SFS-AIM

ABSTRACT
The SFS-AIM System is a file system interface to the popular chat service AOL Instant Messenger (AIM), implemented by extending the self-certifying file system (SFS). This provides a convenient way to access the chat service from anywhere, and allows the user to manipulate chat objects using programs designed for standard UNIX files. The SFS-AIM system can also be extended to implement a common interface to a number of different chat protocols to allow cross-protocol communication.

1. INTRODUCTION
With the rise of the Internet in recent years, interpersonal communication clients have become increasingly popular. About 100 million people already used some form of instant messaging (IM) software. By 2004, IM is expected to be more popular than e-mail as a method of communication[5]. As a result, nearly all of the major Internet service providers have developed their own version of an instant messaging system to provide real-time communication abilities to their users. AOL's Instant Messenger, Microsoft's MSN Messenger, and Yahoo's Yahoo! Messenger are just a few examples of the countless instant messaging clients that can easily be found on the Internet today.

Despite their growing popularity, most of these chat protocols are limited solely to the realm of instant messaging, and therefore lack the ability to be used for anything other than real-time communications. In addition, because most chat clients have been designed to meet with the particular specifications of their respective protocols, the different clients are unable to perform cross-protocol communications and thus cannot effectively "talk" to one another. As a result, chatters with friends on different IM networks are often required to establish multiple IM accounts running all the different IM software to access those services.

For our final project, we investigate the possibility of leveraging the convenience and availability of IM along with the flexibility and extensibility of the self-certifying file system (SFS) to create a powerful interface that can be applied to a variety of needs.

We have chosen to use a file system because we believe the extended IM interface should be be simple and transparent enough such that it can be accessed and used from any given system. This would allow the enhanced instant messaging client to be used most widely across an existing application set. In addition, SFS is used because it is open source and can be easily customized to suit our needs.

By extending SFS such that it is able to effectively communicate with a number of different chat protocols, the functionality of the common instant messaging client is greatly increased in a number of easily identifiable ways:

1. Provides a clean and convenient interface to objects that do not typically act like standard UNIX files.
2. User can easily access and manipulate these objects using a file-system interface.
3. Representing these objects as standard UNIX files allows other programs designed for UNIX files to manipulate these objects.
4. Provides a familiar environment for the control of resources that cannot typically be managed by the respective IM protocols.
5. By representing a chat session as a set of virtual files in a directory, we can effectively take advantage of existing UNIX functions – i.e. piping and other text manipulating scripts – providing a greater degree of flexibility and potential for customizing sent messages.

These are only a few of the many advantages of our extended IM interface via the SFS toolkit.

Through this project, we have shown that is indeed possible to extend the instant messaging system such that it can be represented by a filesystem interface. We discuss the design and implementation of this extended IM system and analyze its qualitative performance as it interfaces with a number of already existing applications and tools. Lastly, we will limit our analysis to AOL's Instant Messenger service because of its convenience and the availability of a Firetalk implementation of the AIM/Oscar protocol.

2. RELATED WORK
2.1 Semantic File System
Our project extends the functionality of the self-certifying file system (SFS)[6], by using it to provide a front-end interface to the instant messaging environment.

Another similar system implementation is the semantic file system proposed by David Gifford[3]. The semantic file system provides flexible associative access to the system's contents by automatically extracting attributes from files with file type specific transducers. Like our system it uses virtual directories to provide compatibility with existing file system protocols. The virtual directories in both systems are used to organize files (real or virtual) in a manner based on the system protocol.

2.2 ActiveBuddy

Our system also provides an extension to the AOL Instant Messenger (AIM) service by adding functionality not currently available through a chat service graphical user interface.

Similarly, ActiveBuddy, Inc. has "developed patent-pending technology for creating interactive agents[1]. These agents can deliver a wide range of personalized content and services, including brand and corporate information, on demand via instant messengers."

"The ActiveBuddy technology consists of a suite of tools that empowers companies to leverage the functionality, convenience and appeal of instant messaging according to their needs."

Essentially, ActiveBuddy has provided an alternate interface to AIM which allows it to obtain and query information from a variety of data sources. This technology proves to be highly useful in that a seasoned chat user can comfortably retrieve information from a knowledgeable source without having to deal with the hassle of looking it up his/herself. Instead, the user can simply send a message to an ActiveBuddy Server – which appears as a buddy on their buddy list – and let the ActiveBuddy data interface do the rest.

The ActiveBuddy group has already implemented a number of "bot" applications that utilize their ActiveBuddy interface. Some examples are:

- AgentBaseball - Gets the latest scores, stats and standings.
- RingMessenger - Knows all about the highly anticipated movie "The Lord of the Rings."
- SmarterChild - News, weather, stock quotes, movie times, games and lots more, instantly.

2.3 TNT

TNT is a client for the AIM service using the emacs text editor as a user interface[8]. This provides a certain level of convenience to users who are familiar with the emacs environment. It enables users to utilize the existing emacs keyboard shortcuts and to easily buffer and edit incoming and outgoing messages.

2.4 SIMPLE

"SIP for Instant Messaging and Presence Leveraging," or, SIMPLE, for short, is attempting to establish a uniform means for short text-massaging over the Internet[2]. "With a common standard in place, IMs could easily travel between different SIMPLE networks – just like calls can be carried between Sprint and AT&T phone networks." Furthermore, the SIMPLE interface uses an Internet Engineering Task Force (IETF) standard known as the Session Initiation Protocol, or SIP. This protocol was originally designed to handle much more complex amounts of data such as streaming digital video and voice. As a result, the SIMPLE protocol would have the additional advantage of being scalable enough to handle "digital movies and telephone-like conversations."

2.5 Jabber and Nuance

Jabber.com, Inc. and Nuance have partnered together to provide a voice interface to Jabber's IM software using Nuance's speech recognition engine[7]. This will allow Nuance customers and partners to develop IM applications that utilize voice, and to take their existing voice applications and add IM capabilities. Furthermore, Jabber users will be able to add voice recognition to their implementations.

By combining the technologies of IM and voice-recognition, a user will be able to call in from a standard telephone and be logged into their instant messaging service. From that point on, all spoken dialogue will then be recognized and parsed into text. Likewise, all incoming text messages can then be converted to voice using text-to-speech technology and transmitted back to the user via the phone line.

3. DESIGN

3.1 Overview

The SFS-AIM system has two major components, as shown in Figure 1. The user would access the AOL chat services via the *SFS Client*, using standard UNIX file system commands in a designated directory `/sfs/xxx/aim/username/`, described in more detail in Section 3.3.1. The SFS Client would send these file system commands over the network to the *SFS Server* using SFS RPC calls. The SFS Server interprets the received file system commands according to the semantics described in Section 3.3.2, then sends the appropriate requests to the AIM Server using datagram specifications described in the AIM/Oscar (Open System for Communication in Realtime) protocol. Based on the AIM Server's responses, the SFS Server modifies or lists the appropriate virtual files or subdirectories in `/sfs/xxx/aim/username/` to correctly reflect the executed user commands.

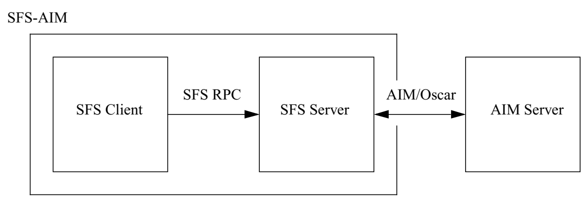

Figure 1: The SFS-AIM System.

3.2 Extending the SFS System

The SFS-AIM system is built by extending a standard SFS system to communicate with the AIM Servers. A standard SFS system already implements the RPC interface between the SFS Client and the SFS Server, and the SFS Server's capability to execute ordinary UNIX file system commands.

Therefore, the chat extension is implemented by only modifying the SFS Server to interpret the file system commands in the designated directory `/sfs/xxx/aim/username/` and to communicate with the AIM Servers.

A modified SFS Server has three main interfaces, as shown in Figure 2. The *Client Interface* listens for SFS RPC calls and issues the corresponding system call. This interface will be modified to correctly interpret the user commands to access the AOL chat services. The *File System Interface* communicates with the underlying FFS via system calls to execute ordinary UNIX file system commands, and remains unmodified from a standard SFS system. The *AIM Interface*, a new extension to a standard SFS system, contacts the AIM Server and listens for incoming messages using the AIM/Oscar protocol.

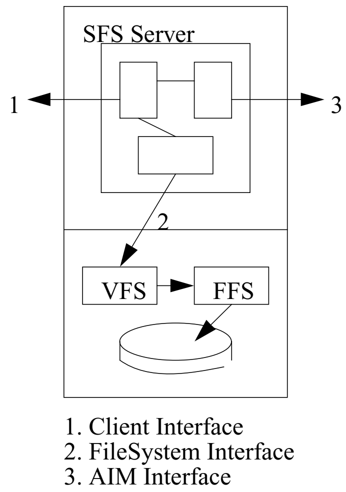

Figure 2: The SFS Server.

3.3 Semantics

This section defines the means by which the end-user can access AIM functionality through the file system interface.

3.3.1 Directory Tree

The AIM features are embedded in the following directory tree.

`/sfs/xxx/aim/` - The root directory for the SFS-AIM proxy. "xxx" refers to the server name/key pair for accessing SFS servers.

`/sfs/xxx/aim/username/` - The directory for an AIM user. Any AIM functions executed under this directory will be executed as "username" when the SFS server communicates with the AIM server. "username" must log in before accessing any AIM functionality under this directory. As such, "ls" will return only one file, "login" until the user has logged in. Once logged in, "ls" returns "logout", "list-setup" and "buddies", which are described below.

`/sfs/xxx/aim/username/login` - Username must log in by writing the appropriate AIM password to this file.

`/sfs/xxx/aim/username/logout` - Username may log out by writing the same password to this file.

`/sfs/xxx/aim/username/list-setup/` - This directory contains subdirectories corresponding to AIM category names (the AIM protocol is able to organize "buddies" into "categories"). Categories can be created and destroyed with mkdir and rmdir operations.

`/sfs/xxx/aim/username/list-setup/cat1/` - Under a given category name (e.g. "cat1"), AIM "buddies" can be added and removed from the user's contact list through mkdir and rmdir operations. Buddies can be moved between categories with the mv operation. A buddy is represented as an empty directory.

`/sfs/xxx/aim/username/buddies/` - All messages sent to and received by buddies are accessed through subdirectories of this directory, which contains directories for each category name, plus a directory named "offline". The offline directory contains entries for every buddy under list-setup that is not currently logged in to an AIM server. Buddies that are logged in appear under the appropriate category name.

`/sfs/xxx/aim/username/buddies/c1/b1/` - This directory represents a user "b1" under category "c1" that is currently logged in. The file creation date of "b1" will reflect the user's log in time and the file modified date will reflect the user's idle time. The contents of b1 are as follows.

`/sfs/xxx/aim/username/buddies/c1/b1/outgoing` - This write-only file will take any data written to it and send it as a message to user b1. If "username" is blocked from messaging b1, or if b1 logs out, a file write error will occur. The reported file size will reflect the number of bytes written to this file thus far.

`/sfs/xxx/aim/username/buddies/c1/b1/incoming` - When user b1 sends a message to "username", it is accepted and stored on the SFS server until this file is read. This read-only file contains all messages received from "b1" thus far. The file will report a file size equal to the length in bytes of all received messages. The last-modified date will reflect the most recent incoming message.

3.3.2 AIM commands

```
LOGIN
cd /sfs/xxx/aim/username
echo "password" > login

LOGOUT
cd /sfs/xxx/aim/username
echo "password" > logout

ADD GROUP
cd /sfs/xxx/aim/username/list-setup
mkdir newgroupname

REMOVE GROUP
cd /sfs/xxx/aim/username/list-setup
rmdir groupname
```

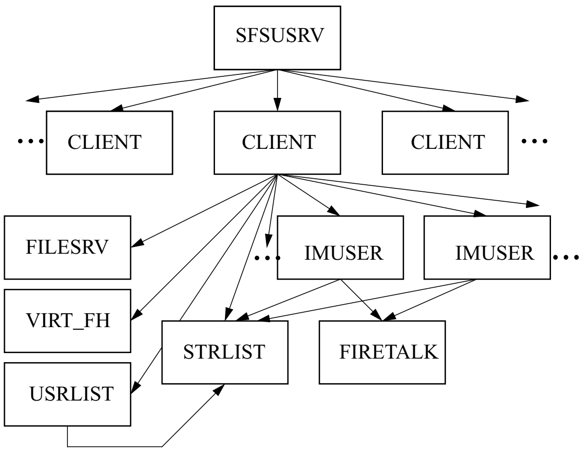

Figure 3: The Code Modules.

```
ADD BUDDY
cd /sfs/xxx/aim/username/list-setup/groupname
mkdir newbuddyname

REMOVE BUDDY
cd /sfs/xxx/aim/username/list-setup/groupname
rmdir buddyname

SEND MESSAGE
cd /sfs/xxx/aim/username/buddies/groupname/buddyname
echo "message" > outgoing

RECEIVE MESSAGE
cd /sfs/xxx/aim/username/buddies/groupname/buddyname
cat incoming
```

4. IMPLEMENTATION

4.1 Overview of Major Modules

SFS-AIM is implemented as an asynchronous, object-oriented system. The SFS server `sfsusrv` launches a new `client` whenever a new client connects to the server. Each `client` accepts and interprets RPC calls from its corresponding client. Since every connection has a separate, asynchronous `client` object, a particular client's crash will be isolated from all other clients, and the server can schedule the asynchronous client requests to maximize its performance.

`Clients` use the other code modules based on actions required by the client requests.

- `Filesrv` is used to communicate with the underlying file system to execute ordinary UNIX file system commands.
- `Virt_fh` deals with the virtual files and directories needed to extend the standard SFS system to a SFS-AIM system.
- `Usrlist` maintains a list of known AIM users, and other information needed to bridge the client interface and the AIM interface.
- `Imuser` is the high level abstraction of all the AIM commands associated with a particular user. `Firetalk` is the actual AIM interface used by `Imuser` to communicate with the AIM server using the AIM/Oscar protocol.
- `Strlist` is a tool used to add to, delete from, and enumerate a list of strings.

4.1.1 *Virt_fh*

An enumeration vContext is used to define the context for a filehandle, the context being the directory or file the filehandle belongs to. There is an enumeration for every possible scenario, from being a non AIM directory to an incoming message file in a buddy within a user's buddy list.

The methods in *virt_fh* form the relationships between a filehandle and its properties, like vContext, user name, category name, and buddy name.

4.1.2 *Usrlist*

This is an intermediary layer which operates without a specific imuser object; its function is to maintain the list of known users and return the applicable imuser object given a string name. There are login and logout methods which take in a user name and password as arguments.

4.2 Client Interface

The client interface is built on top of SFS, the self-certifiying filesystem developed in the MIT Laboratory of Computer Science. SFS avoids internal key management by putting the public keys in the SFS file names, a self-certifying pathname. Since the SFS clients do not have to worry about any kind of security administration, inter-realm file sharing is made trivial. In addition, there are many ways for the client to authenticate servers and in one file system there can be different authentication techniques being employed. SFS also runs in user space and uses an existing network file system protocol to speak to its client. These characteristics give SFS security, extensibility, and portability. Although we could have used another framework like classfs, we chose SFS because we were familiar with its workings.

Modifications were made to the user-level NFS3 server that SFS uses in order to ensure that it could handle RPC's for the AIM functionality. Filesystem calls referencing an NFS-mounted directory are intercepted by the NFS client and forwarded to the NFS server via RPC. The modified NFS server handles the request and returns an RPC response to the client. The handling of the following RPCs were changed in order to accommodate for AIM requests:

- `NFSPROC3_LOOKUP` – Modified to look up virtual directories and virtual files
- `NFSPROC3_READDIR` – Uses *Usrlist* and *Imuser* modules to return the filehandles in a directory
- `NFSPROC3_MKDIR, NFSPROC3_RMDIR` – Used for adding and removing buddies and groups
- `NFSPROC3_READ, NFSPROC3_WRITE` – Used for providing password for logging in and out, in addition to sending and receiving messages.

Since all AIM directories and files are virtual, the user-level server never actually makes file system calls, rather it keeps all of the file and directory information in its *Usrlist* and *Imuser* modules. From the client's point of view, it appears like a normal filesystem because each virtual directory and file has a unique filehandle.

4.3 AIM Interface

4.3.1 Firetalk

The *Firetalk* module communicates directly with the AIM server asynchronously using the AIM/Oscar protocol. It provides functionalities such as logging in and out of AIM, adding and deleting buddies, receiving and sending messages.

Firetalk creates a nonblocking connection with the AIM server for each user. It logs the user on by sending the user's AIM screenname and roasted password (each byte xored with the corresponding roasting string) to the server, as specified by the AIM protocol. *Firetalk* does not wait for a response after sending commands to the server, so that a time-consuming or bad command does not prevent other commands from being issued.

Firetalk also listens for commands and error messages from the AIM server, then issues the corresponding callback function when receiving the signal from the server. The callback functions are established by *Imuser* when a connection is first started; *Firetalk* only facilitates the registration and execution of the callbacks, and not the implementation.

The source code for Firetalk was obtained from [4].

4.3.2 Imuser

Imuser is the top level abstraction for the AIM interface. Each *Imuser* object corresponds to an AIM user, keeping track of the user's connection, username, and password. It also provides all the AIM functionalities for the user, such as sending and reading messages to/from a particular buddy, and listing all his buddies. *Imuser* is used by *Client* to keep track of each user and to perform AIM commands for that user; *Imuser* in turn uses *Firetalk* to communicate the actual AIM commands to the AIM server.

When an *Imuser* object is created, it establishes an asynchronous connection with the AIM server and registers all the callback functions using *Firetalk*. It also logs all received messages for the AIM user, enabling the messages to be retrieved and manipulated later as virtual files.

5. EVALUATION

5.1 Implementation Issues

For the most part implementation was successful; we did not encounter any problems which impeded our progress significantly, and most of the original goals were met. Our only limitation was the `Firetalk` module which did not provide us with all the functionality we needed. Several of the features we tried to implement were not supported by `Firetalk`.

Our intention was to implement categories in our buddy list and list-setup directory so that buddies would be divided into smaller groups. However `Firetalk` did not support any of the functionality for categories, so all the buddies were put into a virtual directory called "all". The modifications made to the user-level NFS3 server took these changes into account by always using "all" for anywhere a category was required.

When adding a buddy to the list-setup directory `Firetalk` did not provide a method to permanently add the buddy to the list-setup. Instead the method provided would add the buddy to the buddy-list for only the duration of the login session. This did not significantly change our goals, since the session features worked as planned. Finally, there was no functionality provided to get the login time of the buddies, so it was impossible to see how long each buddy had been logged in for.

5.2 Tested Functionality

In the original concept of SFS-AIM, we brainstormed several practical applications of the system. The evaluation of our implementation should, therefore, test some of these intended applications.

5.2.1 Test construction

We were able to construct the following test applications. All test scripts were run on `random-beatings.lcs.mit.edu` with the SFS-AIM server running on `blood.lcs.mit.edu` and AIM clients running on various machines.

message echo - bounce back a copy of any received messages. `tail -0f incoming >> outgoing` Alternatively, relay them to a different user. `tail -0f incoming >> ../otheruser/outgoing`

jive/chef - reply to received messages with a translated version of the received message. Jive and chef are yylex-based programs that translate text to "jive-speak" and "Swedish chef-speak". `tail -0f incoming | jive >> outgoing` or `tail -0f incoming | chef >> outgoing`

write - relay incoming messages to a user on the current machine using the UNIX `write(1)` command. `tail -0f incoming | write username`

tcsh - consider incoming messages to be the stdin of a new tcsh shell. Send stdout to the same IM user. IM client now has access to a tcsh shell running on the SFS client. `tail -0f incoming | tcsh -s -i >> outgoing`

5.2.2 Remarks

All of the tests demonstrated that our implemented system was able to be applied in the expected ways. The most impressive was the tcsh pipe. Using this pipe, we were actually able to recompile SFS code and check in updates to our CVS server.

However, in all these cases, we noted a high message latency through the tested system (on the order of seconds). To the extent that we can determine, it appears to be due to buffering of file reads and writes in the programs that read the incoming file (tail) and write the outgoing file (various). Though all of the applications have been effectively demonstrated for our system, the buffering latency greatly undermines its practicality. Our efforts to unbuffer the I/O have been ineffective thus far. This remains the greatest limitation so far in our correct, but slow, implementation of SFS-AIM.

5.3 Possible Applications

There are numerous applications which could utilize the functionality described above. One group of applications would take advantage of the message bouncing functionality in conjunction with a perl script. Users could have a translator script which would return the sent message in another language. Another very interesting application in this category would have a perl script which parsed the sent message and provided a tailored response. Presumably the user would ask a question and receive a response in the same fashion of a psychologist program like "Eliza." The answering script could also be developed so that users could ask questions about the weather, sports, or stock prices and the script would call upon Internet resources to find an answer and reply, similar to the ActiveBuddy agents.

Another different kind of application, though still related to message bouncing, would be a script which would receive the message from the AIM user and forward the message to another user via Zephyr. Any chat client which could have a command line interface developed could be used in this manner.

The potential for applications seems to be enormous, since perl scripts can just be developed to use the incoming message for a different purpose. Although many of the applications of SFS-AIM can be done by other agents, SFS-AIM provides a common and universal interface for ease of use.

6. CONCLUSION

We have implemented a SFS-AIM system that is a file system interface to the popular chat service AOL Instant Messenger. This extension of the AIM service has provided the following benefits:

- Integrate AIM objects, such as buddies and messages, into the file system as virtual directories and files, thus providing a convenient interface to these AIM objects that do not typically act like standard UNIX files.
- Users can then access and manipulate these AIM objects using standard UNIX file commands, such as `ls`ing the buddies directory to see a list of the user's buddies, or writing to a buddy's outgoing file to send a message to that buddy.
- Since the received AIM messages are represented as virtual UNIX files, the messages can be manipulated using programs designed for UNIX files, such as jive and message bouncing. These text manipulation programs can also be used to provide customized messaging.
- Our SFS-AIM system can also be used to control resources, such as sending tcsh commands, using AIM protocols

7. REFERENCES

[1] J. Borland, and S. Olsen. http://news.cnet.com/news/0-1005-200-5726313.html

[2] P. Eng. http://abcnews.go.com/sections/scitech/CuttingEdge/cuttingedge010

[3] D. Gifford, P. Jouvelot, M Sheldon, and J. O'Toole Jr. Semantic file systems. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, pages 16-25, Pacific Grove, CA, October 1991.

[4] I. Gulliver. http://freshmeat.net/projects/firetalk, December 2000.

[5] I. Hardy. http://news.bbc.co.uk/hi/english/sci/tech/newsid_1531000/1531112.s

[6] D. Mazieres, M. Kaminsky, M. Kaashoek, and E. Witchel. Separating key management from file system security. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, pages 124-139, Kiawa Island, SC, October 1991.

[7] K. Newcomb. http://www.isp-planet.com/news/2001/jabber_voice.html

[8] A. Poindexter, et al. http://sourceforge.net/projects/tnt/