THIN SERVERS - AN ARCHITECTURE TO SUPPORT ARBITRARY PLACEMENT OF COMPUTATION IN THE INTERNET

J.C. Diaz y Carballo, A. Dearle
School of Computer Science, University of St Andrews North Haugh, St Andrews
Fife KY16 9SS Scotland,U.K.
Email: jcd@dcs.st-and.ac.uk, al@dcs.st.st-and.ac.uk

R.Connor
Dept. of Computer Science and Information Systems,University of Strathclyde,Livingstone Tower
,26 Richmond Street,Glasgow G1 1XH, Scotland,U.K.
Email: Richard.Connor@cs.strath.ac.uk

Key words: Thin Server, Code mobility, Security, Persistence, Cingalet, XML

Abstract: The Internet is experiencing an overwhelming growth that will have a negative impact on its performance and quality of service. In this paper we describe a new architecture that offers a better use of Internet resources and help improve security at the server nodes. The Thin Server Architecture aims to dynamically push code and data in arbitrary locations throughout the Internet. We give implementation techniques related to code mobility, the choices we have made for our architecture, our on-going implementation work, and future directions.

1 INTRODUCTION

Our hypothesis is that it is viable to construct Internet applications that will be more efficient than current Internet application systems by the appropriate geographical placement of computation. To investigate this idea we are involved in the construction of an experimental prototype that consists of a collection of inexpensive networked machines that we call Thin Servers.

The Internet presents two basic execution scenarios that do not always allow the optimal physical placement of the execution. These are:

- Execution of pre-installed programs, which can access and perhaps change the state of the execution context,
- And execution of dynamically installed programs on clients, which usually cannot.

Currently most Internet services are based on the paradigm of providing services in a centralised way by means of the classic client-server architecture. By contrast, we propose to move and execute code close to where the data is, rather than close to where its user is. Our proposal permits computation to occur safely in arbitrary physical locations, through the introduction of new security domains that can be placed within the existing Internet. Positioning Thin Servers outside the protection domains of the conventional servers manifestly avoids any security risk. The Thin Server Distributed System Architecture that we describe here sets sights on increasing efficacy of other devices currently in the Internet rather than their replacement.

2 ARCHITECTURAL MODEL

This section describes the model chosen together with the motivation for our approach. To construct the prototype we have focused on the primordial mechanisms to test and study the technology.

To achieve the goal of performing and initiating computation in geographically appropriate locations, the technology needs the capacity to push code to, and start computation on, remote servers. Our

approach is based on: a new programming methodology, QNX Neutrino™ [QNX ; Krten 1999] a real-time operating system with no more than a minimal kernel, and ability to support persistent application software.

Security and Performance are motivating drivers for the Thin Server Architecture. If a computation is sited far from the data over which it operates, it will perform worse than a computation in close proximity to its data. We propose to capitalise on this observation by permitting computation to occur in the most appropriate geographical locations. In some cases, this will be far from the initiator and close to the data. In other cases it will require data to be fetched close to its user, for example with web caching. When a service requires a considerable amount of data to be transmitted to perform a task, for example, in data mining applications, it would make sense to execute the query at the producer site. However, such code placement directly comprises security and many high value sites would not permit arbitrary computations to run on the same server as the data. These restrictions are motivated by a number of security concerns namely, agents may attack other agents on the host, the agent may attack the host and the host may attack the agent.

The first two cases are relatively well known, and some solutions based on cryptographic techniques have been given to address these issues [Baumann J. 1998; Minar 1999]. Mechanisms like authentication and strong typing help in inter-agent communication security policies. For example, the Java sand-box in combination with an appropriate host security policy based on fully trusted non-mobile code like shadows in the case of HIVE [Minar 1999], provide a solution for agent-to-host interaction. Good examples of host protection against malicious agents are the Java Applets. The third problem, code (agent) protection against a malicious host is complex and endemic in mobile code. Only partial solutions to this have been achieved [Sander 1997], and none is being used in real-world applications [Baumann J. 1998]. We believe that Thin Servers running code in protected security domains fully address this problem. Consequently this new architecture focuses on security and performance. We believe that new pioneering services may significantly benefit from our approach in terms of better-customised service strategies

2.1 Persistence

The persistence of deployed code and the data created by it is independent of the lifetime of the process in which it runs. Should the node or process in which a pushed object is running fail, it will be restarted by the underlying operating system and continue to execute. The pushed code must be notified of this event to permit necessary housekeeping such as rebinding to the environment, or making adjustments to user level protocols that may have timed out etc. This simple mechanism permits code to be written without the application programmer having to be concerned with the management or longevity of data. In addition to making the applications simpler to write and reason about, less code needs to be sent across the network since the persistence of code and data are handled by the Thin Server.

2.2 Naming and Binding

A common requirement in distributed systems is for data to be shared between multiple processes. Sharing may be achieved by interaction with local or remote servers and via the local file system. However, the availability of a local file system compromises system security. In the Thin Server Architecture we propose to eliminate this security loophole and replace it with an integrated naming model. Figure 1 shows the naming model. Objects may be globally or privately symbolically named. If they are privately named, they may only be accessed in the environment in which they were created. Objects that are globally named may be used from any addressing environment, which can name the addressing environment in which they were created. Two classes of operations, namely invoke and copy, may be performed on named objects: objects may have their operations invoked and may be copied to other addressing environments.

When this naming mechanism is recursively applied, an Algebra is created which has many similarities with the Ambient model of Cardelli [Cardelli 1998]. In the architecture proposed here, for pragmatic reasons, we intend to recursively apply the mechanism once making addressing environments either public or private to a particular node. Thus, addressing environments may themselves be named using a naming scheme similar to that used for objects. The copying of object closures in object-oriented systems presents many problems: deep copy may result in a prohibitively large object graph being copied across the network. Conversely, sending partial closures often results in

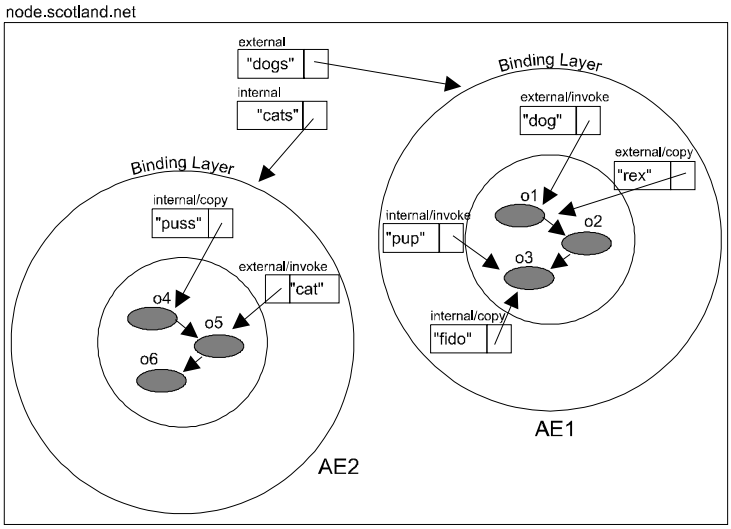

Figure 1: The Naming Model

semantic inconsistencies, which results in systems that cannot be reasoned about. One approach to this problem is to implement coherency protocols across the distributed system. However, this is impractical in a massively distributed system such as the Internet. We propose to address this problem by shallow copying combined with a causal viewing mechanism [Vaughan 1994] based on log structuring [Hulse 1996]

2.3 Architectural Components

The Thin Server Architecture is based on the concepts of Cingalets, Pushers, Suckers, Dispatchers, Runners, Thin Servers, Persistent Stores, and a micro-kernel real-time OS. Both Persistent Stores and the micro-kernel OS [Neutrino] are not endemic components of our technology and out of the scope of this paper. Nonetheless, these concepts are well described in [Brown 1998] and [QNX ; Krten 1999], respectively. The first six components listed above are described below.

```
public interface CingaletInterface {
    public void init();
    public void restart();
    public void clone();
    public CingaletLocalEnv
          getLocalEnvironment();
    public CingaletGlobalEnv
          getGlobalEnvironment();
}
```

Figure 2. The Cingalet Interface

A Cingalet is an entity that contains the code and the collateral information to instantiate it on a Thin Server. In other words, in our technology code mobility is performed via Cingalets. Figure 2 shows the Java interface of a Cingalet. This interface is designed to support code that is to be stored, executed or pushed onto or from a Thin Server. Any class whose instances intend to be treated as a Cingalet by a Thin Server should implement the *CingaletInterface* interface. The class has to define the five methods described below:

- **init** is used to instantiate a Cingalet when it starts running on a new Thin Server.
- **restart** is used to recreate the environment of a Cingalet in cases such as when a node starts up after a failure, or shutdown.
- **clone** is used to copy the Cingalet and its local environment to a new node.
- **getLocalEnvironment** is used to get the local environment of a Cingalet.
- **getGlobalEnvironment** is used to get the global environment of a Cingalet.

We now put the different environments of a Cingalet into context by referring to the naming model described in 2.2 and depicted in Figure 1. The local environment is a namespace private to the Cingalet. In Figure 1 the local environment of the Cingalet named *AE2* encloses two name-value pairs containing the names “puss” and “cat”. The global environment is a namespace shared by all Cingalets running in a node. In Figure 1, the global namespace contains two Cingalets named “dogs” and “cats”.

Pushers are entities written in Java sited in both conventional and Thin Server nodes which construct Cingalets and send them to Thin Servers. We have developed a protocol inside the Pushers to construct Cingalets, and to set up communication channels with Suckers at the remote nodes. This protocol is based on a *classloader* that scans the *classpath* of the code to be moved in the persistent media. This mechanism identifies and loads the code and its transitive closure. In other words, it builds a Cingalet. The classes that are part of the **java, javax,** and **system** standard class containers are rejected (not loaded), as they have to be present at every node in the architecture. Finally, once the Cingalet is constructed the Pusher generates an XML wrapper containing the Cingalet together with its necessary data parameters, which is sent to the destination Thin Server. We consider that encoding Cingalets into an XML [W3C] framework can only present advantages, as XML is becoming the *de facto* standard vehicle to interchange data in the Internet.

Suckers are Java dæmon processes placed in Thin Server nodes, which act as the counterpart of Pushers. Consequently, we also have developed a protocol inside the Suckers to download and install Cingalets on Thin Servers. A Sucker basically reverses the process performed by a Pusher. That is, it extracts the Cingalet from the XML wrapper, and puts it into the persistent store before signalling the Dispatcher to clone or execute the code as directed by the XML encoded instructions.

Dispatchers are C++ dæmon processes that start up JVMs to create Runner agents. They are built as Neutrino native components called Resource Managers [Krten 1999].

Runners are Java agents that allow to instantiate Cingalet agents as threads (same JVM) in a Thin Server. Their functionality is achieved by means of a customised *classloader*.

Figure 3 depicts the internal structure of a Thin Server, and a Conventional node. This figure shows a typical use of the architecture. After downloading a Cingalet, the Sucker proceeds to install the code in the persistent store. The code is then loaded by an instance of the Runner created by the Dispatcher. Finally, the Pusher clones the code (Cingalet), and pushes it to another Thin Server.

Conventional nodes are not permitted to run Cingalets since they are not fully trusted entities in the architecture. Consequently, a Pusher is the only supporting software that has to, and its permitted to, be deployed on them.

3 EXPERIMENTAL FRAMEWORK

For many different reasons fully described in [Dearle 1992], conventional operating systems do not provide a suitable implementation platform for the kind of the Thin Server Architecture we present in this paper. A small configurable operating system (OS) providing suitable support for the persistence model we propose (see 2.2) is required. In doing so, we started off the development of our own OS, called Charm [Dearle 2000]. However, since the task of implementing an OS is extremely time consuming, we have abandoned this approach and now run our technology on the top of what we consider as an excellent alternative, the QNX Neutrino real-time operating system [QNX]. The Neutrino OS offers us the appropriate functionality to construct the naming and binding mechanisms described in section 2 above it.

In an approach such as ours, a choice has to be made as to the language and the code format that may be pushed across the Internet. Thus, we mainly make use of the Java language and its associated software resources and environment as it provides a powerful vehicle owing to its cross-platform capabilities. For similar reasons we have chosen XML as the technology for the code format.

4 RELATED WORK

A number of researchers have described work, which apply a high level semantics to the existing Internet. Cardelli and Davies [Cardelli L. 1997] describe language features that allow the explicit inclusion of the concepts of non-determinism and transfer rates within computations. Much more closely related is Cardelli's more recent work on Mobile Ambients [Cardelli 1998], which gives a formal algebraic approach to the naming and connectivity models among autonomous processes or agents.

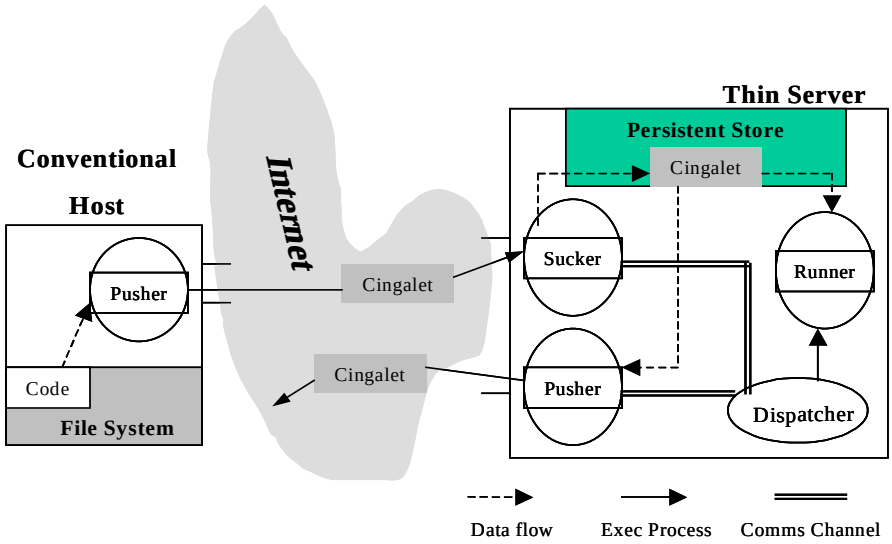

Figure 3: Inside a Thin Server and a Conventional Node

A plethora of Java based Agent Systems ranging from commercial products such as Voyager [ObjectSpace], Kafka [Fujitsu] to a number of research projects such as Aglets [IBM], MOLE [Baumann J. 1998], HIVE [Minar 1999], ARA [Peine 1997] and D'Agents [Gray 1995]. These systems cover the entire code migration spectrum from weak to strong schemes. All the systems listed above present a weak migration policy except ARA and D'Agents. In all the cited systems an agent may be defined as a process that migrates among the nodes (called places in MOLE and ARA, cells in HIVE etc.) within a network to perform its task, and which operates on behalf of its user. Every node that an agent may visit must support a special infrastructure to deal with the management of system services and resources. This is usually done via non-mobile agents that are fully trusted code. Agents generally collaborate via asynchronous message passing schemes, and can clone themselves and communicate with non-mobile agents via RPC or Remote Method Invocation. The common concepts of Agent Systems overlap with the approach that we propose here.

The Infospheres project from Caltech [Chandy 1996] has some overlap with the project proposed here. They propose a system of distributed Java processes (dapplets) that can be connected using asynchronous message passing. Whilst their work focuses on designing, implementing and verifying services that can be constructed using their dapplet primitives, we are proposing an architecture on which the Infospheres system could be practically constructed. The W3Object model [Ingham 1995] addresses the problems of referential integrity and transparent object (resource) migration through Web resources encapsulated as objects, with well-defined interfaces through which all interactions occur. Like Infospheres, this project is related to the one proposed here but to be successful requires the kind of architecture propose.

Finally, despite affinities with many of the systems described above none of them contains in its core the idea of providing support for persistence.

5 FUTURE DIRECTIONS

An architecture as large and complex as ours cannot easily be implemented by a few researchers by targeting in unison the many different computing aspects such as communication protocols, security, code mobility, referential integrity of objects and repudiation, among others. Subsequently, in moving towards the goal of building up a prototype that allows us to demonstrate the viability and capacity of our approach, we have focused so far on: Development of technology to push code onto a Thin Server, the construction of a XML framework to wrap the Cingalets, and development of mechanisms whereby code resident on a Thin Server can be executed in independent address spaces.

The next aspects we will investigate are: to provide persistence for computations running on Thin Servers, to develop protocols for exchange of code and data between Thin Servers, to provide our

code push technology with capacity to work with authentication mechanisms, and the development of a naming and locating service for XML encoded java classes.

6 CONCLUSIONS

In this paper we have discussed Internet execution scenarios and observed that the placement of computation in arbitrary geographical locations can enhance performance of Internet applications. We have also discussed the security concerns that this technique gives rise to. During the last decade code mobility has emerged as a new trend in the ecosystem of distributed computing. Code mobility can potentially offer advantages and solutions over the predominant traditional client-server architecture present in the Internet. In this paper we have described a new distributed approach, called the Thin Server Architecture, which embraces this trend in an effort to create an ideal platform which will be benefit existing and new application systems in the Internet. The Thin Server Architecture addresses many of the problems highlighted in this paper in an attempt to offer a new complementary technology that may be integrated without difficulty, with standard Internet technologies.

We have described the central components of our architectural model, and the motivations for our approach. We also have discussed the choices we have made to implement and run the Thin Server Architecture. Consequently we have given reasons for the use of Neutrino, the Java language and support for persistent application software.

7 ACKNOWLEDGEMENTS

This work is based on the project **Supporting Internet Computation in Arbitrary Geographical Locations,** supported by EPSRC GR/M78403.

REFERENCES

Baumann J. (1998). “Mole - Concepts of a Mobile Agent System.” WWW Journal(Special issue on Applications and Techniques of Web Agents).

Brown (1998). Persistent Object Stores, PhD Thesis, University of St. Andrews.

Cardelli (1998). Mobile Ambients. Lecture Notes in Computer Science, 140-155.

Cardelli L. (1997). Service Combinators for Web Computing. Proceedings of First Usenix Conference on Domain Specific Languages, Santa Barbara.

Chandy (1996). A World-Wide Distributed System Using Java and the Internet. Proceedings of International Symposium on High Performance Distributed Computing (HPDC-5), Syracuse, New York.

Dearle (1992). An Examination of Operating System Support for Persistent Object Systems. Proceedings of 25th Hawaii International Conference on System Sciences, Poipu Beach, Kauaii.

Dearle (2000). “Operating System Support for Persistent Systems: Past, Present and Future.” Software - Practice and Experience,30, 4(Special Issue on Persistent Object Systems): 295-324.

Fujitsu “http://www.fujitsu.com/products/software/.”

Gray (1995). Agent Tcl: A transportable agent system. Proceedings of the CIKM Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management (CIKM 95), Baltimore, Maryland.

Hulse (1996). A log-structured persistent store. Proceedings of 19th Australasian Computer Science Conference, Australian Computer Science Communications.

IBM “http://www.trl.ibm.com/aglets/.”

Ingham (1995). W3Objects: Bringing Object-Oriented Technology to the Web. Proceedings of Proc. Fourth International WWW Conference, Boston, Mass., USA.

Krten (1999). Getting Starting Started with QNX Neutrino 2,, Parse Software Devices.

MIME http://www.oac.uci.edu/indiv/ehood/MIME/MIME.htm

Minar (1999). Hive: Distributed Agents for Networking Things. Proceedings of ASA/MA'99, the First International Symposium on Agent Systems and Applications and Third International Symposium on Mobile Agents.

ObjectSpace “Voyager.” http://www.objectspace.com/products/voyager/.

Peine (1997). An introduction to mobile agent programming and the Ara system, Department of Computer Science, University of Kaiserslautern, Germany.

QNX http://www.qnx.com/products/os/rtos6.html, http://www.qnx.com/literature/nto_sysarch/index.html.

Sander (1997). Protecting Mobile Agents Against Malicious Hosts. Springer-Verlag:. Heidelberg, Germany.

Vaughan (1994). Causality Considerations in Distributed Persistent Operating Systems. Proceedings of 17th Australian Computer Science Conference, Australian Computer Science Communications.

W3C XML, http://www.w3.org/XML/