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 standard and consistent way. As such, this content has not been able to interact with other content, making the web a distribution system rather than a distributed system. This is beginning to change, however, as distributed component architectures such as CORBA, JavaBeans, and DCOM are being adapted to integrate tightly with its 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: object migration. The paper argues, however, 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 paper goes on to describe the requirements of such a *content* migration mechanism, and shows how, combined with network traffic profiling, a network can be optimised by dynamically migrating content according to traffic demand..

1. INTRODUCTION

1.1 SOFTWARE RESOURCES

The World Wide Web ('the web') is a platform for distributing rich, consistent content over the Internet. As such, today's web is a large distribution system. The content which is distributed can come in many forms, representing many different media types, each of which must conform to its own universal standard. Each instance of content is a single file, which can be termed a *software resource*. As such, each software resource is either a binary or a text file, and encodes content such as images, or video, in a standardised, consistent way. The web therefore comprises many different standards. The three main standards which define the platform itself are

- the Universal Resource Locator (Berners-Lee, et al 1994);
- HyperText Transfer Protocol (Berners-Lee, et al, 1996);
- HyperText Markup Language (W3C, 1997);

The Universal 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 content in a consistent way across all computer platforms.

Other standards exist which are not part of the specification of the web itself, but which contribute to its ubiquity and openness, enabling machines of any platform to display the same content (e.g. pictures, sound, animation, video, etc.) consistently. For example, the GIF (Graphics Interchange Format) standard, developed by CompuServe, is a standard format for

compressing images. A GIF viewer is a small piece of software which interprets a GIF image file and displays the image it contains. This GIF viewer essentially reads in a generic, platform-independent file which contains an encoding of the image, and converts the encoded information into platform-dependent data which is displayed on the screen as the decoded image in a consistent way across all platforms. The same can also be said for other content formats (e.g. JPEG, MPEG, AVI, QuickTime), each of which encode a specific media type according to a 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.

1.2 STATIC AND INTELLIGENT CONTENT

The content discussed so far, however, has principally comprised static files. 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). In fact, the web has traditionally been composed primarily of static content: individual files without functionality, and without the ability to interact with other files. 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 of the files it is currently distributing is growing along with the web itself. To distinguish between files which contain content (such as an image), and files which have some form of computational intelligence inherent within them, this paper will define the terms *static content* and *intelligent content*, respectively.

Intelligent content currently comprises small self-contained blocks of code which reside on a server, and are downloaded onto a client machine, where they are executed by the client's browser, 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, could not originally communicate 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.

2. DISTRIBUTED COMPONENTS

2.1 AN OVERVIEW OF DISTRIBUTED COMPONENT SOFTWARE

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, regardless of the programming language used to develop the component. Whilst a general component architecture enables software 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. Currently, the distributed component field comprises three different architectures:

Microsoft's Distributed Component Object Model (DCOM), Sun Microsystems' JavaBeans, and the Object Management Group's Common Object Request Broker Architecture (CORBA).

DCOM is the distributed extension to Microsoft COM (Component Object Model), and extends the basic COM 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, but only on machines supporting the Microsoft Windows family of operating systems.

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, platform-independent protocols for interoperable communication across platforms, and a number of platform-independent services which provide the components with a number of useful services required in a large distributed system. These services include such things as security, transaction processing, persistence services for storing an objects state across machines, and naming and location services for finding components across a distributed system.

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.

2.2 DISTRIBUTED COMPONENTS AND THE WWW

Such distributed component systems are increasingly being incorporated into the web. Distributed components are becoming the next software resource to share server space with existing 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.

One of the benefits of a distributed component system is the ability of an application to be distributed across multiple hosts across a network, such that no one host is burdened with providing the computational power required for running the whole application. As such, 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 dstributed component 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 or host's network 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 and involves a fair amount of administration. Most distributed applications comprise many components, and it would be impractical to manually redistribute the components whenever necessary.

As such, various automatic component relocation services exist. These 'object migration' services, as they are termed, are designed to provide a mechanism which 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.

3. OBJECT MIGRATION

Distributed component architectures, such as CORBA, migrate entire objects, including an object's functionality and data, and retain the state of the object from one machine to another. A location forwarding mechanism (which can be used to enable the tracking of migrated objects), is provided as part of the General Inter-Orb Protocol (GIOP), a protocol used by CORBA components to communicate across machine boundaries (OMG, 1995). However, current distributed architectures do not all support object migration. The location forwarding mechanism in CORBA, for example, is not a required part of the CORBA-specification, merely an optional addition, and so a CORBA implementation cannot be relied upon to support it. Other distributed architectures, such as Microsoft's DCOM and JavaSoft's JavaBeans, do not support a migration mechanism at all.

Distributed component architectures are now being adapted for integration with the Web. Netscape, for example, has integrated CORBA functionality into its Communicator 4.0 browser, allowing the browser 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. However, Netscape's browser cannot use DCOM components, and Microsoft's 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. Thus, even if such an architecture does support migration, it is of little use across the Internet if the only resource which can be migrated must be compliant with that architecture. In fact, this is a flaw in current distributed architectures, which, no matter how open, tie the object too closely to the architecture itself, resulting in migration mechanisms which can only migrate objects created specifically to that architecture's specifications. The vast majority of the objects on the web today are JPEG and GIF files, and Java applets, and have no concept of a distributed architecture, much less the services that one can provide. As such, to have any real benefit on the web, the resources being migrated should be consistent with the types of software resources most widely used. For example, a migration mechanism based on CORBA cannot migrate an image file unless the image file is explicitly encapsulated within a CORBA object. This would require both the server and the client to support CORBA.

What is required is a migration mechanism completely decoupled from the resources it can migrate. Such a mechanism must regard all resources as "black boxes": simple binary or text files (the distinction does not matter to the mechanism) which may or may not be aware of the mechanism. To be truly of benefit on the web, the mechanism must fit in with the existing web architecture. As such, it must reference each resource using a standard URL. With so many businesses using URLs on their promotional material, any mechanism which required a new format for resource location would not be accepted.

Within a distributed system, much use is made of the term 'transparency'. 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 objects are migrated to another machine without the object, or a client wishing to access the object, being aware of the move. However, such a mechanism can be made 'translucent'; that is, the objects 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 software resource 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 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. For example, a Java applet is able to use the Domain Name System to resolve domain names into IP addresses; the DNS is an inherent part of the Internet, one which both servers and intelligent content can use at will. If a migration mechanism was implemented such that servers could use it to migrate static content, and intelligent content can use it to migrate itself, then the intelligent content transforms itself into a mobile agent: an object which roams from machine to machine at will.

In this way, a translucent migration mechanism on the world wide web can provide transparent migration services for mobile agents, fault tolerance and load balancing for web servers, and can solve the 'broken link' problem typical of hypertext documents, whereby a URL embedded within an HTML document is rendered useless when the resource it refers to is moved. Such a migration mechanism also has benefits within an intranet. An intranet is an organisation's internal network which uses the same protocols and technology as the Internet. As such, any migration mechanism designed to work on the Internet can also work on an intranet. This has major benefits for large organisations with a distributed workforce. As has been said, a distributed system distributes applications across many networked hosts, but once installed on a host a component cannot be easily moved. With a migration mechanism, however, components can migrate freely, either of their own volition, or transparently by the server hosting them. This means that the components can be dynamically relocated by the servers according to server load. Should a server be under strain, it can use the mechanism to migrate as many resources as required to a server under less pressure. This facility can be extended to provide for automatic server configuration. A server can simply be purchased, connected to the network, and switched on. Other servers under pressure can then simply migrate resources to the new server, without any manual configuration being required. In this way, the new server does not need configuring; it happens automatically. If the network technology used is wireless, (for example Wireless LAN), then the process of introducing a new server into a network would be even simpler.

4. NETWORK TRAFFIC PROFILING

In order to identify which content should migrate it is necessary to be able to determine the demand that exists for each type of resource and from where it is occurring. A means of achieving this is through the use of network traffic profiling. 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 is a statistical multiplexing based protocol, and as such allows the access circuits and core trunks of a network to be better utilised for bursty communications. In theory it allows the allocation of an entire access link to a single Permanent Virtual Circuit (PVC) for the duration of a burst and is, therefore, very suitable for Local Area Network (LAN) interconnections. To ensure a certain throughput to the user over a short period of time, the network guarantees a Committed Information Rate (CIR). This CIR is specified in Kbps and is the rate which is, on average, available to the user. If a user transmits data at a rate exceeding CIR a control bit is set to indicate this frame can be selectively discarded when there is congestion on the network.

To determine 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. Depending on the use of protocols, this can have a major influence on the choice of the CIR. Very often the choice of CIR is not influenced by the traffic volume itself,

but by the budget a company is able to afford. However, choosing too low a CIR can cause not only congestion, data loss, time-outs of protocols and poor throughput but also a financial loss to the network provider. The SLAs, therefore, can provide the business case for introducing content migration as a means of achieving CIR. Traffic profiling is very important in such networks, whereby the traffic is monitored for a period of time 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. The problem companies face, however, is that network traffic over time only increases; whilst for short periods the traffic may be variable, over time the traffic generated steadily increases.

However, with a transparent migration mechanism built into a company's intranet, components and resources can be migrated to balance the load not just across servers, but across the network. As such, a traffic profiling system can be used to monitor the traffic periodically 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.

5. CONCLUSION

A transparent, architecture-independent 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 in a dynamic, adaptive way. Such a migration mechanism can also provide a means for mobile agents to roam across a network without getting lost, and can solve the web's broken links problem. It is the next phase of the web and distributed systems in general.

Such a mechanism is currently being developed by the authors, and an initial design has been produced which can provide such migration services without breaking the existing Internet architecture. That is, the URL is retained for locating resources, the resources can be of any type, the mechanism can be integrated gradually into the existing infrastructure, and the mechanism imposes no foot-print on the client. This mechanism is currently work in progress and will be expanded upon in future publications.

6. REFERENCES

(Berners-Lee, 1994) T. Berners-Lee, L. Masinter, M. McCahill, "Uniform Resource Locators (URL)", 12/20/1994.

(Berners-Lee, 1996) T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext Transfer Protocol - HTTP/1.0", May 1996.

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

(W3C, 1997) Latest specification of HyperText Markup Language, version 4.0. http://www.w3.org/TR/REC-html40/