Winnipeg Transit EBDD RFP Technical Appendix Controller and Communication

2008-09-16

1 Controller Logic

Winnipeg Transit will build a back-end application to service information requests from the display controllers. This section outlines a potential controller/application server communications protocol. The exact format of the messages (e.g. encoding, compression, etc) can be discussed at a later time. Indeed, modern application design dictates that the underlying message protocol is to be abstracted out of the application's core logic, which isolates behaviour logic from data access logic.

This proposal describes a pull model for a display controller consisting of two subsystems. These subsystems would ideally be implemented as independent processes spawned from a main boot/monitoring process.

1.1 Subsystems

The first subsystem is the *system clock*, which is responsible for keeping time internally and feeding time to the display. This subsystem periodically queries the application server to ask for the current time. The response to this request includes the current local time and the number of seconds before the time should be queried again; this allows the application server to dynamically throttle time updates as opposed to having some preset value hardwired into the signs. It is important to note that we would prefer a time service of our own creation (as opposed to a NTP server) so we can ensure that the displays are perfectly synchronized with our schedule data.

The other subsystem is the *schedule controller*, which is responsible for requesting schedule data and updating the display. This subsystem periodically queries the application server to request schedule data for the stop. The stop number is passed[1] to the server so it returns only relevant data. The response includes all the information on the next passing times, including the route numbers and names, via/destination info, and the passing times. The application server can determine how much information should be returned for a stop (i.e. how far into the future to look, the number of passing times for a route, etc). Like the system clock query response, the schedule query response also includes the number of seconds before the schedule data should be queried again, which provides a great deal of flexibility, including the ability to dynamically throttle updates depending on such variables as the time of day, whether there are any schedule exceptions in effect, and even the stop number.

The schedule controller could be broken down into two subsystems, one which deals entirely with retrieving schedule data, and another which deals entirely with updating the display. Under this architecture, the schedule controller would signal the display controller to refresh the display whenever new data were retrieved, as would the system clock when the minute should be incremented.

Other subsystems can be considered to handle things like abstracting the communication process, monitoring and reporting status/error conditions, etc.

[1] This requires the stop number to be hardcoded in each controller, which increases its complexity. If however the stops were assigned static IP addresses then we could resolve the stop number by its IP address, completely decoupling the controller from its stop. This would only work if the IP address were truly static and could be determined when a stop was installed. The modem's MAC address could also be used if it could be accessed by the controller and sent to the application server with each query.

1.2 Initialization

When a sign is initialized or is recovering from a malfunction, the main process should immediately initialize the system clock process. Once it has signalled that it started successfully, the schedule subsystem process should then be activated.

1.3 Logging and Analysis

The application server can record every request received from the signs, which allows for detailed analysis and diagnostics. For example, the system could alert maintenance staff if a stop were to miss sending queries as scheduled.

1.4 Message Structure

Simple REST-style XML web services are an excellent candidate for the underlying encoding for a variety of reasons:

- They are ubiquitous and hence available to a large number of application environments
- They rely on simple, proven technology e.g. the HTTP protocol and web servers
- They are human-readable, making it easy to diagnose problems
- They offer a low-overhead web services solution when compared to SOAP web services

1.4.1 Time Request

The time request would look something like this, using stop 10064 as an example:

```
GET http://ebdd.winnipegtransit.org/time/10064
```

The response would be well-formed XML like the following:

```
<time local-time="2008-08-12T12:34:56" next-query="900" />
```

The controller would parse the date and time info in the `local-time` attribute and use that to reset the current time. The value in `next-query` would be used to determine the number of seconds from now when the time should next be queried.

1.4.2 Schedule Data Request

The schedule data request for stop 10064 would look something like this:

```
GET http://ebdd.winnipegtransit.org/schedule/10064
```

The response might be:

```
<schedule next-query="60">
  <route name="16 Selkirk ">
    <time departure="2008-09-12T12:47:00" destination="Burrows" via="">
    <time departure="2008-09-12T13:05:00" destination="McPhillips" via="">
    <time departure="2008-09-12T14:05:00" destination="Burrows" via="">
  </route>
</schedule>
```

The meaning of this response should be readily understandable. Depending on the stop and time of day, there could be multiple `<route>` elements returned. The application server would determine how many

`<time>` elements should be returned. As with the time request, the `next-query` value would be used to determine when the schedule data should next be polled.

The controller should not make any assumptions as to the ordering/sorting of the elements and attributes nor to the presence of elements and attributes. For examples, times within a route may not be in chronological order, routes may not be sorted by route name, and the `via` attribute may be omitted entirely if it is blank to minimize the amount of data transmitted.

Additional scheduled information could be returned as required.

1.4.3 Other Messages

Other messages type may be required. Examples could include “heartbeat” messages, error reporting messages, etc.