NIREX Internet Resource EXchange

John P. Sanda and Roy P. Pargas
Computer Science Department, Clemson University
Clemson, SC 29631
(sandaj,pargas}@cs.clemson.edu

Abstract

This paper describes NIREX, a dynamic, distributed system that provides the framework for building an open system of services for users over the Internet. NIREX is built on Jini technology. In Jini, a service is any executable code that adheres to the service protocols defined by the Jini specifications. The NIREX architecture is described along with two example applications: one, involving a game-player programming assignment in a computer science class, and another involving sharing of documents by seventh grade math teachers, are provided as examples of how NIREX can be used. The paper concludes with a summary and plans for future work.

1 Introduction

Two observations are the basis for the research described in this paper. First, the Internet provides a powerful infrastructure for communication and exchange of information; however, the amount of information is so large and frequently so fragmented that users are constantly faced with the difficult task of selecting and organizing those items that can be applied in their daily work and lives. This led the authors to the design and implementation of a web-based software system called the NIREX Internet Resource EXchange, or simply NIREX, which offers the user a tool applicable in a variety of web-based information-exchange situations. The second, seemingly unrelated, observation is that a growing number of colleges and universities are requiring incoming freshmen to have laptop computers and to bring them to class. Instructors assigned to teach laptop-required classes are busily modifying their lesson plans, trying to create sound, pedagogical and effective ways to incorporate computers into their lectures. NIREX offers a way for the instructor of a computer science class to give a programming assignment (a game player), that encourages students to submit programs early, test their programs by running them against the instructor's or other students' programs, make improvements, and resubmit.

This paper describes the design, implementation, and application of NIREX. Section 2 details the need among some Internet users for this type of tool. Section 3 presents the NIREX architecture. Section 4 gives the details of a specific application, the NIREX Game Engine. Section 5 describes how the Game Engine will be used and tested in two computer science courses. Section 6 provides a summary and offers possible future work.

2 Problem Statement

The Internet is a wonderful resource, if one can manage the vast amount of information that comes streaming one's way. The Google [9] search engine took approximately 0.28 seconds to provide over 236,000 links in response to a query: "Internet information overload." It can be useful if one has time to sift through the enormous amounts of data provided, but it can also be a bit overwhelming.

Now consider a group of Civil War history buffs, or small business people in a particular town, or computer graph theorists. Each group shares a common interest in a well-defined topic and may enjoy using a simple, transparent, software tool that allows them to exchange documents, still images, video, audio, or shareable code, over the Internet.

In a different context, some colleges and universities now require students to bring to class laptop computers with wireless connections to the Internet. The hope is that the technology will augment teaching and learning. Often, the result is that the instructor is struggling to effectively integrate the computer into the lecture [4]. NIREX was developed with these groups of people in mind.

3 NIREX Architecture

NIREX is a dynamic, distributed system (Figure 1). It provides the framework for building an internet, or open system of services that 1) may be implemented in soft–ware, hardware, or a combination of the two, and 2) may be freely exchanged within and across network boundaries. It is likely that two distinct systems will have different topol–ogies, different hardware, different operating systems, and different runtime environments. NIREX, and Jini in general, make these differences transparent both to developers and to end-users.

The infrastructure of NIREX has been designed and built using Jini technology [12]. Characteristics of Jini-based systems permeate NIREX. First is the concept of *discovery*. Services can dynamically appear and disappear. Clients can "discover" services when they appear on the network, and

they can be notified when services become unavailable. Clients can also be notified when a previously known service reappears on the network.

The second characteristic is the *lookup service* (LUS), a meta-service that provides access to all other services. An LUS can be thought of as a directory or index of available services. When services want to make themselves available to the network, they must "register" with the lookup services. Registering a service involves publishing, or storing, a proxy with the LUS. A proxy is an object that provides functionality for interacting with the service. For example, the proxy may have attributes describing the characteristics of the service, and instructions for communicating with the network. When a client wants to use a service on the network, the client firsts contacts the LUS, then queries the LUS for the sought after service. Once the client finds the service, the client can download and use the service proxy [5].

The third characteristic is the notion of *self-healing*. Many things can go wrong at any time in a network setting, so it is unrealistic to think that problems, such as system crashes, can be completely avoided. Jini addresses these issues through a concept called *leasing*. Jini requires resource holders to obtain leases for their resources, and these leases must be continually renewed. For example, consider a service that is already registered with lookup services and has several clients. Now let's say that some error occurs within the service making it dysfunctional. Clients may have no way of knowing what happened; the fact that the service continues to remain available only serves to confuse the client. With leasing in place, once the service becomes dysfunctional, it will fail to renew its leases with lookup services. After the service's leases expire, lookup services will drop its proxies, thus letting clients know that the service is no longer available. Now that the service has been "unregistered," clients can be notified when the service comes back online [5].

The fourth characteristic is the idea of *zero administration*. Installing and running some new software usually involves copying some files to a local hard disk, and making changes within the file system and/or operating system. Service proxies are analogous to applets in that applets provide an administration-free way to acquire and use an application. So also, proxies provide an administration-free way for using Jini services [5].

In NIREX, the term service is consistent with its usage in Jini terminology. When talking about services in a broad sense, we are not referring to a specific hardware device or software application. Rather, we are referring to a service that is an entity, which provides some functionality to the network, or in this case, NIREX. Services define or expose their functionality through Java interfaces, guaranteeing to provide implementations of all methods listed in the interface.

In Jini, services are always accessed through an object provided by the service itself, called a proxy. Clients download proxies, and know how to use them because the proxies implement a known service interface. Depending on the type of service, a proxy may completely execute on the client machine (as would, for example, a calculator service that performs simple arithmetic operations). Proxies may also carry out some computations on a remote server. Let's say that we have enhanced our calculator service so that it executes a computationally-intensive numerical algorithm. When the client wants to execute the algorithm, the algorithm can be carried out a remote server that is better-equipped for such computations.

NIREX is an internet of services. Although each system that composes this network may vary in size, topology, hardware, and software, there are a few components that a network must have to be considered a NIREX network. These core components are discussed below.

The LUS service is a key component to NIREX. All services must register with known lookup services; clients use lookup services to find services. Every user must have a user account with the NIREX system. Currently, a user account is defined as two strings, one for a userid and one for a password. (This will likely change since the implementation is still in development.) The AccountManager is a core service that manages user account data. An AccountManager handles everything from creating and deleting accounts, verifying userids and passwords, and changing passwords. Since an AccountManager is needed for users to log into the system, it may be common practice for more than one instance of the same AccountManager service to be running on the network.

We intentionally defined a user account loosely, as opposed to defining it as a Java object, a row in a database, or an entry in a file. Doing so gives maximum flexibility to implementations of the AccountManager service. For example, one implementation may choose to use an SQL-relational database for storing user accounts, while another implementation may choose to use JavaSpaces [8].

Every person in NIREX must have a user account. Every NIREX system must provide some means of creating new accounts for new users, and must provide some way for users to obtain the core modules for interacting with the system. Again the specifications here only provide loose definitions to allow maximum flexibility in implementations. The motivation for establishing this flexibility is that we make minimal assumptions about the computing resources available to a given system. For instance, in our current implementation, a web-based registration component is provided via Java Server Pages (JSPs) and servlets [15]. This assumes that clients have access to web browsers. Other systems, however, may not. Consider for example a mobile, wireless network in which all clients run on cell phones. Many cell phones still only provide text capabilities. For this network, we want to provide a registration interface that is well suited for cell phones and cell phone users.

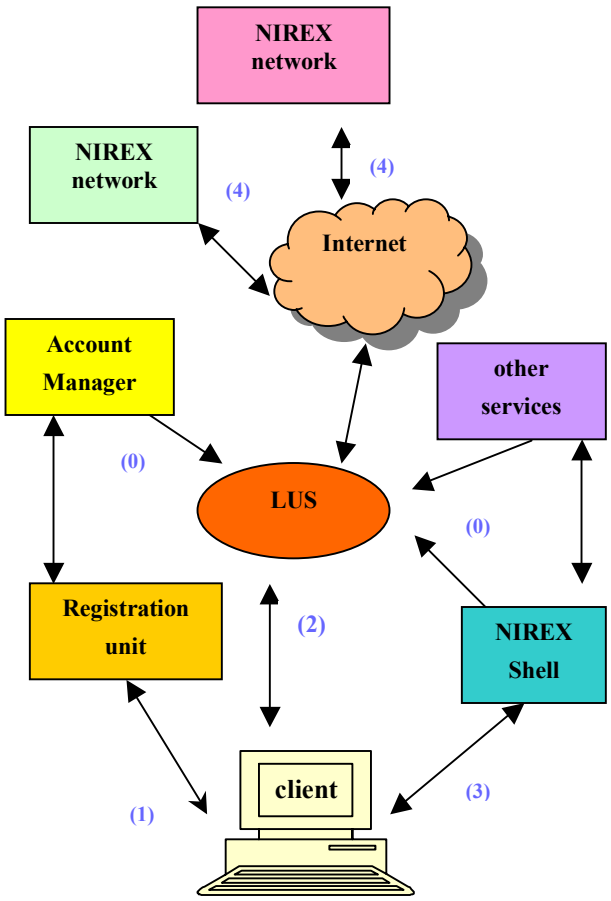

Figure 1. NIREX Architecture. In step 0, the network becomes visible - services come online by registering with the LUS. In step 1, a user creates a new account through the AccountManager. The user will also download and install the core modules at this time. In step 2, the client locates the Shell via the LUS. Then in step 3, the client runs the NIREX Shell to interact with the system. And finally, in step 4, clients interact with services on disparate networks, with network boundaries becoming transparent to the user.

The registration component sets up new user accounts and enables the end-user to obtain the core modules. In our implementation, users can download this component from the NIREX HTTP server.

The NIREX Shell, much like a shell of an operating system, is responsible for providing a way for end-users to interact with the system. The specifications say nothing about *how* the Shell should be implemented. The Shell specifications only define a small number of commands that must be implemented, including: listing available services, loading and running a service, updating a password, and viewing an online help system. Because NIREX may exist with many differing, disparate networks, giving the Shell a loose definition that promotes flexibility is important. On a network consisting of PC's or high-end workstations, a rich GUI Shell is a likely implementation. With our cell phone network example, a text-based interface may be the only suitable implementation.

The NIREX Shell is dynamic. Service developers have the option to write services that either directly implements the Shell interface or some sub-interface of the Shell. When an end-user queries the system through the Shell and finds a service of interest that is itself a Shell-based service, the user can run that service through the currently running Shell without any recompilation or reconfiguration. In this way, the Shell is extremely dynamic; the functionality it offers reflects changes within the system.

4 Applications

NIREX can manage and provide a variety of services: enterprise applications, network management tools, games, and educational programs. These are just a few of the types of applications that may exist in NIREX. For example, it is easy to see that NIREX is an excellent vehicle for assigning programming assignments to computer science students. To demonstrate this, we developed a NIREX service that allows students to submit programs that are evaluated automatically with the results of the evaluation available for the students' viewing. This service is called the NIREX Game Engine.

The NIREX Game Engine is a highly interactive online gaming system. It currently offers games like Nim [3], TicTacToe, and Connect-4. The players of these games are programs written by students. The Game Engine provides a well-defined interface offering methods to load and unload players into the system, to view available players and games, and to play games.

Students can write and compile their player programs and then submit them to the system. Once a player has been entered into the system, it becomes publicly available so that any student may play a game using the entered player. After games have been played, students can view the game results via a web browser.

The Game Engine has several components: a service proxy, a structure for storing players and game results, and the web components for displaying results. The Game Engine is *distributed*. Rather than maintaining a centralized server where all computations are performed, games are played on the client's workstation. (Future versions of the Game Engine may support thin clients in which most operations are carried out on a remote server).

Player objects and game results are stored in a dynamic, shared memory called a *JavaSpace* [8]. A JavaSpace is an instance of a Jini service. It is a shared, network-accessible storage space for objects. Processes use the space as a memory for persistent objects and as a means to exchange objects. To modify an object, a process must remove it from the space, perform the change, and then place it back into the space.

Several features of JavaSpaces make them well-suited for distributed computing in general, and NIREX in particular. First, spaces are *shared*. Any number of remote processes can interact concurrently with a single space. Spaces handle all of the details of concurrent access. Second, spaces are *persistent*. Once an object is stored in a space, it remains there until a process explicitly removes it. Third, spaces are *associative memories*. Objects in a space are located by associative lookup, not by a memory location or an identifier. Fourth, spaces are *transactionally secure*. JavaSpaces technology ensures that an operation on a space is atomic. Finally, spaces allow exchange of *executable content*. While in a space, an object is just passive data, but when a process reads or take an object from a space, it can execute the object's methods just like any other object [8].

We now turn our attention to some of the implementation details of the Game Engine. Students may load player objects into the system. For example, consider a student who has developed a game player. When the student is confident that the program is playing the game properly, the student may submit the executable code into the system. There are several important things to note here. First, a user may have only one player for each game in the system. If a user has already entered a player, it will be removed when a new one is submitted. Second, if the userid of the player object does not match the userid of the student, the Game Engine will refuse to load the player into the system. This is done as a security measure, as well as to ensure the integrity of programs that students are submitting. Once a user has successfully loaded a player into the system, it becomes publicly available and open to challenges by other players.

Users may also unload players from the system. Unloading a player physically removes the object from the system; it is no longer available for use. Students may only unload players that they own (The userid of the player being unloaded must match the userid of the student).

Students may view the list of available players. This list can change from one instant to another since different students may potentially be loading and unloading players at any time. It is important students understand that the results of this operation are dynamic and may change very frequently.

Finally, students can play games. The system will prompt the student to select the game to be played, the players to use, and the number of games to be played. Players are downloaded to the client machine; all games are executed on the client workstation. Once all of the games have completed, the system will report a URL to the user where the results can be viewed.

Although the Game Engine executes almost entirely on the client, its architecture makes it highly scalable. The Game Engine proxy provides a layer of abstraction between the client and the JavaSpace(s). As a result, the system can easily scale from one space to many. This may provide for better load balancing and fault tolerance.

Game results are served up by an HTTP server using Java Server Pages (JSPs) and Java servlets [15]. The web application uses the Game Engine to retrieve results to display them in HTML format. The content of these pages is very dynamic, and likely changes whenever another game has been played. Students can step through individual moves of a game to see exactly what their player did. Hopefully, by providing detailed data, students can discover flaws in their logic and algorithms, and ultimately improve their program and their programming skills.

A second application of NIREX is underway. One of the authors is currently working as an NSF fellow with a middle school (seventh grade) math teacher. The math teacher has expressed interest in building a web-based service that will allow her to exchange documents (math quizzes and tests, instructional materials, PowerPoint presentations, notes, etc.) easily with other teachers in the school district and beyond. The goal of this application is to build a NIREX service that will provide a transparent means for K-12 teachers to share and exchange documents. The system will be built and tested during this (2002-2003) academic year and if the service proves to be of value to the middle school teachers, plans are to provide access to other school districts and to seek a permanent site for the service under the auspices of the South Carolina public school system. In general, NIREX can accommodate many applications that require a web-based exchange of objects.

5 Testing Plans

Lively and interactive programming assignments always seem to excite and energize computer science students. Something about interaction or competition, especially competition with the professor of the course, seems to rev them up [1,2,6,7,10,11,14]. We have had success with the Nim game-player assignment [13], in particular. Students enthusiastically design and develop interesting strategies using a variety of data structures. We expect similar results when the NIREX Game Engine is used in a Connect-4 game-playing assignment this semester.

The NIREX Game Engine was completed this past summer and has been tested on a small scale. More extensive testing will be undertaken in two parts. First, an assignment to develop a Connect-4 game player will be given to a class of 26 sophomore computer science majors taking a CS-4 level course in data structures. (Students in this class are required to bring laptop computers to class.) Possible approaches to developing strategies and the data structures that they require will be discussed in lecture. Laptop exercises demonstrating the strategies will be conducted in class.

Two game players developed by each of the authors will be made available early to provide competitors against which students can test their players. Similarly, students will be encouraged to submit their players early for other students to challenge. In this manner, they can evaluate the results of games involving their players and incorporate improvements into their programs.

On the due date, a three-hour practice session will be conducted in which players will be continually matched randomly with other players and results placed on the Web. The students, working with their laptops, will be allowed to view the results, remove their players, make changes to their players, and resubmit. After the third hour, no additional submissions will be accepted and a final tournament round will be played. A scoring system will determine the top game players of the class.

The three-hour practice session will test the robustness of NIREX under moderate pressure. Students will be loading and unloading games as random parings of game players are made and matches played. At the end of the assignment, a short survey will be given to the students asking them to compare this assignment with other programming assignments they have had in the past. The goal will be to assess the effectiveness of this approach as a pedagogical technique.

NIREX will be tested a second time as follows. The same Connect-4 game player assignment will be given to a class of 25 computer science majors (seniors and graduate students) taking a course in Advanced Java Programming. The architecture of NIREX and the important sections of code will be explained to these students. Their task will be to attempt to overload and crash the system while staying within the parameters of the Connect-4 assignment. In other words, the students will be asked to find ways to bring NIREX down through loading and unloading of game players, requests for matches, and other legitimate use of NIREX functions. The goal is to reveal, and to repair, basic weaknesses in the structure of the code.

Both tests will be conducted in mid- to late October with the results documented by the end of October, 2002.

6 Summary and Future Work

The Internet provides a powerful infrastructure for communication, and exchange of information encoded in a wide variety of formats: documents, audio, video, still images, animated stills, and executable code. A challenge for the user today is figuring out how to select, organize and exchange data that is of use in our daily lives. This paper describes a tool, NIREX, which provides one possible solution. Possible services include enterprise applications, network management tools, games, and educational programs.

Two very different services, one already developed and ready for testing, another in the process of development, are described. The first is a programming assignment to be given to computer science majors involving the development of a Connect-4 game player. Students load and unload their game players into and from the system. Students may then request that matches be played between players submitted. The results of all matches are immediately available on the Web. The second service provides a web-based means for middle school teachers to exchange documents and materials they use in their every day teaching: quizzes and tests, instructional materials, presentations, notes, etc. A goal is to make the underlying system as transparent as possible so that the service, and not the system, is what the teachers see.

References

[1] Adams, J.C. Chance-It: an object-oriented capstone project for CS-1, *ACM SIGCSE Bulletin*, v.30 n.1, (March 1998), 10-14.

[2] Astrachan, O. and Rodger, S.H. Animation, visualiza–tion, and interaction in CS 1 assignments, *ACM SIGCSE Bulletin*, v.30 n.1, (March 1998), 7-321.

[3] Bogomolny, A. *The Hot Game of Nim*. MAA Online. May 2001. www.maa.org/editorial/knot/May2001.html

[4] Campbell, A.B. and Pargas, R.P., Laptops in the classroom, *Submitted to SIGCSE 2003*.

[5] Edwards, W.K. *Core Jini*. Prentice-Hall, New York, NY, 2001.

[6] Fell, H.J., and Proulx, V.K. Exploring Martian planetary images: C++ exercises for CS1, *ACM SIGCSE Bulletin*, v.29 n.1, (March 1997), 30-34.

[7] Fell, H.J., Proulx, V.K., and Rasala, R. Scaling: a design pattern in introductory computer science courses, *ACM SIGCSE Bulletin*, v.30 n.1, (March 1998), 326-330.

[8] Freeman, E., Hupfer, S., and Arnold K. *JavaSpaces Principles, Patterns, and Practice*, Addison-Wesley, New York, NY, 1999.

[9] *Google*. Google. 2002. www.google.com.

[10] Holmes, G. and Smith, T.C. Adding some spice to CS1 curricula, *ACM SIGCSE Bulletin*, v.29 n.1, (March 1997), 204-208.

[11] Jiménez-Peris, R., Khuri, S., and Patiño-Martínez, M. Adding breadth to CS1 and CS2 courses through visual and interactive programming projects, *ACM SIGCSE Bulletin*, v.31, n.1, (March 1999) 252-256.

[12] *Jini Network Technology*. Sun Microsystems. June 2001. wwws.sun.com/software/jini/whitepapers/jini-datasheet0601.pdf

[13] Pargas, R.P., Underwood, J., and Lundy, J. Tourna-ment Play in CS1, *ACM SIGCSE Bulletin* v.29, n.1, (March 1997), 214-218.

[14] Robergé, J. Creating programming projects with visual impact, *ACM SIGCSE Bulletin*, v.24 n.1, (March 2002), 230-234.

[15] *The Source for Java Technology*. Sun Microsystems. Sept. 2002. www.java.sun.com