APM

POSEIDON HOUSE • CASTLE PARK • CAMBRIDGE CB3 0RD UNITED KINGDOM
+44 1223 515010 • Fax +44 1223 359779 • Email: apm@ansa.co.uk • URL: http://www.ansa.co.uk

FlexiNet

Java Distribution and Deployment

Matthew Faupel

Abstract

By virtue of having an easily transportable, architecture-neutral representation, Java has simplified the task of writing distributed and distributable applications. Nevertheless, to date the exploitation of this possibility has mostly been limited to writing “applets”. This paper examines the architectures that have been proposed for more complex distributed and distributable computing systems that use Java and the technologies being developed to support these systems. It then identifies niches within these architectures that still require further research and development.

APM.2038.02	**Approved** **Technical Report**	09 October 1997

Distribution:
Supersedes:
Superseded by:

Table of Contents

1 Introduction

The Java language, its execution environment, and the suite of core APIs and classes being built for it together provide facilities that recommend it for use in implementing distributed systems. Its ability for the same code to be run on different platforms, its safety features and its support, through the class loader system, for dynamically incorporating new code makes it particularly suitable for systems whose behaviour and configuration are expected to change over time.

Nevertheless, to date there has been little exploitation of these qualities beyond the relatively trivial use of "applets" to spice up web page content. This is due partly to the relatively short time since the language was introduced in a stable form and partly to the lack of infrastructure components to support more complex distributed applications. This lack is being addressed by industrial vendors who are developing architectures for distributed systems based on or incorporating Java, and providing components within those architectures.

This paper summarises the architectures and technologies related to distributed computing in Java that have been announced so far, concentrating on offerings from commercial organisations. It concludes with an analysis of the state of the market and an examination of the issues and components that still require further research and development. It has been prepared as part of the ANSA "Flexinet" project, which is investigating the issues surrounding wide-scale deployment of component-based distributed systems. It will be updated as further relevant industry offerings are identified.

Being in part a survey of current commercial offerings, this paper makes reference to a number of trademarks. The ownership of all such trademarks by their respective organisations is hereby acknowledged.

2 Java Deployment Architectures

2.1 Overview

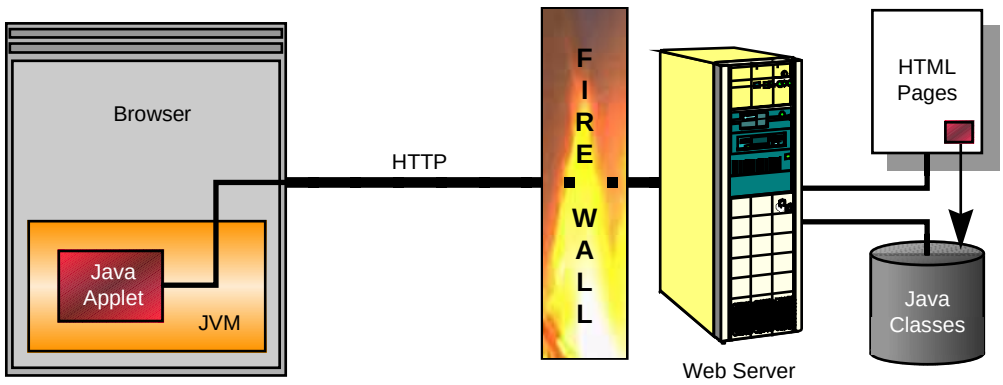
An architecture is a conformance framework within which a solution is designed. It defines mandatory and optional components, specifies their interactions to some degree and provides rules for building solutions that conform to the architecture. To date few architectures have been developed that present a coherent and well thought out framework for the partitioning and deployment of distributed applications. Most appear to be post-hoc rationalisations of existing or future product ideas; a notable exception is the OMG's Object Management Architecture.

This chapter highlights a selection of the more prominent architectures that have been put forward, with the focus being on commercially supported frameworks for employing Java in a distributed fashion.

2.2 Java Applets on the Web

This is the architecture most commonly used at present for distributed systems using Java. It is presented for comparison with the subsequent architectures in this chapter.

Figure 0.1. Java applet architecture



Briefly, the browser downloads an HTML page from a web server which contains a reference to a Java class that represents the applet. This causes the applet class to be fetched by the browser and executed by its Java Virtual Machine; this in turn will fetch from the server any further classes that the main applet class makes use of but which are not already available to the browser.

The Java applet is generally a single self-contained entity; it can communicate back on the original web server, but otherwise it has no other interactions with anything either on its own host or on other machines elsewhere. This is a limitation imposed by the current JVM applet security policy [Sun97b].

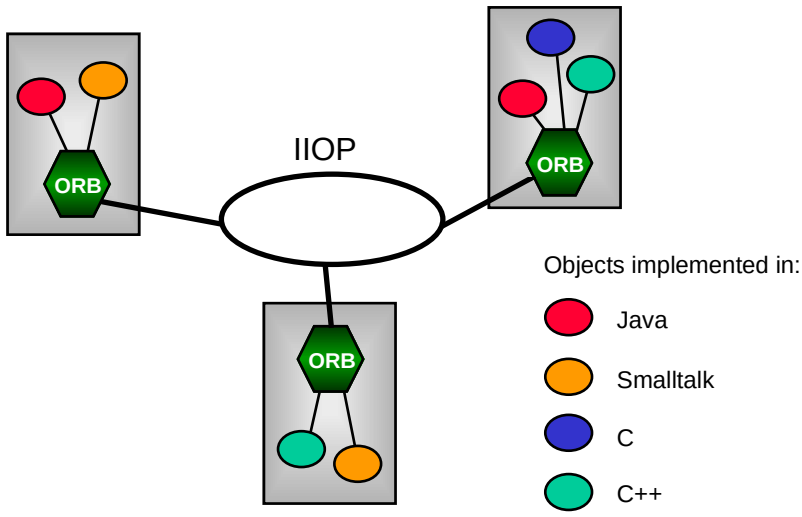
The firewall is optional but usual if the server is accessible from outside of its own administrative region. Firewall policies commonly limit allowable incoming connections to the server, and connecting to arbitrary ports is generally forbidden. This results in any communication back from the applet usually having to be done using HTTP, possibly aided by CGI scripts at the server side. With the advent of JDK 1.1 [Sun97c], this communication can be done using Java RMI [Sun97a] to invoke methods on Java objects running on the web server.

RMI can tunnel over HTTP if required, transparent to the applet. This is done using the following algorithm: attempt to contact host using a direct connection to the remote RMI server; if this fails, attempt to contact the host on the same port, but wrapping the RMI invocation in an HTTP POST request; if that fails, send the same POST request to port 80 on the remote host. This strategy should enable requests to pass through firewalls at either end which limit traffic to HTTP only, or more strictly to HTTP on a well-known port only. At the host end, HTTP requests received directly are automatically unpacked by RMI's default server socket implementation, while requests to port 80 are forwarded to the RMI server by a CGI script supplied with the RMI package. This does rather defeat the object of having the firewall in the first place though!

2.3 Java within the OMA

OMG's Object Management Architecture (OMA) is an architecture for object-oriented distributed computing that has been under development since 1989 [OMG97a]. At the heart of the architecture is the concept of Object Request Brokers (ORBs) that provide location transparency for the objects. The specification of these ORBs and their associated services is known as the Common ORB Architecture (CORBA) [OMG97b]. It was designed from the outset to be language neutral, hence fitting Java into it is a natural step.

Figure 0.2. Example CORBA system



The interfaces of CORBA objects are defined in a neutral Interface Definition Language (IDL). CORBA objects can written in any language for which a mapping has been defined from IDL to that language and which has an ORB that supports that language. The IDL to Java language mapping was approved by the OMG in July 1997 [OMG97c], and there are already several Java ORBs on the market (e.g. Visigenic's VisiBroker, Sun's Joe). The OMG has also issued RFPs for mapping from Java to IDL [OMG97d] and for supporting pass-by-value semantics for object parameters in operations [OMG96]. These together would then allow a distributed application to be written entirely in Java without need for separate IDL definitions of the objects involved.

The OMA allows ORBs to intercommunicate using any mutually understood protocol, however it requires all ORBs to support the Internet Inter-ORB Protocol (IIOP), which in practice is often the only protocol provided. This presents a problem when firewalls are introduced into the picture as IIOP allocates TCP/IP ports on the fly for inter-object communication, rather than using a single well-known port; firewall configurations commonly disallow arbitrary port usage in this manner, both in passing out of the client domain and passing into the server domain.

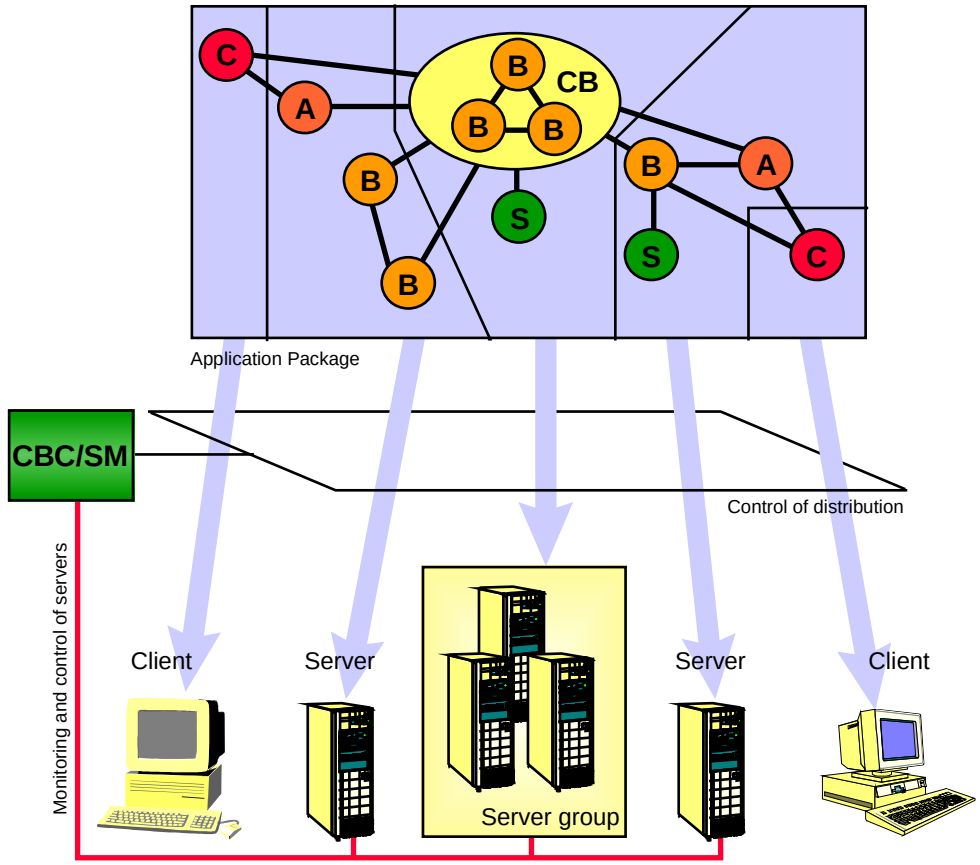
Two solutions have been developed to solve this problem: first, HTTP tunnelling, i.e. the client wraps IIOP messages as HTTP POST requests to be unwrapped by a server at the far end that supports this, and secondly IIOP-aware firewalls, which understand the IIOP protocol well enough to be able to filter messages and act as a proxy for objects that want to be visible outside the firewall. The latter approach is superior in performance, flexibility and security, but does require the client domain to permit clients to open a connection to a non-standard port, which is sometimes disallowed, hence leaving tunnelling as the only viable alternative. Visigenic's VisiBroker ORB [Visigenic97] supports HTTP tunnelling, in combination with their IIOP-GateKeeper at the server end, as does Sun's Joe and NEO [Sun97d]. Iona's Wonderwall is the only current example of an IIOP-aware firewall. It also supports tunnelling in combination with their OrbixWeb client [Iona97].

At present, the OMA provides standards that cover ORB functionality and interoperability, mapping of IDL to specific languages, and supporting services such as naming, trading, transactions and security. It is however purely an architecture for supporting the interworking of already distributed objects; it doesn't standardise how those objects are distributed in the first place, nor does it place any constraints on how the objects are organised to make up applications. Steps are being taken in this direction through the issuing of RFPs, e.g. for standardised components [OMG97e], and an agent framework [OMG95], but as yet no update to the architecture has been presented that links these into a coherent whole.

2.4 IBM's Component Broker

IBM have produced a development, execution and management framework based on the CORBA model for distributed computing called Component Broker [IBM97a]. This was released in beta form in May 1997 and is due for formal release in September 1997.

Figure 0.3. IBM's Component Broker Architecture



The development part of Component Broker supports the modelling of an enterprise as a collection of Business (B) and Composed Business (CB) objects, which interact with clients (C) supported by Application (A) objects, and hold persistent state in State (S) objects. Having developed this model, there is then support for implementing it as a set of CORBA objects (i.e. implementable in any supported language, including Java), with client views being provided via Java applets or ActiveX technology, the whole being known as an Application Package. A full set of CORBA services and facilities (e.g. naming, security, transactions) are provided for use by the applications, as are facilities for helping to interface with legacy systems.

The execution and management facets of Component Broker are handled by an execution environment called Component Broker Connector (CBC). This consists of a central management system, CBC Systems Management (CBC/SM), and a runtime component installed on all hosts within the system (CBC/SM Agent) which deals with the deployment, monitoring and control functions on the host.

An administrative model is held centrally by the CBC/SM, which defines the allocation of objects to servers and server groups and shows management boundaries. Policy decisions, e.g. for caching, replication etc., are made based on this model. Installation, removal and version control of application packages on servers and clients are also handled centrally, as is start-up and shutdown of the servers and applications. Finally, performance and problem monitoring is carried out by the CBC/SM.

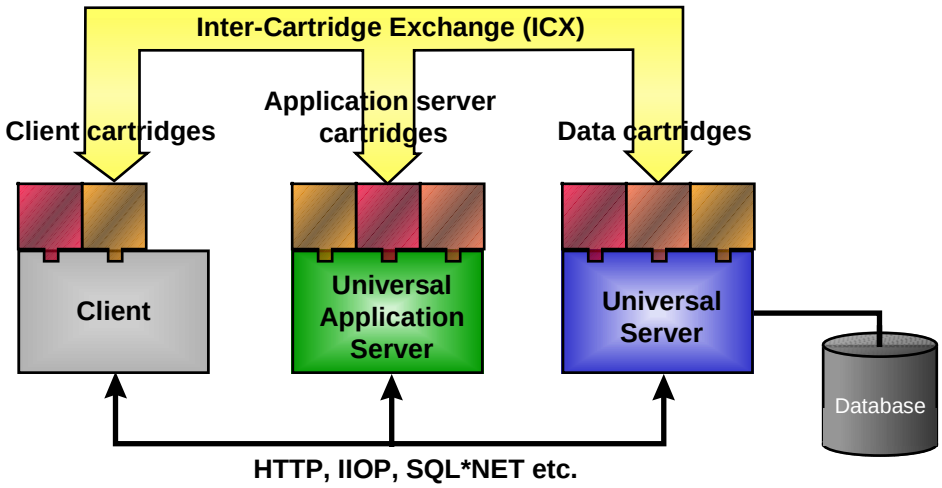
Once an application package has been installed by the CBC/SM, objects are instantiated on demand when a client requires them. They are partitioned based on management

policy and access requirements, with each partition under control of an application adapter. These control object access to resources including mapping to databases, legacy systems, and so on. A transaction manager is provided that concentrates client IIOP requests and directs them to the appropriate application adapter, and can (as directed by the centrally defined policy) dynamically create and destroy application adapters for load-balancing purposes.

2.5 Oracle NCA

The Oracle Network Computing Architecture (NCA) is Oracle's strategy for drawing together a number of disparate strands in distributed computing (CORBA, the web, client/server databases etc.) and merging them into a coherent framework for developing distributed applications.

Figure 0.4. Oracle's Network Computing Architecture (NCA)



The architecture, as outlined in Oracle's white paper [Oracle96a] has three key items:

- clients, application servers (an extension of the web server concept) and database servers that are extensible through "pluggable" objects fitting standard interfaces
- cartridges, which are the "pluggable" objects that provide the extensions
- A software bus, called the Inter-Cartridge Exchange (ICX) over which the cartridges can communicate with each other.

The cartridges are managed objects, whose interfaces are defined using CORBA IDL, and which conform to a basic interface that allows them to be installed, activated and used (i.e. plugged in). The clients and servers communicate with each other using whatever protocols are appropriate (IIOP, HTTP, SQL*Net etc.); additionally, the cartridges can communicate with each other using the ICX service, which in turn makes use of IIOP and/or HTTP.

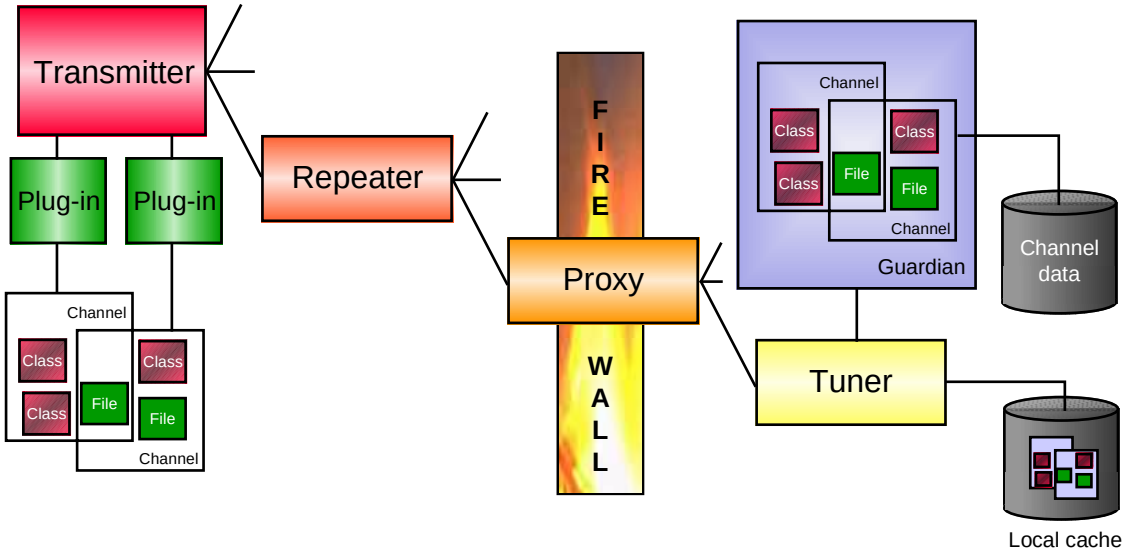
Various services are available to cartridges directly to provide support for administrative functions such as installation, activation, monitoring and security. Additional services are available via the ICX, e.g. support for transactions and access to local facilities such as the user interface of a client or the database on a database server.

At the time of writing, little concrete information exists on the NCA beyond the white paper. What documentation and software is available from Oracle does not seems to conform to this vision yet. There is no publicly available formal specification of any of the interfaces in IDL, and the published specification of the ICX software bus [Oracle96b] shows it to be a C-based interface for communicating using raw HTTP.

2.6 Marimba

Marimba supplies Castanet, a product that tackles the problem of ensuring that end users have the most up-to-date version of an application. Castanet handles Java code, HTML pages and data files.

Figure 0.5. Marimba's Castanet Architecture



Castanet follows a broadcast metaphor, with transmitters, channels, tuners and repeaters. Channels are the Java programs (or HTML pages) to be distributed, which are made available on transmitters (and possibly replicated on repeaters). Tuners on client machines tune in to a channel and (this is where the metaphor breaks down) based on a knowledge of individual user preferences collected by the tuner and relayed to the transmitter, the transmitter sends the incremental updates required for that tuner to have the latest version of the application appropriate to that tuner [Marimba97].

Channels are constructed from a number of files; these files may be shared across channels (e.g. class libraries), and the Castanet system can cope with different versions of the same file in different channels if required. When a tuner requests an update for a channel published by a transmitter it passes configuration information back to the transmitter, including optional arbitrary profile and logging information generated by the existing copy of the channel held locally by the tuner. Which files are returned to the tuner are controlled by a simple policy defined by the publisher; this can be overridden by the optional publisher-supplied plug-in that can interpret the profile information to customise the choice of files returned.

Repeaters are used to spread the load in the system. A tuner will initially connect to the main transmitter for a channel; this will then (transparent to the user) redirect the tuner to

use an appropriate repeater based on location and a round-robin allocation system. Proxies are used to optimise the crossing of firewalls in the system. Once downloaded by the tuner, channel Java applications are run within the Guardian. This is a Java Virtual Machine which imposes security restrictions on the channels similar to that imposed on applets by browsers, with the exception that channels are allowed limited access to a reserved area in the local file system for recording persistent information.

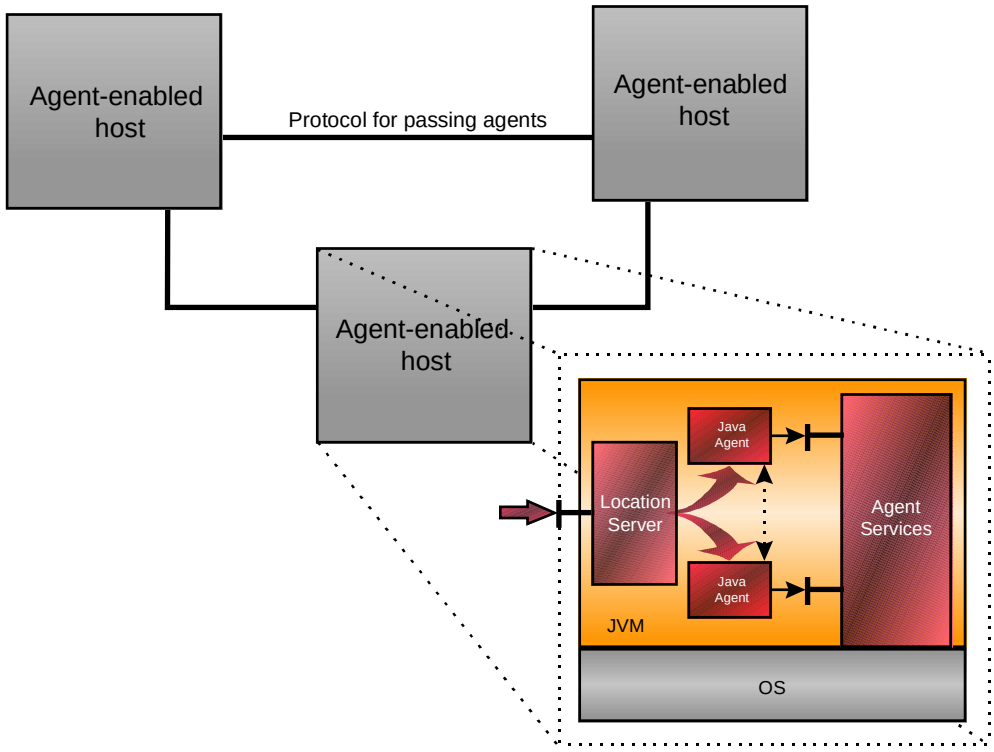
At present there are no restrictions imposed on tuners downloading channels (other than through limiting access to the transmitter itself) and no authentication is performed of the users of the tuners. These facilities could be added now in an ad-hoc manner through use of the plug-in system, and Marimba have promised some form of authentication (probably a password based system) for the future.

Marimba have recently released a new product, UpdateNow, which applies this deployment technology to applications written in languages other than Java. This can also be used for distributing Java applications that can be run outside of the Guardian sandbox (and thus have full access to system resources).

2.7 Mobile agent systems

A mobile agent is an object that can migrate itself between systems and that initiates this migration autonomously. A number of systems have been released that support the development and deployment of agents written in Java: Aglets [IBM97b], Odyssey [GM97], Concordia [ME97], Kafka [Fujitsu97], and Mole [SBH96]. Although they differ in details such as security policy, transport mechanism and class structure, they all have a broadly similar architecture for distribution and execution, as shown below.

Figure 0.6. Generic Java mobile agent architecture



An agent-enabled host has a server running within a Java Virtual Machine (JVM) that acts as a logical “location” for running agents. The world of agents is made aware of the location through a standard mechanism such as a trading service and then agents can be sent (or send themselves) for execution at the location using an appropriate protocol for passing the agent’s code and data. The location server is responsible for installing and executing the agent within the JVM, and possibly also for locating and fetching any subsidiary classes required by the agent. A well-defined set of agent services is usually available. These provide facilities such as inter-agent communication, restricted access to the host system to allow the agent to perform its task, and support for finding and moving to other locations.

Security is clearly an issue in agent systems - hosts are executing foreign code while clients are trusting objects that are owned by them and which carry their policy to be examined, executed and passed around by remote systems. A sound model of trust is required, as is supporting technology for identifying the ownership and rights of agents and to prevent tampering or information leakage. No such model has yet been developed (though Mitsubishi’s Concordia system appears to have taken steps in this direction), and security in the publicly available agent systems is of a similar nature to that in browsers at present, i.e. a highly restricted service is provided to agents originating from external systems.

A more detailed evaluation of various commercially available agent systems can be found in [APM97b].

3 Java Middleware Technologies

This chapter covers the available technology applicable to the distribution and deployment of Java applications.

3.1 Java RMI

Java Remote Method Invocation (RMI) is Sun's own Java remote procedure call system [Sun97a]. In contrast to CORBA, it aims to be a lightweight system tailored for use in Java only applications. It dispenses with some of the overheads that CORBA requires for language neutrality such as having separate IDL definitions of interfaces, and exploits certain Java-specific facilities to provide features unavailable at present in CORBA, e.g. pass-by-value semantics for objects (including downloading the class code for the objects if not already on the client) and dynamic download of stub code to clients.

The penalty the programmer pays in return for this increased simplicity and extended functionality is the loss of access to the range of additional facilities that the CORBA environment provides. RMI provides only a simple URL-based object naming service; there is no equivalent to CORBA's security and transaction services and the like.

The OMG are to standardise mapping from Java to IDL and support for pass-by-value semantics, so the differences between CORBA and RMI will be less clear-cut. Visigenic's VisiBroker/Java ORB development environment already allows interface definition in Java, pass-by-value semantics, class download and a naming service based on URLs. Sun are tackling this from the opposite direction by adding the possibility of using IIOP as the underlying transport for RMI instead of Sun's proprietary Java Remote Method Protocol (JRMP), along with defining the subset of the RMI interface that can be used if this option is chosen. As support for pass-by-value etc. in IIOP is provided, this subset will be widened.

Once the OMG standards are adopted, RMI's competitive advantage could be reduced to the fact that it is free with the JDK, and so will be likely to remain first choice for simple inter-Java communication (such as from an applet back to its host).

3.2 CORBA-based systems

There are now a wide range of CORBA-compliant ORBs and associated development systems that support the newly approved IDL to Java language mapping. A few systems that go further are as follows. IBM's ComponentBroker [IBM97a] adds distribution management functionality as outlined in the previous chapter. Sun's NEO [Sun97d] ships with an all-Java ORB called Joe that can be downloaded into a browser to enable it

to make use of CORBA objects. OrbixWeb from Iona [Iona97] provides similar facilities. Visigenic's VisiBroker/Java [Visigenic97] (the ORB from which is incorporated into Netscape's Navigator 4.0) adds the facilities mentioned above to allow the development of CORBA clients and servers entirely in Java, without use of IDL.

Cetus Links [Cetus97] provides a good summary of the ORBs currently available.

3.3 Agent systems

Aglets, Odyssey, Kafka, and Mole have been made available for public, non-commercial use, with commercial exploitation being subject to a separate licence. Mitsubishi's Concordia is available for evaluation through application to Mitsubishi. Guideware Corporation produces an SDK for writing Java agents [Guideware97]. None of these systems has yet progressed beyond the "beta" stage.

3.4 ObjectSpace Voyager

Voyager from ObjectSpace [ObjectSpace97] combines the features of ORB-based systems such as CORBA and RMI with those of agent systems. Voyager supports transparent method invocation on remote objects but adds the twist that any object enabled for remote access is also potentially mobile between Voyager systems. Agents are simply a special case of mobile objects that understand how to move themselves. Furthermore objects can be remotely instantiated on other Voyager systems, a technique which can be used to perform some of the tasks agents are often used for without the extra complexity of creating a local object and then transmitting it.

Various method invocation paradigms are supported (e.g. one-way, synchronous, deferred synchronous, multicast), which are all handled through the Messenger abstraction: the invocation details are handed to a Messenger which then delivers the invocation to the remote object(s) (and returns the result if appropriate) according to its particular policy. The use of a default Messenger can hide this complexity from the programmer if desired. The invocation protocol is a proprietary one carried over TCP/IP.

Voyager incorporates services for naming, events, remote object creation and persistence, and a system for grouping objects hierarchically to allow for scalability in message and event distribution. Security at present is limited to the installation of a Java security manager; there is no concept of object ownership and/or authorisation beyond the distinction of whether the objects originated locally or remotely, and no tamperproofing, secure transmission or the such like. It is implemented entirely in Java and is available free for commercial use in binary form. Source code can be acquired to allow extension of the core system e.g. to add new Messenger policies or to modify the security manager.

3.5 Marimba

Release 1.1 of Marimba's Castanet, the product behind the architecture outlined in the previous chapter, is now available. This version is based on JDK 1.1, which means that it is possible for the Castanet Channels to make use of Java RMI to communicate with the Castanet Transmitter. UpdateNow is also available.

3.6 Microsoft's Application Channels

With Internet Explorer 4.0 (due for full release in September 1997) Microsoft have extended their Internet Component Download system (introduced with Explorer 3.0) in a number of ways, including extensions to cope with Java, and a definition language for specifying information about "applications channels", i.e. executable content intended to evolve over time that is accessed using the "push" or "smart-pull" paradigm. The system allows the specification of the components of an application, their versions and the interdependencies between them, and a browser that is able to understand this specification can automatically download and install the application, as well as periodically check for changes, patches and so on. As such it has an overlap with Castanet as a method of deploying and maintaining Java software.

Application channels are specified in a Channel Definition Format (CDF) file [Microsoft97a], which uniquely names the application, gives its version and a description and points to a code base (or multiple code bases) via a URL. The CDF file is also responsible for specifying such things as update schedules. The code base referenced is intended to be an Open Software Description (OSD) file [Microsoft97b] (or an archive file containing an OSD file). The OSD points to the actual code required, allowing variants to be specified based on processor type, operating system etc., and shows dependencies on other components and minimum system requirements. The syntax of both CDF and OSD files conforms to the proposed Extensible Markup Language (XML) standard, and both have been submitted to the W3C as proposed standards.

Explorer 4 understands CDF and OSD files and uses the information in them to install components appropriately. The technology is clearly early in its development cycle though. The versions of the OSD and CDF standards submitted to the W3C differ from the latest available from Microsoft, which differ again from the descriptions of their use elsewhere in Microsoft's documentation. Furthermore, although the purposes of CDF and OSD files seem clear and distinct, there is actually much overlap between the two, with both files specifying version information, supported OSs etc. A second problem is that although the CDF and OSD formats are being pushed as open standards, they are only supported by the Explorer browser at present; that in turn only supports its own proprietary component installation mechanisms and Cabinet archive file format (.cab).

3.7 Lotus Notes

Lotus Notes [Lotus97] is a document-oriented database (amongst other things) that supports version control, user authentication, replication of database contents amongst a number of peer servers, and the download of contents to clients. The system now supports the storage of Java code within the database so providing an alternative solution to Castanet for the distribution and control of Java applications.

3.8 Java Beans

JavaBeans [Sun97e] is a component architecture for Java. It defines APIs and support for producing reusable software components, with the intention that these component can then be rapidly assembled into a complete application. As such, it is not strictly speaking a technology for distribution or deployment of Java, but it is likely to influence the

development of such technologies by defining a standard for distributable objects that can interwork remotely. It will almost certainly be referenced by the responses to the OMG's component architecture RFP [OMG97e], and its event model has already been used as the event model for Voyager (0). JavaBeans events were also used as the basis for APM's Object Monitor distributed event system [APM97a].

3.9 JECF

The Java Electronic Commerce Framework [Goldstein96] is Sun's offering for supporting electronic commerce applications written in Java. It has two new features that have a bearing on the issues of distribution and deployment of Java. First, it defines a "cassette" system for persistent downloadable code. This is similar to the applet idea, but provides for long-term information such as services classes or security keys to be downloaded, installed, and then retained after their usage is over rather than being discarded as applets are. This concept bears some resemblance to the "cartridges" of Oracle's Network Computing Architecture.

Secondly, it extends the Java security model to support the notion of principals and capabilities. This extension allows the definition of identifiable roles with particular rights. These rights can be granted on whatever level of granularity is required, down to per-object and per-method, and can be transferred. If this framework becomes widely accepted, it could provide the basis for a flexible system for authorisation of the control of deployment of distributable objects and access to services by those objects.

4 Analysis

4.1 The State of the Art

Static object-oriented distributed computing, i.e. where the location of an object is fixed once it has been instantiated, now has a reasonably well defined architecture in CORBA as well as a wide range of stable conformant technology. With Java there is the additional alternative of RMI. Other products are emerging for supporting the deployment and management of objects (e.g. ComponentBroker and Castanet) and, separately, for supporting objects that have mobility between systems (Aglets etc.).

Of the products that attempt to move beyond the static model, only Castanet is currently shipping in a production version; the rest are only available in alpha or beta versions at present. Plans for moving to production versions have been announced for Component Broker and Voyager.

4.2 The Missing Pieces

4.2.1 An Architecture for Dynamic Systems

Neither CORBA nor RMI currently address the issues of initial deployment of objects and their possible redeployment during their lifetime (i.e. mobility). Of those products that do tackle this area, most focus on solving particular narrow technical problems as opposed to defining a coherent architecture for dynamic distributed computing covering deployment, communication, mobility, management and security, and then placing themselves as a component within that architecture.

There has been no examination of the different applications of mobility with the aim of teasing out a common architecture (or explaining why such cannot exist). This has resulted in completely different systems being produced for solving problems that are all essentially to do with code mobility: application front-ends are deployed using applets and browsers, or perhaps Castanet; IBM's Component Broker can be used to deploy the elements of a distributed application; load balancing is handled by a separate transaction monitor, and mobility for local access to resources is handled by agent systems.

4.2.2 Modularity and Transparency

JavaBeans is an attempt to provide the Java world with a long-standing software engineering dream, namely the ability to build applications by plugging together

standardised off-the-shelf components. Deployment and mobility technology provides the means for getting those components to their users and automatically updating them with the latest versions. The two together give a vision of future applications being built from multiple components obtained from multiple vendors, with the ability to evolve as newer components become available with improved performance or added functionality.

Two important enabling features are still missing from most of the systems described above though: modularity and transparency. To realise this type of programming environment requires middleware that supports deployment and control at the level of individual components (modularity), and that does so without requiring every component to be aware of and in control of that deployment (transparency).

Castanet, the only deployment solution at present, only deals in complete applications (channels); there is no provision for mixing together components from separate transmitters. Conversely the various agent systems, which have the potential for deploying components from many sources fall down on transparency - only objects that have been written specifically to be agents can be mobile.

4.2.3 A Policy Framework for Dynamic Systems

An application can be viewed in terms of its functionality (a computational view), or in a more concrete fashion in terms of how it is deployed within a system (an engineering view). The mapping between the computational and engineering views is governed by policies on security, quality of service, resource usage and so on. These policies tend to change with time, e.g. through the emergence of new technology, changes in system configuration, or changes in business relationships, often more rapidly than the basic functionality of the application. The technology for dynamic deployment and mobility described above theoretically enables the configuration of applications to be modified in response to policy changes without the application itself being modified.

This would be difficult in practice though, first because of the lack of mobility transparency mentioned above and secondly because no framework yet exists for the specification and (automatic) interpretation of policy. Furthermore if this framework is to handle policy relevant to inter-domain interactions then an appropriate security model must be developed and supported by middleware.

4.2.4 A Legal and Commercial Framework

The world of tomorrow could be populated by distributed object-oriented applications which are built dynamically from components from multiple sources, and which are deployed (and re-deployed) automatically through the interpretation of users' policies and goals within current environmental conditions. However, for this scenario to be realised requires not only the gaps identified above to be filled in, but also a legal and commercial framework to be developed. For instance, the issue of charging for resource usage by foreign agents must be resolved, as must the legalities of the rights and obligations of mobile objects and their host systems. Can you murder an agent with impunity, misdirect it, or trap it? Can an agent "sign" an agreement on behalf of its owner in a legally meaningful manner? Dynamic distributed systems will develop without resolution of these issues, but for full exploitation of their possibilities over the Internet answers to these questions must be found.

5 Implications

5.1 The Flexinet Project

As shown in this analysis, the key failings in all current offerings are a lack of a coherent architecture for distributed computing that encompasses dynamic deployment, and the inability to separate out policy decisions from engineering. So, the clear goal of the Flexinet project is to develop such an architecture which allows separate specification of policy, and with the eventual aim of providing automatic implementation of that policy.

A number of key constraints on the architecture are also highlighted: first, there are many different solutions on offer, some complementary, some competing, but with no clear market leaders as yet. Hence, care must be taken to design the architecture with flexible abstractions from the ground up so as to be able to incorporate whatever emerge as the standard components, naming schemes, and protocols of the future. Secondly, the popularity and utility of agent systems has demonstrated that the architecture must take into account continuous (re-)deployment of objects; static one-off deployment before the application starts is not sufficient.

5.2 The Digitivity CAGE Product Line

Two facts relevant to the Digitivity CAGE [Digitivity97] product line emerge from this survey: first, that the number of ways of delivering Java to the executing host is increasing, and secondly, that the framework infrastructure within which the Enterprise CAGE is planned to operate (e.g. security and transaction services such as exist in the CORBA world) does not yet exist for the native Java and RMI world.

The multiplicity of Java delivery mechanisms affects the current CAGE. The probability here is that it will not be extended to attempt to cope with all of these mechanisms; instead the most popular will be identified through market research, and only those handled. Changes to handle code mobility as opposed to just applet download could support applications more efficiently that the current simple webserver caching. An interesting problem is how to deal with the CAGE's redeployment policies may result in components downloaded to the same logical location being deployed to different physical locations; this will have an effect on the operation of components that believe that they are actually colocated.

The Enterprise CAGE is aimed at long-term end-to-end secure business links. The lack of framework offers the possibility of developing complementary products such as a transaction service, or an RMI-aware firewall. The emphasis on long-term links means that arbitrary mobility is not required, though it might be useful for off-line working.

References

[APM97a]
Object Monitor; Schwiderski, S., Hayton, R.; APM Ltd., January 1997. APM.1938.01.

[APM97b]
Comparison of autonomous mobile agent technologies; Bursell, M., Ugai, T.; APM Ltd., June 1997. APM.1989.01.

[Cetus97]
Cetus Links: 5380 Links on Object Orientation / Object Request Brokers. http://www.rhein-neckar.de/~cetus/oo_object_request_brokers.html

[Digitivity97]
Secure Mobile Code Management; Herbert, A.; Digitivity, Inc., May 1997. http://www.digitivity.com/html/SecMCode.pdf

[Fujitsu97]
Kafka: Yet Another Multi-Agent Library for Java; Fujitsu Laboratories Ltd., June 1997. http://www.fujitsu.co.jp/hypertext/free/kafka/index.html

[GM97]
Introduction to the Odyssey API; General Magic Inc., May 1997. http://www.genmagic.com/agents/odysseyIntro.pdf

[Goldstein96]
The Gateway Security Model in the Java Electronic Commerce Framework; Goldstein, T; Sun Microsystems Laboratories/JavaSoft, November 1996. http://java.sun.com/products/commerce/jecf_gateway.ps

[Guideware97]
AgentWorks; Guideware Corporation, July 1997. http://www.guideware.com/site/agentworks.html

[IBM97a]
IBM Component Broker - Technical Overview; IBM Inc., May 1997. http://www.software.ibm.com/ad/cb/litp.htm

[IBM97b]
Aglets Library API - User's Guide; IBM Inc., July 1997. http://www.trl.ibm.co.jp/aglets/api/API_users_guide.html

[Iona97]
OrbixWeb for Java; Iona Technologies, July 1997. http://www.iona.com/Products/Orbix/OrbixWeb/index.html

[Lotus97]
Lotus Notes; Lotus Development Corporation, July 1997.
http://www2.lotus.com/notes.nsf

[Marimba97]
Marimba Castanet Datasheet; Marimba Inc., 1997.
http://www.marimba.com/datasheets/castanet-ds.pdf

[ME97]
Concordia: An Infrastructure for Collaborating Mobile Agents; Mitsubishi Electric ITA, February 1997.
http://www.meitca.com/HSL/Projects/Concordia/MobileAgentConf_for_web.pdf

[Microsoft97a]
Channel Definition Format (CDF); Microsoft Corporation, July 1997.
http://www.microsoft.com/standards/cdf.htm

[Microsoft97b]
Specification for the Open Software Description (OSD) Format; Microsoft Corporation & Marimba, Inc., August 1997.
http://www.microsoft.com/standards/osd/

[ObjectSpace97]
Voyager Core Package Technical Overview; ObjectSpace Inc., July 1997.
http://www.objectspace.com/Voyager/VoyagerTechOview.pdf

[OMG95]
Data Interchange Facility and Mobile Agent Facility RFP (ORBOS RFP11); Object Management Group Inc., November 1995, 1995/95-11-03.
ftp://ftp.omg.org/pub/docs/1995/95-11-03.pdf

[OMG96]
Objects-by-value RFP (ORBOS RFP2); Object Management Group Inc., December 1996, orbos/96-06-14.
ftp://ftp.omg.org/pub/docs/orbos/96-06-14.pdf

[OMG97a]
A Discussion of the Object Management Architecture; Object Management Group Inc., January 1997.
http://www.omg.org/library/oma/oma-all.pdf

[OMG97b]
CORBA 2.0/IIOP Specification; Object Management Group Inc., February 1997, formal/97-02-25.
ftp://ftp.omg.org/pub/docs/formal/97-02-25.pdf

[OMG97c]
IDL Java Mapping 1.0; Object Management Group Inc., March 1997, orbos/97-03-01.
ftp://ftp.omg.org/pub/docs/orbos/97-03-01.pdf

[OMG97d]
Java to IDL RFP (ORBOS RFP5); Object Management Group Inc., March 1997, orbos/97-03-08.
ftp://ftp.omg.org/pub/docs/orbos/97-03-08.pdf

[OMG97e]
CORBA Component Model RFP (ORBOS RFP8); Object Management Group Inc., June 1997, orbos/97-06-12.
ftp://ftp.omg.org/pub/docs/orbos/97-06-12.pdf

[Oracle96a]
Network Computing Architecture White Paper; Oracle Inc., September 1996.
http://www.oracle.com/nca/html/nca_wp.html

[Oracle96b]
Web Request Broker Programmer's Reference, Release 3.0 beta 3; Oracle Inc., September 1996.
http://www.olab.com/datasheets/wrbref.pdf

[SBH96]
Mole - A Java Based Mobile Agent System; Straßer, M., Baumann, J., Hohl, F.; University of Stuttgart, October 1996; ECOOP '96.
http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole/ECOOP96.ps.gz

[Sun97a]
Java Remote Method Invocation Specification; JavaSoft: Sun Microsystems Inc., February 1997.
http://java.sun.com/docs/jdk1.1/rmi-spec.pdf

[Sun97b]
Frequently Asked Questions - Applet Security; JavaSoft: Sun Microsystems Inc., June 1997.
http://java.sun.com/sfaq/index.html

[Sun97c]
JDK1.1.3 Documentation; JavaSoft: Sun Microsystems Inc., June 1997.
http://java.sun.com/products/jdk/1.1/docs/index.html

[Sun97d]
Solaris NEO; Sun Microsystems Inc., July 1997.
http://www.sun.com/solaris/neo/solaris_neo/index.html

[Sun97e]
JavaBeans: The only component architecture for Java; Sun Microsystems Inc., July 1997. http://java.sun.com/beans/

[Visigenic97]
Visigenic VisiBroker; Visigenic Software Inc., June 1997.
http://www.visigenic.com/prod/vbrok/vbjDS.html