MYRIO™

Myrio Client Architecture

Version number 1.3

Created: 10/2/2000
File Name: Myrio 2 Client Architecture
Created by: Mark Martino

Abstract/Executive Summary

This document describes the architecture of the software client which runs on Myrio set-top boxes.

Revision History

Description	Author	Date	Version
Initial writing	Mark Martino	9/25/2000	0.1
Inserted Figure 1	Mark Martino	9/26/2000	0.2
Added class design section	Mark Martino	9/27/2000	0.3
Made changes from reviewers	Mark Martino	9/28/2000	1.0
Updated Fig. 1; added Control section	Mark Martino	10/2/2000	1.2
Adde Fig. 2; rewrote section 5	Mark Martino	10/3/2000	1.3

Approvals

Name	Approval	Date	Version

Table of Contents

1 Introduction

1.1 Purpose

Read this document to understand the Myrio client, which runs on Myrio set-top boxes, and how it communicates and interacts with the Myrio server. Use it as a guide when designing software submodules for the client.

1.2 Scope

This document describes the Myrio software client, its major software modules, how those software modules interact, and how the client interacts with the Myrio server.

This document does not include implementation issues such as computer language, compiler, or operating system details. It provides enough information to select an appropriate programming language, and design the software components in the chosen language at a level low enough to be coded.

2 Myrio System Activities

2.1 Actions: A Way to Define and Understand the Client Modules

Before explaining the design of the modules in the Myrio client, we describe the actions the client, the user controls, and the server perform to provide Myrio users with the features offered by the Myrio product. These actions describe how the client interacts with the server and the user controls. What the client does internally is discussed in the sections after the modules are defined.

It is critical to define these actions with the appropriate level of granularity. By "appropriate" we mean a level high enough to avoid implementation details, but low enough to clearly place the action exclusively on the client, the controls, or the server.

The actions are presented as five groups, one for each of the Myrio services available to the user and one for managing state. Since this discussion is centered on the client, actions directed to the server are referred to as outgoing and symbolized by the left pointing angle bracket (<). Actions coming from the server or the controls to the client are symbolized by the right pointing bracket (>).

2.2 Action Assignment Guidelines

These are the guidelines used to assign actions to the appropriate entity:

1. Although a server has more computing power when compared to a client box, it does not have more than all of the client boxes combined. Therefore, computations that happen regularly and often, that is more than every few minutes, should happen on the client.
2. For reasons similar to item (1), computations specific to one client should happen on that client.
3. Data distribution or computations which happen regularly, but not very often should be assigned to the server.
4. The bandwidth taken up by the video stream is large in comparison to other data exchanged between the server and client, therefore the amount of data exchange should not be a factor in determining assignment.
5. For security and database stability, database access should be performed by the server.

2.3 Action Group 1: User Controls

2.3.1 > Accept and interpret button presses from the remote control.

2.3.2 > Accept and interpret standard ASCII key values from the keyboard.

2.3.3 > Accept and interpret non-standard, that is, "hot key" values from the keyboard.

2.3.4 > Accept and interpret button presses from the control panel of the set-top box.

2.4 Action Group 2: Digital TV Service

2.4.1 < Join an IP Multicast group on the server to get a video stream for a particular channel.

2.4.2 > Accept and display the video stream and information overlay for a particular channel.

2.4.3 < Send the client a request for the information for a particular channel.

2.4.4 > Accept, format, and display the information for a particular channel.

2.4.5 < Send the client a request for information on a range of channels by channel range.

2.4.6 < Send the client a request for information on a range of channels by time period.

2.4.7 > Accept, format, and display the information for a range of channels.

2.4.8 < Send the client a video control command.

2.5 Action Group 3: Video On Demand Service

2.5.1 < Send the client a request for a complete video title.

2.5.2 > Accept and play the video stream for the title.

2.5.3 < Send the client a request for the trailer of a video title.

2.5.4 > Accept and play the video stream of the trailer.

2.5.5 < Send the client a video control command.

2.5.6 < Send the client a request for a list of titles from the movie library.

2.5.7 > Accept, format, and display the list of movie titles.

2.5.8 < Send the client a request for information about a specific title.

2.5.9 > Accept, format, and display the information about a title.

2.5.10 < Send the client a request for information using a filter.

2.5.11 > Accept, format, and display the filtered information.

2.6 Action Group 4: Web Service

2.6.1 < Request connection to web server.

2.6.2 > Accept connection and display browser.

2.7 Action Group 5: Assist Service

2.7.1 < Send the client a request for a help screen.

2.7.2 > Accept, format, and display the help information.

2.8 Action Group 6: Myrio.i.02.05 State Management

2.8.1 < Send request to server for set-top box authentication or verification.

2.8.2 > Accept and interpret authentication or verification verdict.

2.8.3 < Send request to server for subscriber authentication or verification.

2.8.4 > Accept and interpret authentication or verification verdict.

2.8.5 < Send request to server for client software update.

2.8.6 > Download client software update.

2.8.7 > Accept request from server for one or more set-top box state properties and their values.

2.8.8 > Send one or more set-top box state properties and their values to the server.

3 Myrio Client Modules

The actions defined in the previous section are performed by these modules. These are modules and not classes. Although it is likely most of them will become classes, it is also likely they will exist in a class hierarchy with classes above and below them along with supporting and helper classes. They are not packages or libraries because they are not meant to indicate how to bundle code together.

The following definitions describe the purpose of the modules and hint at what some of their internal methods might be. At this writing there are four service modules, but this architecture is designed to allow for adding new service modules easily without having to make many modifications to the non-service modules.

One of the guidelines to using this architecture is to use events and event listeners to communicate between these top level modules. This decouples the modules and aids in building a multi-threaded implementation. It means any module can communicate with any other module by listening for its events. It also means a module can be activated and deactivated by directing it to turn off all of its listeners except the one which turns it back on.

3.1 ClientMessenger

The ClientMessenger provides a common way for the other modules to communicate with the server. At this writing that communication will probably be some form of XML over HTTP. This module provides a common communication interface, common error handling, common header data, and perhaps some common data translation. It is likely it will need to be able to provide both synchronous and asynchronous methods both on the server side of it and on the internal client side of it.

3.2 UserDeviceHandler

This module implements the actions in Group 1. It accepts user input from the keyboard, remote control, and buttons on the set-top box. It converts the raw key and button presses into action events the rest of the system can listen for and respond to.

For instance, if the up arrow key is pressed on the remote or it the up toggle button is pressed on the remote or the front panel of a set-top box, an action event called something like GO_UP is generated. GO_UP is then received by an active module and could be interpreted a number of ways. It could mean to increment the channel to the next highest one. It may also indicate moving the cursor up on the screen or up to the some visual cell on the screen.

A special event needs to be available to pass key values through to modules using an external window as is the case in for a web browser.

3.3 ServiceManager

This module assists in performing actions in Group 2 and Group 3. The main purpose of the ServiceManager is to determine which service should be activated. It then turns on the service that needs to be activated, and turns off the previously active module. By active, we mean receiving and responding to events.

3.4 DTVService

This module implements the actions in Group 2. It provides the video feed, channel guide, and channel information overlay. It can also use supporting objects to provide virtual VCR features.

3.5 VODService

This module implements the actions in Group 3. It provides the video feed, movie guide, and movie information display. It can also use supporting objects to provide virtual VCR features.

3.6 WebService

This module implements the actions in Group 4. It can generate an instance of a default browser or find a specific kind of browser and instantiate it. The browser is initialized, started, and presented to the user.

3.7 AssistService

This module implements the actions in Group 4. It can generate an instance of a default browser or find a specific kind of browser and instantiate it. The browser is initialized, started, and presented to the user.

3.8 PersistentDataManager

This module implements the actions in Group 5. The main purpose of the Persistent Data module is to upload information about the user and the set-top box when queried by the server. It may also own daemons to gather information at scheduled times and to push the data up to the server.

4 Client Module Control and Communication

There are advantages to using an event driven means of communication and control for the top level modules of the client. One advantage is flexibility. Once an event object and a listener object are coded, it is easy to attach them to any of the modules. Another advantage is distributed control. No one module needs to know a lot about the other modules. This in turn allows us to add new services with little impact on existing services.

One small disadvantage is drawing diagrams for system level interactions. Since any module can talk to any other module, there can be a lot of arrows flying around the diagram. To get around this, we only attempt to diagram the most important interactions and we divide these interactions into control and communication.

4.1 Module Control Flow

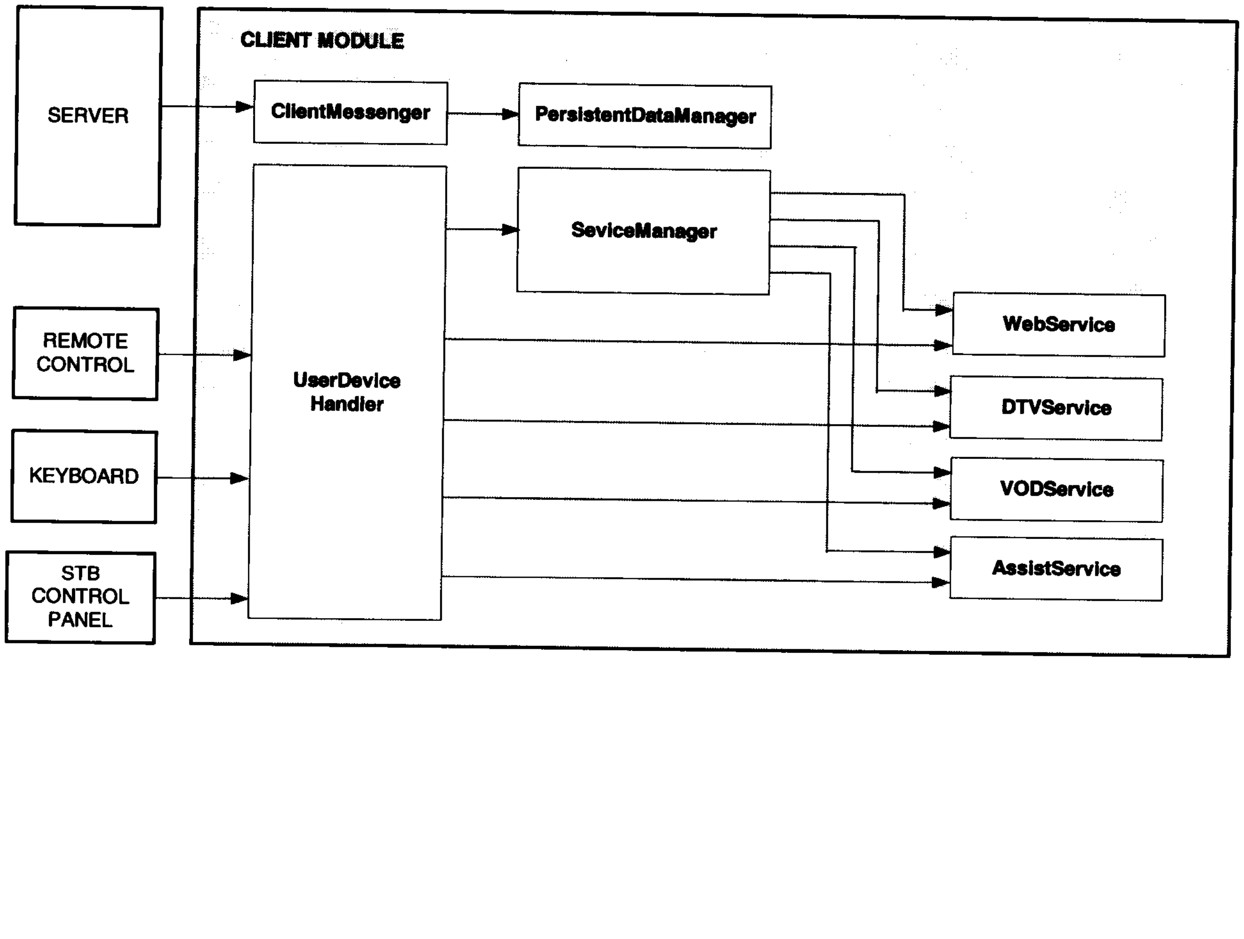
The flow of control in the client is mostly one way. Events from the outside world propagate through the modules, the user sees the results of what the modules produce. The user then triggers one or more new events which start the flow again. This flow is depicted in **Figure 1.**

In a typical scenario the user clicks on the remote button to select digital television. The UserDeviceHandler translates the button press event from the remote control into a service selection event. The ServiceManager is listening for such an event. It posts events to select the DTVService and another event to deactivate the currently active service.

The active service is listening for the event and turns itself off. That is, it hides its display if necessary and stops listening for all events except the activate event. In the meantime, the DTVService responds to its activate event by building an overlay of information about the current channel, switching to the video stream for that channel, and displaying the overlay of the information about the channel.

The server can send a command through the ClientMessenger to the PersistentDataManager to gather some statistical data from the other modules or to retrieve some stored data about the system.

Figure 1. Client Module Control Interactions



4.2 Module Communication

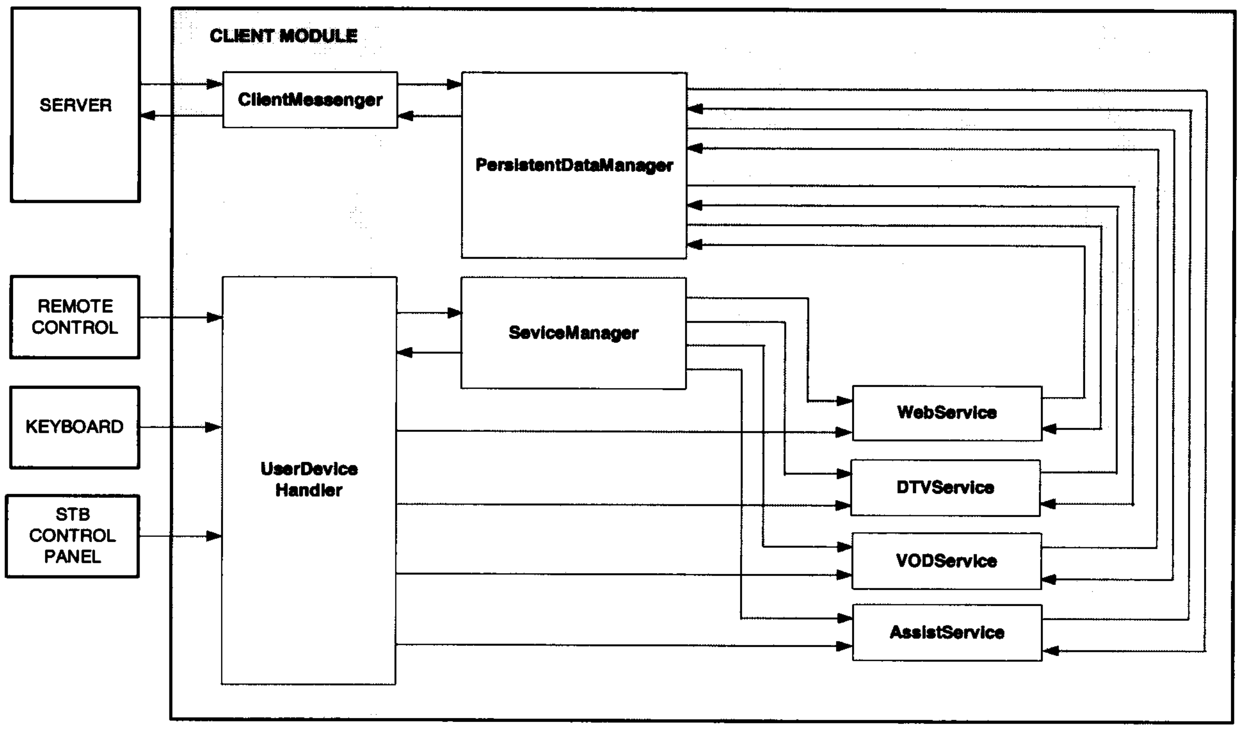
This section describes the flow of communication through the modules in the client. This flow is the preferred path of communication. There are situations where a module may deviate from this flow in order get information from a module with which it does not normally communicate. As much as possible, adhere to the flow described here to help make the client easier to understand, debug, and maintain.

This communication flow is shown in **Figure 2**. Starting in the upper left corner, the server can send instructions and information to the client through the ClientMessenger and the client can return information about itself through the ClientMessenger. All of the persistent information about the system is gathered and stored in the PersistentDataManager. Persistent is a relative term. In some cases the data may exist in volatile memory for the length of a TV program or for the duration of a session. In other cases the data is stored in its own file on the hard disk. The PersistentDataManager may even calculate some statically data and return it to the server. As shown, all of the services can send data to this manager and all services can query it.

In the lower left quadrant of the diagram the flow of communication from the user controls is the same as it is in the control flow. There is no way for the UserDeviceHandler to change how the controls behave. Any UserDeviceHandler events which are commands to change the services go the ServiceManager. The UserDeviceHandler can query the ServiceManager to find out which services are active.

All other UserDeviceHandler events are accepted by any active service. If the action triggered by the event warrants it, the service can send information to the PersistentDataManager.

Figure 2. Client Module Communications



5 Conclusion

With this model of the Myrio client, a programming language can be selected and the software components to implement this model can be designed and coded. Some modules, like the UserDeviceHandler may fit neatly into one component. Some modules may turn out to be one component with supporting components within or helper components used externally. It is probably good to create a common base component for the services and a generic user input device component.

6 References
