PLATform Design Whitepaper

William Nagel

July 16, 2004

This document is a work in progress. I am going to attempt to give an overview of what PLATform is, and architecturally how I think it should work at a high level. I'm hoping that this will spark some good discussion on the mailing list[1] that will further define the system.

0.1 System Overview

The name PLATform comes from two acronyms scrunched into one. The first acronym, *PLAT*, is historical, and really no longer applies. So, I'm going to leave things at, "it's an acronym" and not go in to exactly what it stands for, to avoid confusion over the direction PLATform should take. The second acronym, *form*, is where the real meat of what PLATform is lies. The name *form* stands for function oriented routing management.

The basic goal behind PLATform is to develop a set of network services that will allow network components to route data to each other based solely on statements of the functionality that a device or service supplies and/or needs. For example, lets look at a home control network.

In the typical home control network, you would have a controller, which might reside on a PC. Additionally, you may have controllers in each room to turn on lights, and thermometers that can feed back the temperature in each room, and a thermostat that can control the temperature. Traditionally, the controller would discover the other devices on the network (either through some sort of automatic protocol, or via manual entry by the user), and maintain references to each device. The interface would them provide a means for the user to send or receive data from one or more devices at a time, in order to control the system.

The limiting factor, flexibility-wise, in the system above is that the controller needs to specifically address each device (or multiple devices simultaneously via multicast—the important thing here is that the controller is directly accessing the device).

Under the PLATform model, each device will register information about itself with routing services resident on the network, providing information about data that it needs, and data it can provide. For instance, some of the home light controllers in the example above could register that they are capable of taking light levels for the living room, and the thermostat could register that it requires temperature data (which the thermometers would register that they provide). Then, the controller doesn't need to know anything at all about the network. Instead, when the user tells it to turn

[1] platform-devel@lists.sourceforge.net

the lights in the living room to 50%, the controller just tells the PLATform network services that it wants the lights at 50%, and it tells everything that has requested to know that information that the lights should go to 50%. This would allow other devices that the controller doesn't even know about to act appropriately. For example, the stereo system could be set to start playing Barry Manilow when the lights go to 50%, since it probably means a romantic scene is being set.

0.2 Architecture

The current implementation of PLATform is honestly pretty bad, but it does give us a point to start discussions from. The current implementation was designed for a specific control system, and some of its implementation is tied into that, or at least makes some assumptions based on that. There are also some components of the system that are integrated into proprietary code that we have not released. That may get released if it's useful at some point in the future, but likely not until then, since it's integrated into proprietary code that is definitely not getting released.

The basic architecture of PLATform is currently built on top of CORBA, and centers around the "Deliverable Coordinator." When a device connects to the PLATform network, it uses the CORBA NameService to locate the deliverable coordinator. It then registers with the coordinator all of the deliverables that it requires and/or provides. Registration comes in four forms:

- Deliverables that the device provides by pushing them to the network
- Deliverables that the device provides in response to a request from the network
- Deliverables that the device requires whenever another device pushes them
- Deliverables that the device requires when it requests them

Currently, deliverable payloads are sent in an XML form, but there is no real requirement for that in the system.

The current implementation of the deliverable service uses the CORBA notification service for its data routing.

0.3 Design Goals

At this point, I think that PLATform needs a complete re-design to realize its potential. However, I think the current implementation holds a lot that can be used as a starting point for that redesign.

I think the first phase of the design needs to get down what we really want PLATform to do. Here are a couple of the design goals that I can think of off the top of my head:

- Decoupled routing of deliverables with both a push and pull mechanism
- Automatic discovery and registration of devices
- Quality of service features (priority, timing, etc.)
- Necessary service programs that can be run on the network
- Library code for clients to use when connecting

I'm sure there's also a lot of other stuff that I'm missing, but that should give us a starting point for discussion. This whitepaper is meant as a request for comments, to get discussion on design for PLATform started. If you have any comments, please send them to the development mailing list, mentioned in the beginning of this paper.

This document will almost certainly be modified as discussion on PLATform gets going. I'll keep the latest version of the document available on the web site.[2], and will probably periodically post changes to the mailing list.

[2] http://platform.sourceforge.net