REPORT DOCUMENTATION PAGE

AFRL-SR-AR-TR-02-

0412

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing ins gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this of information, including suggestions for reducing this burden to Washington Headquarters Service, Directorate for Information Operatio 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE (DD-MM-YYYY)	2. REPORT DATE	3. DATES COVERED (From - To)
17-05-2002	Final	15 April 1998 - 14 November 2001

4. TITLE AND SUBTITLE

System Software Support for Mobile-Agent Computing

5a. CONTRACT NUMBER

5b. GRANT NUMBER

F49620-98-1-0439

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Professor Joseph Pasquale

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

The Regents of the University of California
University of California at San Diego
Office of Contract & Grant Administration - 0934
9500 Gilman Drive/La Jolla, CA 92093-0934

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

USAF, AFRL
Air Force Office of Scientific Research
801 N. Randolph Street - Room 732
Arlington, VA 22203-1977

10. SPONSOR/MONITOR'S ACRONYM(S)

AFOSR

11. SPONSORING/MONITORING AGENCY REPORT NUMBER

12. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

SEE ATTACHED

20030106 105

15. SUBJECT TERMS

agent-based computing, distributed computing, middleware

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE		7	Joseph Pasquale
					19b. TELEPONE NUMBER (Include area code) 858/534-2673

FINAL REPORT

System Software Support for Mobile-Agent Computing

Joseph Pasquale

Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114
pasquale@cs.ucsd.edu

September 2002

ABSTRACT

We investigated new system software architectures that support mobile code, based on network (and distributed) computing. The primary goal was to support an application's reliance on network resources, rather than local ones (which are often scarce in defense mobile-computing situations), to meet their communication and computational demands. We have developed a powerful extension of the client/server model, which we call the *Extended Client/Server* (ECS) model of distributed computing. Rather than a client sending requests to a server and receiving its reply as in the traditional client/server model, the client produces a mobile code object called an *active extension*, which then carries out the request/response interaction with the server. This active extension can execute at the server, at the client, or (most importantly) at another location where, presumably, it can interact with the server more advantageously (e.g., with higher performance, higher reliability, or higher security) than if it was co-located with the client. We explored the design (and implementation) space for the ECS model, we completed multiple prototypes, implemented independently by different students, all of which provide basic support for remote computing, but each focusing on different areas of concern, such as security, resource allocation, client/agent and agent/server protocols, etc.

OBJECTIVES

1. Understand the strengths/weaknesses of the Extended Client/Server (ECS) model
2. Understand the dimensions of the design space for the ECS model
3. Explore the design space by developing prototypes; evaluate the prototypes and determine what the important variables are
4. Develop a single design for system-level mechanisms that support the ECS model based on the previous experience
5. Develop some prototype applications to exercise the ECS mechanisms to determine their effectiveness, insofar as performance, reliability, and security, are concerned

STATUS OF EFFORT

The research is now complete, and we have completed all of our objectives. In the following section, we describe research highlights and accomplishments.

ACCOMPLISHMENTS

Our primary contribution is a new model of distributed computing based on mobile code, called the Extended Client/Server (ECS) model. This model is especially applicable in challenging environments (e.g., defense), with mobile clients connected via wireless links, and promotes higher performance, reliability, and security. We briefly describe the model and its use.

DISTRIBUTION STATEMENT A
Approved for Public Release
Distribution Unlimited

In the ECS model, we allow either the client or server to be decomposed into a base and (active) extension. In most cases, it will be the client that is extended, as it is typically the client that will connect via a wireless link. Because of the wide variety of user devices that will act as clients, especially in defense applications, the ability to tailor responses from existing servers, without forcing these servers to be aware of this variety and therefore have multiple versions of the same content, is critical.

Prior to contacting a server, the client would deploy an extension to a node that supports the execution of extensions. The most flexible approach is for the extension to be in the form of mobile code, which originates either with the client, or from a remote repository. In the latter case, the client would supply a reference (rather than the code itself) that would indicate where to retrieve the extension.

The middleware that supports this model is also structured according to the client/server model. An extension runs on an extension server, which provides its execution as a service. Obtaining this service might require authentication, a check on whether the extension is safe to run, and other controls of admission. Once accepted, the extension server would provide processing, memory, and communication resources (e.g., buffers, bandwidth) to effectively run the extension.

Once the client s base and extension are in place, whenever the client is to make a request to a server, this request may originate at the client base or its extension. If from the base, it first gets passed to the extension, and this transfer can occur using a specialized protocol, e.g., designed to specifically address wireless link issues. The extension also has the opportunity to modify the request, or keep a copy, before passing it to the server. As far as the server is concerned, it receives the request from the client, unaware that it came from the extension or that it originated at the base.

As implied above, the request may also originate from the extension. This might occur if the request that was saved as a copy at the extension is to be repeated; or, it might occur if the extension did not immediately forward the request after it was received, as in a real-time situation where the extension is to forward it at a specific time, avoiding the potential variability that might be introduced by a wireless link had the request been sent directly from the base.

While we have focused on the common case of extending the client, extending the server is useful also. The obvious scenario is where the server faces problems analogous to those we considered for clients: limited processing capabilities and connectivity via a wireless link. A good example is a small wireless monitoring device that senses its environment and acts as a simple server by accepting requests for the current state.

But there are other situations that motivate extending the server that have nothing to do with processing limitations or wireless connectivity. For example, a server extension can reduce the distance, and therefore latency, to a client by providing a presence and limited selection of services near the client. Such a service might be a cache: if the cache contains the content the client requests, it provides it without accessing the server base.

While this example shows how to improve performance as well as reliability (if the base is temporarily down, the extension can still provide limited services), extending the server can also be applied to improve security. Consider the problem of denial of service attacks, where the server is bombarded with requests and effectively shutting it down. Now consider a server that launched numerous extensions, thus allowing the server to now have multiple presences. A client would send its requests to one of them (which it sees as the actual server) that would then simply be passed along to the server base. A denial of service attack upon one of the extensions would be isolated to the machine serving it (which would need to be able to recognize such an attack and then not pass along the requests).

One of our objectives was to explore the design space of the ECS model. We now describe three projects in this regard. These projects focus on various practical aspects of the issues presented above, and show how we realized them in useful working systems. We first describe a Java-based implementation of middleware that supports extensions. Next, we describe a system that specifically deals with extending Web applications to customize content for non-standard user devices. Finally, we describe a system that supports protocol agents and their pattern-based development.

Java Active Extensions

The Java Active Extensions middleware system addresses the practical issues involved with distributing an endpoint s functionality by providing mechanisms to load, execute, and interact with code at a remote location. We use Java as the implementation language for these mechanisms because of Java s widespread availability, platform independence, and support for mobile code (e.g. dynamic code loading from remote

sources, fine-grained security policies, and object serialization). The results of our work include an API that specifies the syntax and semantics of components in the system, a middleware implementation that fulfills the system components of the API, and test results that indicate the overhead introduced by an implementation of our model is acceptable compared to common network operations.

A Java Active Extension (JAE) is a Java object containing code intended for execution at a remote location. An application dynamically loads a JAE to a remote host that is willing to execute the JAE on the application s behalf. The JAE enables the application to use the resources of the remote machine in a manner specific to its present needs (and future needs, if the JAE adapts to changing conditions). The ability to execute code remotely enables an endpoint to deploy an extension that implements customized communication with the endpoint s base.

The machines that execute JAEs are part of an Extension System. An Extension System manages the resources of a set of machines willing to host JAEs. A client of an Extension System must first reserve a set of resources before any of its JAEs can be loaded to the system. An Extension System involves five core components: (1) JAE — A Java object written by the application developer to perform the necessary remote tasks. At runtime, the endpoint loads the JAE to a suitable remote machine for execution; (2) Manager — Resource broker for an Extension System. This component is responsible for servicing resource reservation requests from clients in accordance to an Extension System s policy; (3) ExtensionServer — An execution environment for JAEs. The ExtensionServer represents a set of resources allocated to a client of an Extension System by a Manager. The client may load multiple JAEs to the ExtensionServer for simultaneous execution; (4) ExtensionHandle — An application s link to a loaded JAE. The application may use this object to communicate control data to the running JAE; (5) ExtensionCommunicator — Fulfills the JAE s side of the communication channel by enabling control data communication with the corresponding ExtensionHandle.

Application developers are only responsible for implementing the JAE they wish to execute remotely. Extension System middleware provides implementations of the remaining components.

In addition to executing JAEs, an Extension System provides a rudimentary data channel to address the need to bootstrap the base and extension with information required for more sophisticated communication. The communication mechanism is a queue of messages accessed by the ExtensionHandle in the endpoint s base and the ExtensionCommunicator in the endpoint s extension. Messages are any Java objects agreed upon by the base and extension.

Web Customizers

The Web Stream Customizer (WSC) Architecture tailors the Extended Client/Server distributed computing paradigm specifically to Web browsing. Customizers are distributed web customization software modules that are dynamically deployed and used by clients during a Web session (although servers and even third parties can deploy and use them). Customizers are seamlessly integrated with the basic Web transaction model, simplifying their programming and operation. This is because the WSC system exploits the Web's proxy capabilities, and makes use of standard code mobility mechanisms (with Java as the language of choice given its portability). Thus, importantly, Customizers will work with standard browsers and Web servers, without requiring any modifications to them.

A key feature of the WSC architecture is that it supports cooperative customization at two points along the path between client and server. Many types of customizations require such cooperation and distribution of functionality. For example, data compression (e.g., to reduce bandwidth requirements, and perhaps latency) requires that compressing be done before the data crosses any relatively low-bandwidth links, and that decompressing be done afterwards. To accomplish this, a Customizer is comprised of two components: a *local component* (LC) and a *remote component* (RC). The LC runs on a *Local Customizer Server* (LC-Server), and the RC runs on a *Remote Customizer Server* (RC-Server).

In order to use Customizers, the Web browser is configured to use the LC-Server as its proxy, so that its requests are automatically forwarded to the LC-Server. Thus, when a Customizer is being used, the request passes from the client to the LC, then to the RC, and then to the server (and vice-versa for responses in the opposite direction, from server to RC to LC to client). The LC and RC have the opportunity to modify the request and the response when they are received, before they are forwarded along the communication path.

There will generally be many Customizers, each one being a separate (LC, RC) pair, simultaneously active on behalf of a single client. All of the LCs (for that client) will run on a single LC-

Presentation at DARPA, 3/11/99
Presentation at IDA (Institute Defense Analyses), 4/8/99
Presentation at NSF, 8/8/99
Presentation at DARPA, 2/15/00
Presentation at AFOSR, 4/28/00
Presentation at UC Irvine, 9/30/01
Presentation at DARPA, 1/15/02

NEW DISCOVERIES, INVENTIONS, PATENTS

None

HONORS AWARDS

Elected Senior Member IEEE

Teacher of the Year in Computer Science, UCSD

Date: April 12, 2002

Agency: AFOSR

To: Becky Scott, 0914

Fr: Patti Williams, 0934

Re: Report of Subcontracts

PI: Dr. Joseph Pasquale

Award No: F49620-98-1-0439

Fund No: 31841A

Period: 04/15/98 through 11/14/01

Final or annual? **Final and Annual**

If subk, prime agency name:

If subk, prime award#

If subcontracts in proposal, name of subcontractor:

Please fill out subcontract info; use additional paper if necessary, and **return to me, Mail Code 0934.**

1999	**2000**	**2001**
✓ No Subcontracts	✓ No Subcontracts	✓ No Subcontracts
_____ Yes Subcontracts See Attached	_____ Yes Subcontracts See Attached	_____ Yes Subcontracts See Attached

Final Subcontract Report

✓ No Subcontracts
_____ Yes Subcontracts
See Attached

Becky Scott

Becky Scott