



# Fire Service Architecture: A Proposal

Revision 1.0  
June 16, 2004  
Graham Booker  
[gbooker@users.sourceforge.net](mailto:gbooker@users.sourceforge.net)  
<http://fire.sourceforge.net>

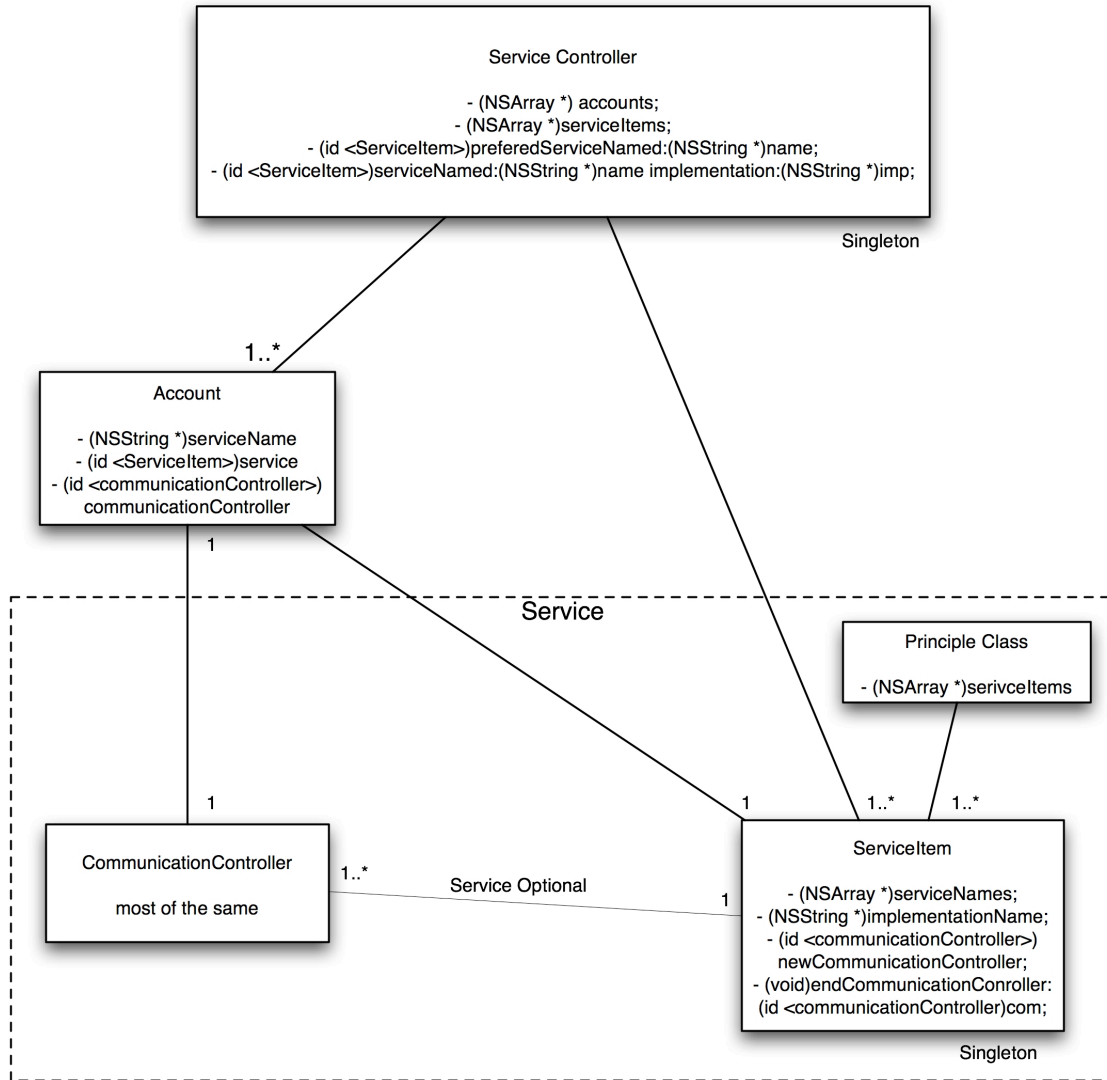
## Introduction

Services comprise the various protocols by which Fire uses to connect to a particular IM or chat system. A service is the collection of code which handles sent and received messages, file transfers, buddy lists, and other features specific to an IM or chat system and translates them into a common form that Fire's internal system can handle. For the purpose of this document, a service will refer to an implementation of a protocol for connection to an IM system or chat. Two different implementations are separate services even if they connect to the same system. Further more, the term front-end will refer to Fire's UI, back-end will refer to the services themselves, and middle-end will refer to everything in between.

Historically, Fire has been limited to a single login at a time for each service. This was further restricted to a single service per service name within the service controller. This was demonstrated with the arrival of the AIM-Oscar service, which has the same service name ("AIM") as the AIM-Toc service. Only one can be used at a time and switching required restarting Fire.

This proposal presents an architecture that can eliminate many of the above restrictions and provide greater flexibility for the future. In this document, I will focus on 4 services to demonstrate the new architecture's abilities. I am not recommending that these services be structured this way, but just showing that they could be. The services are AIM-Oscar/ICQ-libfaim, AIM-Toc, ICQ-libicq2000, and irc. Both AIM-Oscar and ICQ-libfaim use the libfaim library currently present in AIM-Oscar and are the same service, but the service will have two service names. AIM-Toc and irc both use the firetalk library and for the purposes of demonstrating capability, can be combined into the same firebundle. ICQ-libicq2000 is the current ICQ service implementation in Fire. Since the back-end comprises a list of firebundles, much of the back-end lives in its own little world and is allowed to break the rules outlined in this document with the exception of its interface with the rest of Fire.

# The Big Picture



## Global Changes

A few global changes need to be made. The first is the removal of all the shared instance calls to the both the service and communicationController (such calls within a service's back-end are acceptable with a `sharedInstance` on communicationController only if multiple logins are not supported by this service). The second global change is all references to the serviceItem are removed in favor of references to an account or a service name. Removing references to the service can also help eliminate bugs where a buddy loses its service association when a bundle fails to load. The only classes in the front-end and middle-end allowed to maintain a pointer to a serviceItem are the ServiceController and the accounts.

## Firebundle Principle Class

The principal class within any `NSBundle` is the first to be examined when the bundle loads. Currently this is the service's `serviceItem` implementation, whose instance is retrieved by the `sharedInstance` class function. I propose changing this so the class is only required to implement one class function:

```
+ (NSArray *)serviceItems;
```

For most of the current firebundles, the principle class can remain the `serviceItem` and this function added which simply returns an array containing a single object, an instance of that class. Since AIM-Toc and irc both use the firetalk lib, they can be combined into a single firetalk bundle, but the service protocols themselves are so different they cannot be easily combined into a single service. In such a case, the principle class will simply return both services in this array.

## ServiceItem Design

The `serviceItem` will change to a protocol that defines basic service interaction. Additional calls internal to the service may be added. The first major change to note is the removed of the numerous `supports...` calls and replacing them with a single

```
- (BOOL) supports:(id) key;
```

This allows addition of later keys without much difficulty. Another added selector is one that returns an array of service names:

```
- (NSArray *)serviceNames;
```

This is added with ICQ-libfaim and AIM-Oscar in mind. All of the account interaction has been removed from the `serviceItem` and a new function:

```
- (NSDictionary *)accountKeys;
```

has been added. This will be addressed further in the Account Design section later. The emoticons have been removed in favor of the theme system. The final change is the addition of two selectors:

```
- (id <CommunicationController>)
newCommunicationControllerForServiceName:(NSString
*)service;
- (void) endCommunicationController:(id
<CommunicationController>)communicationController;
```

The selector `newCommunicationControllerForServiceName` is a request that will be made by the account for an instance of this service's `communicationController`. The service may return a `sharedInstance` in the case where it only supports a single login, or may return a new one. The `endCommunicationController` selector is also called by the account, but after it has logged off and is about to release the `communicationController`. This function is to let the `serviceItem` know that the `communicationController` is no

longer going to be used. Many services will use this selector to remove the communicationController from lookup tables that are used by its c functions.

## ServiceController Design

Much of the service controller can remain the same since it is mostly a container for the serviceItems. The only real big change is serviceNamed will be renamed to

```
- (id <ServiceItem>)preferredServiceNamed: (NSString *)serviceName;
```

and a new selector

```
- (id <ServiceItem>)serviceNamed: (NSString *)serviceName implementation: (NSString *)implementationName;
```

is added. Service controller will maintain the full list of services as well as the preferred service implementation name for a particular service name.

## CommunicationController Design

The communicationController will largely stay unchanged. The most notable change is the removal of service specific items. These calls were the result of menus. Instead, the communicationController will be given an opportunity to add menu items via a selector. It can set the target and selector for these menu items in the back-end and thus keep it hidden from the rest of Fire.

## Account Design

The account is the largest change proposed in this document. All account classes within services will be eliminated in favor of a unified account class. This account class supports transactions (storing buddy list changes while offline to be replayed when back online) and stored them along with the service name, preferred implementation name, and a dictionary of persistent properties with the following access functions:

```
- (id)propertyForKey: (id) key;  
- (void)setProperty: (id)prop forKey: (id) key;  
- (NSArray *)propertyKeys;
```

The dictionary, accountKeys, obtained from the serviceItem tells the account editor which keys in this properties dictionary correspond to common items such as server, port, username, password, mail alerts, and display name. Thus, by changing service implementations, these values may change. Non-existent keys in the keys dictionary will result in a disabled field in the account editor. This is used for keys that don't make sense, such as server for Rendezvous.

All changes to the buddy list, which formerly went through the communicationController, will now go through the account. The reason behind this change is that such calls can be made whether the account is online or not. In the case where the account is online, it will simply pass on the call to its communicationController, but if it is not online, it will record a transaction. The functions for this are as follows:

```
- (void)addDeleteBuddy:(BuddyItem *)buddy;
- (void)addAddBuddy:(BuddyItem *)buddy;
- (void)blockBuddy:(BuddyItem *)buddy;
- (void)unblockBuddy:(BuddyItem *)buddy;
- (void)moveBuddy:(BuddyItem *)buddy fromGroup:(NSString
*)oldGroup toGroup:(NSString *)newGroup;
```

## Sample Bundling

The four services mention in the Introduction can be bundled in the following manner:

Bundle:	Service(s):
Faim	Faim
ICQ2000	ICQ2000
Firetalk	Firetalk-AIM and Firetalk-irc

The services can have the following service and implementation names:

Service:	Service name(s):	Service Implementation Name:
Faim	AIM and ICQ	libfaim
ICQ2000	ICQ	libicq2000
Firetalk-AIM	AIM	libfiretalk
Firetalk-irc	irc	libfiretalk

## Sample Processes

### Login Process

1. Account is told to login
2. Account checks internal preferences on service implementation and requests serviceItem from ServiceController.
3. Account requests a communicationController from ServiceItem. If nil, fail and post error (single login per service most likely).
4. Retain communicationController
5. Tell communicationController to login.
6. The remainder of the login process mostly goes the same as it has before.

### Logout process

1. Account told to log off.

2. Account tells communicationController to log off. CommunicationController closes all connections and shuts down services.
3. Account tells ServiceItem to endCommunicationController.
4. Account releases communicationController and sets it to nil.