Strategies for Content Migration on the World Wide Web

M.P.Evans[†], A.D.Phippen[‡], G.Mueller[†], S.M.Furnell[†], P.W.Sanders[†], P.L.Reynolds[†]

Network Research Group
[†] School of Electronic, Communication and Electrical Engineering, University of Plymouth, Plymouth, UK.
[‡] School of Computing, University of Plymouth, Plymouth, UK.

e-mail contact: Mike.Evans@jack.see.plym.ac.uk

ABSTRACT

The World Wide Web has experienced explosive growth as a content delivery mechanism, delivering hypertext files and static media content in a standardised way. However, this content has been unable to interact with other content, making the web a distribution system rather than a distributed system. This is changing, however, as distributed component architectures are being adapted to work with the web's architecture. This paper tracks the development of the web as a distributed platform, and highlights the potential to employ an often neglected feature of distributed computing: migration. The paper argues that all content on the web, be it static images or distributed components, should be free to migrate according to either the policy of the server, or the content itself. The requirements of such a content migration mechanism are described, and an overview of a new migration mechanism, currently being developed by the authors, is presented.

KEYWORDS

Distributed Computing, Migration, Resource Location

INTRODUCTION

The World Wide Web ('the web') is a platform for distributing software resources across the Internet, which are then presented as rich, consistent content by applications on the client (usually a browser). The three main standards which define the platform are:

- the Uniform Resource Locator (Berners-Lee, et al 1994);
- HyperText Transfer Protocol (Berners-Lee, et al, 1996);
- HyperText Markup Language (Berners-Lee, et al, 1995).

The Uniform Resource Locator (URL) is used to locate software resources; the HyperText Transfer Protocol (HTTP) is the protocol used to distribute the resources; and HyperText Markup Language (HTML) is used to present the information contained within the software resources in a consistent way across all computer platforms.

As such, today's web is a large distribution system. The software resource is a single, self-contained unit of data (usually a binary or text file), which the web can locate (using the URL) and distribute (using HTTP). It encodes content, which is presented on the client by applications according to the media type the content represents (e.g. images, video, etc.). Each media type must conform to its own universal standard, which is not part of the specification of the web itself, but which contributes to its ubiquity and openness. The content is decoded from the software resource by the browser or its own application (generally termed a 'viewer' or 'plug-in'), and is presented consistently across all platforms according to the layout and style specified by the HTML page. For example, the GIF (Graphics Interchange Format) standard, developed by CompuServe, is a standard format for compressing and encoding images. A GIF viewer is an application which works inline with the browser to interpret a GIF image file and display the image it contains. This GIF viewer essentially reads in a generic, platform-independent file (the software resource) which contains an encoding of the image, and converts the encoded data into content: platform-dependent information which can be displayed on the client's screen as the decoded image in a consistent way across all platforms according to the layout and style specified by the HTML. The same can also be said for other content formats (e.g. JPEG, MPEG, AVI, QuickTime), each of which encodes a specific media type according to the media's own defined standard. In fact, for any type of content to proliferate on the web, it must have its own platform-independent standard with its own platform-specific viewers generally available on every platform. To the web's distribution mechanism (i.e. the web servers and HTTP), everything is a generic software resource (see Figure 1). Only when the correct application receives it on the client does it become content.

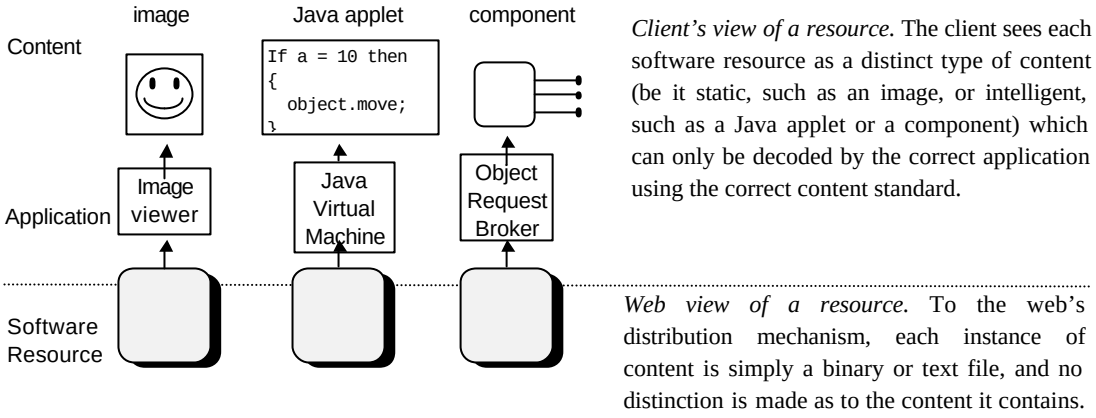

Figure 1: Relationship between content and the software resource

Static and Intelligent Content

Web content has traditionally consisted of static files without functionality, and without the ability to interact with other software resources. A GIF file, for example, contains the information required to display the image it encodes with a suitable viewer, but there is no computational intelligence contained within it; consequently, the user cannot interact with it (the use of 'image maps' within a browser, whereby a user can click on an image to navigate to another page, is controlled and formatted by the HTML in the web page, not the image file). Currently, then, the

web is a *distribution* system, not a distributed system. However, this is changing. As the web matures, its functionality is increasing, and, more importantly, the intelligence contained within the resources it is currently distributing is growing along with the web itself. To distinguish between resources which contain some form of static media content (such as an image), and resources which have some form of computational intelligence as part of their content (such as a Java applet), this paper will define the terms *static content* and *intelligent content*, respectively.

Intelligent content currently consists of small self-contained blocks of code which reside on a server as software resources, and are downloaded onto a client machine, where they are executed by a suitable application, usually inline with an HTML page. Java applets are an example of such content, as are Microsoft's ActiveX controls. This type of content is limited, however, by its self-contained nature: a Java applet, for example, cannot commun1icate with other Java applets on machines other than the server it originated from. In order to distribute the intelligence of a large scale application, the components of the application must be able to interact with each other across a distributed environment; to achieve this, a distributed component architecture must be employed.

DISTRIBUTED COMPONENTS

Component software develops on the potential of object-based software by constructing software from components which encapsulate functionality and data. This is similar to object orientation, but allows dynamic component interaction at runtime (known as 'runtime reuse'). This is achieved through the use of a *component architecture*, which is a defined platform with rules for interaction between components. Any component developed on top of this platform will be able to interact with any other component built on the same platform. Whilst a general component architecture enables components on the same computer to interact, distributed component architectures add to the functionality by enabling interaction across a distributed network environment. A client component can not only use the services of components on its host machine, but also any other machine which supports the distributed architecture. Components within such architectures are also termed distributed objects, primarily because the architecture itself is based on the object-oriented paradigm. Currently, the distributed component field is dominated by two major architectures: Microsoft's Distributed Component Object Model (DCOM) and the Object Management Group's Common Object Request Broker Architecture (CORBA).

DCOM is the distributed extension to Microsoft's COM (Component Object Model), and extends the basic functionality to incorporate transparent network distribution and network security mechanisms into the architecture. Through DCOM, ActiveX controls can interact with one another, and with other COM-based components, across a network.

CORBA is a complete architectural specification developed by the Object Management Group (OMG, 1995) which specifies both a component architecture, and component services. CORBA is entirely generic, defining platform-independent data-types, protocols for communication across platforms, and a number of platform-independent services which provide the components with a number of useful services such as security, transaction processing, and naming and location services for finding components across a distributed system. CORBA's functionality is

implemented through an Object Request Broker (ORB), which provides the transparencies required by the architecture.

Both architectures offer the developer similar features and similar benefits. They both provide a component distribution mechanism employing network and location transparency, marshalling mechanisms, etc., and both expose functionality through language-independent interfaces. They are reliable distributed platforms upon which large scale distributed applications can be built. Such distributed component systems are increasingly being incorporated into the web. Distributed components are becoming the next type of software resource to share server space with existing types of static and intelligent content. This allows the web to become a true distributed system, being able to provide distributed applications and services via a client's browser. Netscape, for example, has integrated CORBA functionality into its Communicator 4.0 browser, allowing it to interact with CORBA components on CORBA-enabled servers. Equally, Microsoft's Internet Explorer 4.0 browser is DCOM-enabled, allowing it to communicate with DCOM components on DCOM-enabled servers. In this way, the web is evolving into a complete distributed system, termed the 'Object Web' (Orfali et al, 1996) to reflect the object-based nature of the distributed architectures being employed.

CONTENT MIGRATION

An Overview of Migration in a Distributed System

One of the benefits of a distributed system is the ability of an application to be distributed across multiple hosts across a network, such that no one host has to execute the whole application on its own. With a fast enough network, this 'load balancing' functionality can greatly increase the efficiency and performance of the application in a way which is entirely transparent to the client machine. However, the drawback to this distributed paradigm is the static nature of the location of each component. Once a component has been installed on a host, it cannot easily be moved to another host. Thus, should the host's, or its network's, performance degrade in any way, access to the component will be affected. Invocations on the component's interfaces will be slowed down, which in turn will impact on the performance of the application as a whole. The component can be manually relocated to a different host, but this is time-consuming. Most distributed applications comprise many components, and it would be impractical to manually redistribute them all whenever necessary.

As such, various automatic component relocation mechanisms exist. These 'migration mechanisms' can transparently move a component from one host to another such that the client has no awareness of the move. These mechanisms are provided by some (though not all) distributed architectures as a way of dynamically relocating components to provide load balancing and fault tolerance. Distributed component architectures can migrate entire components, including their functionality and data, and retain the state of the component from one machine to another.

The Problems with Distributed Components and the WWW

The distributed component is a new type of intelligent content, which has the ability to interact with other content of the same type. However, components of different architectures cannot directly

communicate with each other. Thus, Netscape's CORBA-compliant browser cannot use DCOM components, and Microsoft's DCOM-enabled browser cannot use CORBA components. As such, neither component architecture provides its content (the distributed component) with true ubiquity across the web in a way in which traditional content does.

This problem impacts on the architectures' use of migration. Most, including DCOM and Enterprise JavaBeans, do not support migration at all. However, even if they did, current distributed architectures cannot successfully employ a ubiquitous migration mechanism across the web, because no matter how open they are, the type of resource that can be migrated is tied too closely to the architecture itself. The web treats each software resource as a generic unit. The URL is used to reference it, and HTTP to distribute it, regardless of the resource's content type. In contrast, distributed architectures work only with their own content, and use their own reference formats to locate the components. Thus, only components created specifically to an architecture's specifications can be migrated, and only if both hosts involved in the migration support the architecture. Currently, however, the vast majority of content on the web today consists of JPEG and GIF files, and Java applets, which have no concept of a distributed architecture, much less the services that one can provide. Equally, servers supporting CORBA or DCOM are uncommon, leaving very few places for a component to physically migrate to.

Requirements for a Migration Mechanism on the WWW

For a migration mechanism to be successful on the web, then, it must recognise the diverse range of content that exists. As such, it must be completely decoupled from the content that it can migrate, and instead focus on the software resource: a generic unit of data which may or may not be aware of the mechanism (see Figure 1). Additionally, to truly be of benefit, the mechanism must fit in with the existing web architecture, rather than build its own set of standards on top of the existing web platform. In this way, it can be used by existing web content as much as by intelligent content such as distributed components, and can provide services which distributed components can use to enable the web to become a distributed platform.

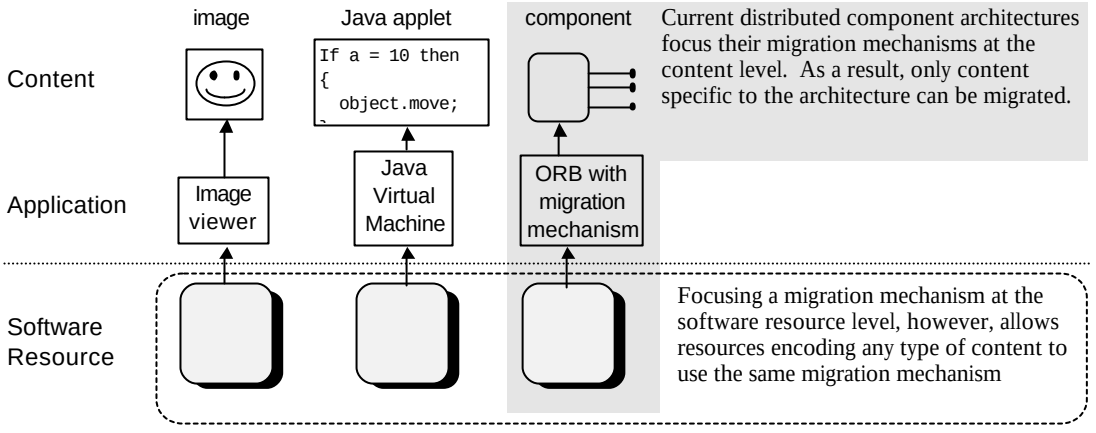

Figure 2: Migration Mechanisms at the Content and Resource Levels

True content migration, then, where content of any type can be freely migrated, relies on implementing migration at the resource level. In order to achieve this, the following set of

requirements for a resource-level migration mechanism have been identified:

- *Universal client access* - The mechanism must be accessible to clients of any type and should not require clients to be altered in order to use it. Thus, existing software does not need to be rewritten, and future software will not require any extra facilities in order to use it.

- *Content neutrality* - A web-based mechanism must be completely decoupled from the content it can migrate, enabling it to migrate all resources, no matter what type of content they encapsulate (see Figure 1).

- *Full integration with the web's current architecture* - the mechanism must reuse as much of the web's existing architecture as possible. Specifically, this means the reuse of HTTP and the URL. There is too much investment in the infrastructure supporting HTTP to change it overnight, and the URL is becoming accepted by the public as the only way to navigate to web resources. With businesses now using the URL as part of their advertising campaigns, URLs can now be recognised even by people without access to the web.

- *Practical design* - Resource migration can be *technically* achieved in many different ways, but adopting a *practical* approach means focusing on the requirements of web developers, existing web software, and (most importantly) web users, rather than focusing on a technically optimal design. A practical design also means one that takes into account the dynamics and characteristics of the web (and, by implication, the users of the web); an approach that technically works will not achieve ubiquity if it results in the web appearing to run more slowly.

In the next section, this set of requirements will be used to evaluate existing approaches to migration to see which is best suited to the development of a migration mechanism for the web.

DEVELOPING A WEB-BASED MIGRATION MECHANISM

Methods of Resource Migration

For any migration mechanism, there are four different methods through which a resource can be tracked once migration has occurred (Ingham et al. 1996). These are:

- Forward Referencing;
- Name Service;
- Callback;
- Search.

Forward Referencing

Forward Referencing involves leaving behind a reference in place of the migrated resource which points to the resource's new location. Thus, an object leaves behind a chain of references on each host it visits. For example, the migration mechanism of the 'W3Objects' system (Ingham et al. 1996), and 'Voyager', from ObjectSpace (an agent-oriented CORBA implementation), both adopt this approach.

When an object migrates in Voyager, a 'virtual reference' is left behind to forward messages to the new location. As an object migrates, more virtual references are created, forming long chains which eventually resolve onto the object itself. The W3Objects approach is similar, in that 'forward references' are created each time a resource migrates; however, to prevent long chains building up, 'shortcuts' can be created which allow a reference holder (that is, a resource with a link to the migrated resource) to bypass the chain of references, and reference the resource directly.

Suitability for the web

Voyager's mechanism is unsuitable for the web as, like most other distributed architectures, it only migrates Voyager-aware content, and is therefore not content-neutral. Surprisingly, the mechanism used by W3Objects will also only work with its own, object-oriented resource (termed a 'W3Object') and a specially-defined reference (termed a 'W3reference'), and so it too is not content-neutral. Furthermore, in order to use the W3Objects system, each client's browser must be adapted to work with W3References rather than URLs.

However, the Forward Reference method itself is unsuitable for use on the web. Each link in the chain of forward references adds another point of potential failure (Ingham et al. 1996), and if the chain breaks, then the resource is lost completely. Further, the characteristics of the web will make managing the chains unrealistic, as the number of forward references will increase with both time (some resources, such as autonomous agents, will migrate constantly) and space (every resource will require a chain of references to be maintained).

Name Service

The Name Service method employs an external system to maintain references to registered resources at all times. Such mechanisms generally focus on the use of the name used to identify a resource, and attempt to abstract any location-dependent information out of the name itself. For example, the Uniform Resource Name (URN) is a proposed standard by the Internet Engineering Task Force (IETF) for naming a resource *independently* from its location (Sollins and Masinter, 1994). Specifically, a URL is used to *locate* a resource, whilst a URN can be used to *identify* a resource (Berners-Lee, et al. 1994). The URN can then be mapped onto the URL through an external Resolver Discovery Service (RDS), which maintains the location of the resource. Should the resource have migrated, the RDS will resolve the URN into a URL that points to another RDS which can resolve the URL. Thus, a chain of references is built up across the resolver service, rather than across each visited server.

Suitability for the web

The URN identifies a resource independently from its location, and so subsumes the URL, treating it not as a name, but as a pointer to a location. Thus, whilst the URN has content-neutrality, it does not support full integration, as the URL cannot be used at the user level.

Also the Name Service method suffers from the same problems inherent with any 'chain' of references, as described above. Further, the method is not practical, as it does not take into account the characteristics of the web users: it requires, for example, that a resource's name remain invariant throughout its lifetime (which can be "for hundreds of years." (Sollins and Masinter,1994)), but in real life, the *ownership* of a resource can change within its lifetime, and the new owner may wish to give the resource a new name.

Callback

The Callback method relies on a resource to inform all other resources with references to it of any change in its location, in order to ensure referential integrity. The benefit of this approach is that there is no indirection, and so no chain of references need be maintained.

The Hyper-G system (Kappe, 1995) adopts this approach, maintaining a large database on the references used between resources. Should a resource move, the database is informed, and all references are updated. This is similar to the Name Service approach, in that an external service is used, but it is the *relationships* between resources which are maintained by the service, rather than the resources' *locations*.

Suitability for the web

This approach either requires each resource to know which other resource has references to it, or for an external service to maintain the references. However, the former approach is unrealistic, as the web is a federated system, with no central control: a resource has no way of knowing who or what is referencing it. Equally, the latter approach is unrealistic, as the size of the database of references would become impossible to manage, and many web servers are frequently offline, resulting in the database being swamped as it must store pending reference updates until they are online again (Briscoe, 1997).

Search

The Search method does not attempt to update the references between resources, or to maintain the location of a resource. Rather, it uses a sophisticated search mechanism to find the resource if it migrates. To ensure success, the entire system must potentially be searched, which involves flooding the network. This has the advantage that so long as the server hosting a particular resource is accessible, the resource can be guaranteed to be found, as the flood will eventually cover all servers. Thus, the search approach has perfect robustness. The Harvest information system (Mic Bowman et al. 1995) uses this approach to catalogue and index a distributed system's collection of resources. However, the Harvest system is used to index and search for pertinent information *within* resources, and so is effectively a search engine which can index an entire distributed system.

Suitability for the web

Whilst flooding a network provides perfect robustness, it is also the most costly method in terms of messaging overheads (Ingham et al. 1996). A flooding algorithm must be implemented which spans the entire web. To prevent the network being overwhelmed with packets (which, unchecked, would increase exponentially), attempts must be made to restrict the flood. This can be achieved by including Time To Live fields in any messages sent by such a mechanism, but this requires knowledge of the exact diameter of the web (Tanenbaum, 1996).

Selecting a Migration Method

The Callback Approach

The Callback service can be immediately ruled out. As has been said, a resource on the web has no way of knowing who or what is referencing it, and so any implemented Callback service simply cannot be used.

The Chain Approach

The Forward Reference and the Name Service approaches can be grouped together and termed the 'Chain approach', as both rely on a chain references to effect migration. The difference is simply that the Forward Reference approach leaves its references on the servers it has visited, whilst the Name Service approach relies on a separate service to store and maintain its chain of references. The concept of the Chain approach, then, can be examined in its own right, but does not meet all of the requirements specified above. The very fact that a chain exists exposes the whole approach to the chain's weakest link; in this case, the weakest link is the most unreliable server within the chain, meaning that a resource may be lost because *somebody else's* server has crashed. Finding that server can be difficult; worse, the resource's owner will have no control over the maintenance of the crashed server, and if it goes down permanently, the resource may be lost permanently. This is not just impractical, it is unacceptable to a network such as the web which is forming the platform for e-commerce: losing a resource can sever the relationship between an organisation and its customers.

The Search Approach

The Search approach comes closest to meeting all of the requirements specified above. Because the search would be performed within the network, the client need not be aware that a search is being performed; it simply receives the resource once it is located. Thus, *Universal client access* is achieved. The search process would be performed using the resource's URL; consequently, as long as the resource has a URL, it can be located, regardless of its content type. This achieves the requirement of *Content neutrality*. HTTP and the URL can remain. In fact, so long as the identifier is unique, it can be of any format, leaving the way open for future formats of identifier to be used with the same migration mechanism. *Full integration with the web's current architecture* is therefore achieved. However, the message overhead used to locate a resource cannot be ignored. Because it uses a flooding algorithm, the messages will grow exponentially with the size of the network. This is, at best, impractical when considering a network the size of the Internet. Thus, the Search approach fails the *Practical design* requirement. If this can be resolved, however, the *concept* of the Search approach is far more robust and scalable than the Chain approach. With no chains of references to maintain, and the ability to visit all hosts in a network, there is no weak link in the system. Resources, by definition, cannot be lost. As such, adopting a different search algorithm for the Search approach could result in a practical search-based migration mechanism on the web.

Adapting the Search Approach

The problems described thus far relate to a search algorithm which is parallel in nature, generating exponential traffic as the search progresses, and works on *unstructured* data. Such an approach cannot be practical on the web, because its latency overhead occurs at the wrong stage of the

migration process. The process of migration can be divided into two stages: first, a resource migrates; then, it must be located whenever a client wishes to use the resource. Generally, the migration stage can cope with higher latency times than the location stage. This is because there is no user interaction with the resource during the migration stage, whereas a resource usually needs to be located because a user wishes to download it. Currently, there is no migration mechanism on the web; locating a resource is simply a matter of connecting with the appropriate server. Any mechanism that is required to locate a resource will incur its own overhead, and this adds to the latency involved in actually accessing the resource. To the user, this latency is perceived as a slower response time of the web. With the Chain approach, the main overhead occurs during the migration process. Location is simply a matter of following a chain of references, and so long as this chain is not too large, latency should not be appreciably increased. However, with the parallel search approach described above, all of the overhead occurs during the location process, with the latency increasing as the search continues. Worse, the message overhead also increases (exponentially) as the search continues, resulting in a network with more location traffic than resource traffic.

This, however, is simply one end of a spectrum of search algorithms. For example, another approach could involve constructing a lookup table, with the set of all URLs on the web being mapped to each respective resource's *actual* location. The URLs can be ordered as appropriate, and a trivial search algorithm used to locate a specific URL within the lookup table. Whilst this centralised approach is not fault tolerant, and could result in *all* resources being lost, it does illustrate how structuring the data can fundamentally change the performance of the Search approach. What is required, therefore, is an approach which structures the data, but across a *distributed* system of migration-specific machines.

An Overview of a Web-based Migration Mechanism

This is the approach that is currently being investigated by the authors. A migration mechanism is being developed which uses an external (distributed) service to keep track of the URLs and the actual location of the respective resources. This is similar to the Resolver Discovery Service adopted by the URN approach, and provides the indirection required to retain the format of the URL whilst allowing the resource to reside on a machine with a different name. However, whilst the Resolver Discovery Service uses a chain of references to keep track of the migrating resources, the new approach uses what is, essentially, a migration-specific distributed 'database'. This database is constructed and queried using web-based technologies, such as Extensible Markup Language (XML). Rather than searching all of the resources on all of the servers across the web, the set of all resources are represented within this distributed database by their URLs, and it is this database which is searched to locate a resource. Fault tolerance techniques will be used to ensure no resources are lost, and load balancing will minimise the latency incurred. Because the database contains URLs, any content which can be addressed using a URL can safely migrate using this system. All that is required is for the system to be notified when a migration has occurred. This can be done by the server the resource has migrated from, or the server the resource has migrated to (or, for that matter, by the resource itself, if it contains intelligent content).

Development of this system is currently a work-in-progress, and results from the completed system will be published in a later paper. The next section discusses some of the new services such a system can provide to the web.

PROVIDING NEW WEB-BASED SERVICES

How a Migration Mechanism can Enable New Services

Within a distributed system, much use is made of the term 'transparency' (RM-ODP, 1995). This is used to convey the concept that the services performed by the distributed system (such as migration) happen without components being aware that anything has changed. Thus, a transparent migration mechanism is one in which components are migrated to another machine without the component, or a client wishing to access the component, being aware of the move. However, such a mechanism can be made 'translucent'; that is, the components can be moved transparently, but if they require the service themselves, they can use it to initiate their own migration. In this way, the migration is controlled by the component rather than the server hosting the software resource. For example, static content has no intelligence, and so cannot make use of a migration mechanism. As such, if the resource encoding the static content is to be migrated, it must be at the server's discretion. The server is therefore able to migrate the resource without the resource or any other host being aware of the move. Intelligent content, however, has the ability to use any service the network can provide. Thus, a migration mechanism can be used by intelligent content to migrate itself. It may choose to do this for the purpose of network optimisation (for example, if it detects that the server's performance has degraded due to excess demand), or it may do this to achieve a goal on behalf of a user. This would effectively enable the intelligent content to become a *mobile autonomous agent* (Franklin and Graesser, 1996); that is, software which can roam across a network, performing tasks on behalf of a user.

In this way, a translucent migration mechanism on the web can provide a host of new and extended services. The same mechanism can be used by intelligent content (to autonomously roam the web), and by web servers (to optimise the network); it can solve the 'broken link' problem typical of hypertext documents, whereby a URL embedded within an HTML document is rendered useless when the content it refers to is moved. It can also be employed on a company's intranet, allowing resources to migrate freely, either of their own volition, or transparently by the server hosting them. By providing its servers with the ability to monitor their own performance, a company can simply connect a new server to its intranet, and wait for resources to migrate to it from existing servers under strain. Using dynamic network configuration protocols, and wireless network technology such as Wireless LAN, this facility can be extended such that a server need only be brought into range of a mobile basestation, and switched on: the server will connect to the network, and the resources will populate the server, automatically.

Mobile Servers

Basing the migration mechanism on a search approach effectively provides a service which resolves the IP address of a machine given a specific *resource*. Thus, the same host can have many different IP addresses over time (for example, if the host is roaming) yet its resources will still be locatable (providing the host is accessible to the migration mechanism), because the mechanism ensures the resource can be located regardless of the current IP address associated

with it. This implies that mobile servers can be developed with IP addresses which change according to the server's location, without affecting the accessibility of the resources being hosted.

Services for Distributed Component Systems

Distributed component systems can use the mechanism to migrate components. Any type of content can use such a migration mechanism, and this includes intelligent content such as distributed components. Thus the mechanism enables the web to provide a generic migration service to such component systems. In this way, the web can become a distributed *platform*, enabling distributed systems to build their own specific services on top of the web's generic services. For example, system-specific messages between components can be routed to individual resources (components) irrespective of where the resources are located, using the generic services provided by the migration mechanism.

Optimising the Network through Network Traffic Profiling

Deciding which content to migrate and when can optimise both the performance of a server, and a network as a whole. Currently, certain network technologies and Service Level Agreements (SLAs) with network providers insist on the network user specifying the expected quality of service of the network at certain times of the day. For example, Frame Relay can ensure a certain throughput to the user over a short period of time by guaranteeing a Committed Information Rate (CIR). This CIR is the rate which is, on average, available to the user.

Determining the CIR is a difficult process and involves a good knowledge of the local traffic. The network manager has to plan for the expected traffic, keeping in mind that at very busy times he does not have the same throughput and availability of capacity above the CIR for bursty traffic. Traffic profiling is very important in such networks, whereby the traffic is monitored in order to determine the quality of service required. Research by the members of the author team is developing a methodology for profiling traffic in this way. It has been determined that whilst overall network traffic may be variable over the short-term, over time it only increases. The SLAs, therefore, can provide the business case for introducing content migration as a means of balancing the network and staying within the CIR. With a transparent migration mechanism built into a company's intranet, software resources can be migrated to balance the load not just across servers, but across the network. A traffic profiling system can be used to monitor the traffic on a company's network. If network traffic has increased at a particular point, resources can be migrated to ease the flow of traffic at the bottleneck. If the traffic is too great only at certain times of day, the profile will show this, and the resources can be migrated back and forth according to the time of day.

CONCLUSION

This paper has examined the various issues involved in developing a practical migration mechanism for the web. It has identified the requirements of such a mechanism, and examined some of the different approaches that can be used to implement a migration mechanism with respect to these. However, no current migration system meets these requirements, largely because they are not content-neutral. As such, the authors are currently working on a migration mechanism that will meet these requirements, and thus could form part of the web's infrastructure. A transparent, search-based, resource-level migration mechanism for the web, combined with existing distributed

component architectures and sophisticated network traffic profiling techniques should optimise both a server and the network, and can provide a new class of services to users. Whilst the web is currently a *distribution* system, the integration of a migration mechanism can provide the web with the services it needs to offer to become a ubiquitous *distributed* system.

REFERENCES

Berners-Lee, T.; Masinter, L. and McCahill, M. (1994), "Uniform Resource Locators (URL)", RFC-1738, CERN/Xerox/University of Minnesota, December 1994.

Berners-Lee, T., Connolly, D. (1995), "HyperText Markup Language - 2.0", RFC 1866, MIT/W3C, November 1995.

Berners-Lee, T., Fielding, R. and Frystyk, H. (1996), "Hypertext Transfer Protocol - HTTP/1.0", RFC 1945, MIT/UCS/UC Irvine, May 1996.

Briscoe R. J. (1997), "Distributed Objects on the Web", BT Technology Journal, Vol.15 No.2, April 1997, pp.158.

Ingham, D.; Caughey, S. and Little, M. (1996), "Fixing the 'Broken Link problem': the W3Objects approach", in *Fifth International World Wide Web Conference*, May 6-10, Paris, France.

Franklin, S and Graesser, A. (1996), "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents", in *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, Springer Verlag, 1996. http://www.msci.memphis.edu/%7Efranklin/AgentProg.html#agent

Kappe, F. (1995), "A Scalable Architecture for Maintaining Referential Integrity in Distributed Information Systems", in J.UCS, Vol. 1, No. 2, pp. 84-104, Springer, February 1995.

Mic Bowman, C.; Danzig, P. B.; Hardy, D.R.; Manber, U. and Schwartz, M.F. (1995), "The Harvest Information Discovery and Access System", Computer Networks and ISDN Systems, **28,** pp. 119-125.

OMG (1995), "The Common Object Request Broker: Architecture and Specification, Revision 2.0", Object Management Group, 1995.

Orfali, R., Harkey, D., Edwards, J.(1996), *The Essential Client/Server Survival Guide*, Wiley.

RM-ODP (1993), "Open Distributed Processing Reference Model (RM-ODP)", ISO/IEC DIS 10746-1 to 10746-4, 1995. http://www.iso.ch:8000/RM-ODP/

Sollins, K. and Masinter, L. (1994), "Functional Requirements for Uniform Resource Names", RFC 1737.

Tanenbaum, A.S. (1996), *Computer Networks, Third Edition*, Prentice Hall, 1996.