466.

Poslad S, Pitt J, Mamdani A, Hadingham R, Buckle P. Agent-oriented middleware for integrating customer network services. In: Software Agents for Future Communication Systems, Hayzelden A, Bigham J Eds., Springer-Verlag, ISBN 3-540-65578-6, 1999, pp 221-242.

Finin, T., Labrou, Y., Mayfield, J. 'KQML as an agent communication language'. In Software agents, Bradshaw JM (ed.), MIT Press, 1997. pp.291-316.

Garlan, D., Allen, R., Ockerbloom, J. (1995) 'Architectural mismatch: why reuse is hard'. IEEE Software. Vol. 12; No. 6. 1995. pp.7-26.

Martin, D.l., Cheyer, A.J., Moran, D.B. (1998) 'Building distributed software systems with the open agent architecture'. In Nwana, H.S., Ndumu, D.T. (Eds.), Proc. PAAM '98, 1998. pp.355-376.

Messerschmitt, D. (1996) 'The Future of Computer-Telecommunications Integration'. IEEE Communications Magazine Vol. 34 No. 4. 1996. pp.66-69.

Nodine, M.H., Unruh, A. (1998) 'Facilitating open communication in agent systems: the InfoSleuth Infrastructure'. In Singh, M.P., Rao, A., Wooldridge, M.J. (Eds.), Intelligent Agents IV- Proc. ATAL '97, 1998. pp.281-296.

Nwana H.S., Ndumu, D.T., Lee, L.C. (1998) 'ZEUS an advanced tool-kit for engineering distributed multi-agent systems'. In Nwana, H.S., Ndumu, D.T. (Eds.), Proc. PAAM '98, 1998. pp.377-392.

O'Brien, P.D., Nicol, R.C. (1998) 'FIPA towards a standard for software agent's'. BT Tech. J 1998. Vol. 16, No. 3. pp.60-68.

OMG (1992). 'The Common Object Request Broker Architecture and Specification'. OMG Document Number 91.12.1 (Revision 1.1), Wiley & Sons, 1992.

Pitt, J., Anderton, M., Cunningham, J. (1996) 'Normalized Interactions between Autonomous Agents: A Case Study in Inter-Organizational Project Management. Computer-Supported Cooperative Work'. Vol. 5. 1996. pp.201-222.

Pitt, J., Mamdani, E., Hadingham, R., Tunnicliffe, A. (1997) 'Agent-oriented middleware for telecommunications network and service management'. AI for network management systems, 1997. IEE Digest No. 97/094.

Puder, A. (1994) 'A Declarative Extensions of {IDL}-based Type Definitions with Open Distributed Environments'. Int. Conf. Object-Oriented Information Systems. Springer-Verlag, 1994.

Rose, M.T. (1990) 'The Open book: a practical perspective on OSI'. Prentice Hall. 1990. ISBN 0136430163.

Sidor D.J. (1998) 'TNM standards: satisfying today's needs while preparing for tomorrow. IEEE Communications Magazine. Vol. 36. No. 4. pp.54-64.

Thompson, J.P. (1998) 'Web-based enterprise management architecture'. IEEE Communications 1998. Vol. 36 No. 7 pp.105-109.

Wieczorek, D., Albayrak, S. (1998) 'Open scalable Architecture for Telecommunication Applications'. In: Albayrak S., Garijo F.J. (Eds.) Intelligent agents for Telecommunication Applications. Vol. 1437. Springer-Verlag, Berlin Heidelberg New York. pp 233-249.

Wooldridge, M., Jennings, N. (1995) 'Intelligent agents, theory and practice'. Knowledge Eng. Rev. 1995. Vol. 10 No. 2 pp.115-152.

Universal Resource Locator (URL) Addresses

[URL 1] Directory Enabled Networks, http://www.cisco.com/

[URL 2] Nortel Networks Webtone project, http://www.nortelnetworks.com

[URL 3] IETF Integrated Services, http://www.ietf.org/html.charters/intserv-charter.html

[URL 4] IETF Differentiated Services, http://www.ietf.org/html.charters/ diffserv-charter.html

[URL 5] Java home-page, http://www.javasoft.com

[URL 6] Microsoft COM Home page, http://www.microsoft.com/com/default.asp

[URL 7] World Wide Web Home page, http://www.w3.org

[URL 8] Federation of Intelligent Physical Agents home page, http://www.fipa.org.

[URL 9] Java Agent Template Lite (JATLite) home page, http://java.stanford.edu/java_agent.html

[URL 10] CORBA home page, http://www.omg.org.

[URL 11] Knowledge Interchange Format Specification, http //logic.stanford.edu/kif/specification.html

[URL 12] FIPA Agent Communication Technologies and Services (FACTS) ACTS project home page, http://www.labs.bt.com/profsoc/facts.

[URL 13] Multi-Agent aRchitecture for distributed IN load control and ovERload protection, MARINER ACTS project home page, http://www.teltec.dcu.ie/mariner.

[URL 14] McCabe F, Clark K. April: Agent Process Interaction Language. http://www-lp.doc.ic.ac.uk/~klc/april1.html

[URL 15] Kannel Home Page, http://www.cs.helsinki.fi/research/kannel, 1998.

[URL 16] ACTS project Agent based information BROkerage SErvice (ABROSE) project home-page, http://b5www.berkom.de/abrose/.