Using the Styx Network Protocol in Event Driven Systems

M. Shannon, M. Freeman, C. Bailey
Dept. Computer Science, University of York, UK
(marks, mjf, chrisb)@cs.york.ac.uk

1. Introduction

If a system of connected devices is to be useful then there must be the capacity for devices to inform other devices and software objects of changes of state and other events. The Styx protocol allows a collection of remote devices to appear as a unified file system, either as a single file or a directory tree, but event handling is not covered, so we must add an event facility.

A series of discrete events can be viewed as an event stream, which can be implemented as a read-only file of seemingly infinite length. To observe a stream of events, a client needs to keep reading from the file that represents that stream. Each time an event occurs the client will receive more data and can then respond accordingly. When a client is no longer interested in an event stream, it merely closes the file.

2. Overall Design

2.1. Implementing event streams

So that clients can easily locate an event stream, all event sources should be provided in standard locations. We propose to collect them under the /event directory. For example a door being opened might cause an event, the file stream for which could be called /event/door/open. We do not specify the format of each event as this will be specific to the event type.

The Styx protocol implements the process of reading a file by a series of read requests, the replies to which contain the file contents in a number of chunks. Events can be sent in the form of replies to read requests previously made by the client. Thus at a file system level, the events appear to form a single stream, but event handling latency is minimised, as only a single message is required for event propagation.

2.2. Multiple clients for a single event

As an event stream is represented by a file, the simplest implementation would be for any client that is interested in a stream to simply open that file. Unfortunately the Styx protocol forbids a file from being opened by more than one client at a time, so in order to provide event streams to multiple clients a stream splitter or 'multipipe' is required. For each client to be assured of a unique file, a two stage process is required, firstly the client reads from the event file to get the file name of the event stream. It then reads events from the named file.

For a simple event generating device, the contents of the event file would be fixed and contain the name of the event stream. A multipipe implementation would generate a new name every time the name file was opened. This virtual name would map to a new multipipe output.

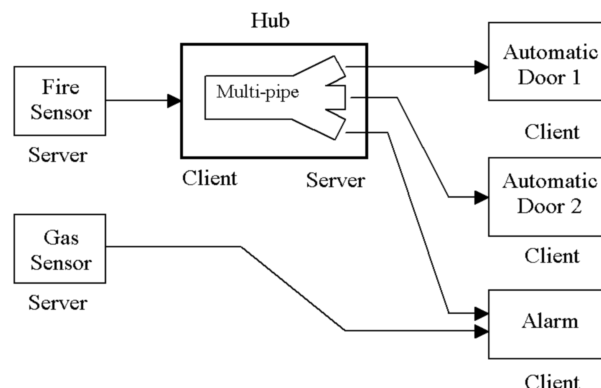

Figure 1 Example system architecture

Figure 1, shows a possible scenario where an infra-red fire sensor triggers the closure of automatic internal fire doors, unlocks external doors and sets off an alarm. In this example the fire sensor is attached to a central hub, which implements the required multipipe, allowing the ‘fire’ event to be forwarded to the attached client. If only peer to peer associations are required a direct connection can be made e.g. the gas sensor and alarm objects.

The sequence for connecting Automatic Door 2 to the fire sensor could be as follows:

1. Open Server:/event/fire for reading
2. Read contents of Server:/event/fire, this will give a file name, for example /pipe/fire/out2
3. Open the named file (Server:/pipe/fire/out2) for reading, creating it if necessary.
4. Close the event file (Server:/event/fire).

2.3. Hubs

For the collection of devices that form a system to be able to communicate with the rest of the world a hub, which unifies all devices into a single namespace will be required. This hub can also unify events under a single /events directory so that all event streams provided by the component devices are visible to external clients. The hub can also provide the multipipe functionality, so that individual sensors can be kept simple. Note that the hub is itself a device and can form a hierarchy with other hubs, any hub can serve as root, indeed each hub sees itself as root, even if it is part of a hierarchy.

2.4. Performance

Since the Styx device will only have to supply messages for events which are being actively sought, this protocol is efficient and should work on low-bandwidth and thus low-cost, networks. The only overhead of adding events to Styx enabled sensors, is the requirement to hold read requests indefinitely and the additional state machine needed to determine whether to send a message or not; no additional message parsing or generation logic is required.

3. Hardware implementation

The Styx communication protocol is part of the Inferno operating system[1] allowing external devices to appear as part of a unified file system. The Inferno operating system can be ported to small embedded systems[3], requiring only a few 10Kbs of memory. However, even this compact implementation may be excessive for very small embedded systems i.e. the processor, ROM and RAM are only used to implement the Styx protocol. For very small embedded applications a direct hardware implementation is more appropriate, minimising the cost and power requirements of these nodes.

An initial prototype system has been implemented in a Xilinx field programmable gate array[4] (FPGA). When designing this system hard limits must be placed on the protocols functions and data used. However, these limits must not invalidate the Styx protocol i.e. a hardware implementation must be interchangeable with a software implementation. The hardware core used in this work is based on an existing IP-Core[5], modified to meet our requirements; file name and subdirectory size increased (12 character, 256 folders), addition of the Styx *stat* and *flush* commands. Also added is the *event* file mechanism type, an infinitely large file that reports the occurrence of a specified event. The final modification is to replace application specific IO ports e.g. byte, stream and RAM, with a generic system bus to produce a Styx bridge IP-Core, as shown in figure 2.

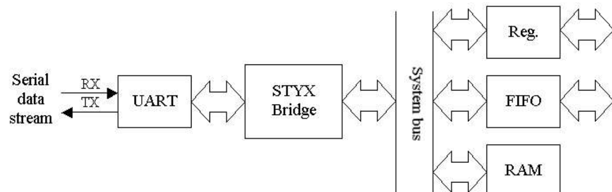

Figure 2 Hardware architecture

This architecture combines these interface types into a single, expandable bus structure. At present a simple ad hoc bus structure is used, with simple byte addressed read and write operations, implementing the required IO ports. However, this can be easily expanded to support the Wishbone[6] or CoreConnect OPB[7] (using Xilinx's OPB master VHDL template) bus standards, allowing a range of standard memory mapped peripheral devices to attached.

Styx is an application layer protocol and can run on top of standard protocols e.g. TCP/IP, PPP, UDP etc, or directly across a peer to peer link e.g. RS232, RS422 etc. For the purposes of this work we are using serial to Wifi modules[8] and UDK communication modules[9] (short range radio and infra-red). Both of these modules provide serial data streams that can be easily interface to the FPGA hardware IP-Core.

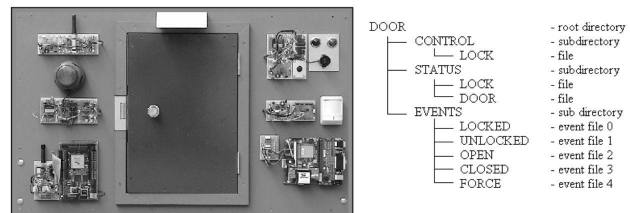

Figure 3 Door case study

4. Conclusions

We are currently finishing development of the hardware and software components needed to construct the scenario shown in figure 1. This will be implemented on our existing door development model, shown in figure 3. In this example the door is represented by three subdirectories: control (write only actuators), status (read only sensors) and events. Future work will focus on developing this IP-Core for different buses architectures and optimising its implementation for embedded application.

References

[1] Vita Nuova, (2006), Web : http://www.vitanuova.co.uk
[2] Multipipe, (2006), Web : http://multipipe.sourceforge.net/
[3] W. Wong, (2000), Inferno Operating System Burns Its Way Into Embedded Systems, Web : http://www.elecdesign.com/Articles/Index.cfm?AD=1&ArticleID=4575
[4] Xilinx, (2006), Web : http://www.xilinx.com
[5] N. Audsley, R. Gao, A. Patil, "Styx SoC: Ease of Interoperability between Ubiquitous Devices", Proceedings of 4th International Workshop on Real-time Networks, 5th July 2005, Palma, Spain
[6] Wishbone, (2006), Web : http://www.opencores.org/
[7] IBM, (2006), CoreConnect bus architecture, Web : http://www3.ibm.com/chips/techlib/techlib.nsf/productfamilies/CoreConnect_Bus_Architecture
[8] Amplicon, (2006), Web : http://www.amplicon.co.uk/dr-prod3.cfm/subsecid/10136/secid/4/groupId/12289.htm
[9] M. Freeman, (2006), Web : http://www.cs.york.ac.uk/amadeus/