oBIX Watch: The Long Polling Approach

This short document describes the enhancement of the original oBIX Watch communication engine, which is implemented in C oBIX Tools project. The *long polling* feature allows oBIX clients receiving near-real-time updates from the server without overloading the network by frequent poll requests.

oBIX (**O**pen **B**uilding **I**nformation E**x**change) is the open standard describing XML-based interface and data model for building automation systems. The current version of the specification is available at OASIS web site[1]. Current paper assumes that the reader has a general view of oBIX concepts.

Classic polling

oBIX defines RESTful client-server relations between devices, which replicate the World Wide Web. Server keeps information about connected devices as a collection of interlinked objects, like web servers contain HTML pages. Each accessible object has unique URI like web pages do, and oBIX clients can access these objects in the same way as HTML browsers access web pages.

The described model works perfectly for sharing static or slow-changing data. But it is a common situation in Building Automation world when a client needs to have an up-to-date value of some variable object at the server. For instance, some user interface application which tracks people movement in the house, or an oBIX adapter for conditioning system which checks whether user has adjusted at the server the value of desired temperature. Assuming that only client can initiate a connection with a server (otherwise, the client should implement some server functionality in order to receive incoming requests), the only way to monitor some value at the server is to read periodically its value, i.e. to send poll requests. With shorter delays between poll requests a client receives faster data updates. On the other hand, frequent polling produces a lot of useless requests, which load the network channel and waste resources of both client and server. The problem becomes even bigger when a client needs to monitor several values simultaneously.

In order to reduce negative effect of the polling, oBIX introduces the Watch engine (OASIS oBIX TC, 2006, pp. 41-44). A client creates a Watch object at the server using standard service, adds URIs of the objects it wants to monitor, and then, instead of polling each object separately, the client polls created Watch object. Server checks all objects added to the Watch and returns the list of those, which have been updated since the last Watch poll request. Thus the Watch engine reduces network traffic in two ways: First, by reducing amount of poll requests, because each client can poll only one Watch instead of polling each object separately; and second, by reducing redundant server answers, because an empty answer is returned when subscribed objects are not changed. Figure 1 shows the difference in the process of polling data with and without using Watch objects.

[1] OASIS oBIX Technical Committee web page: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=obix

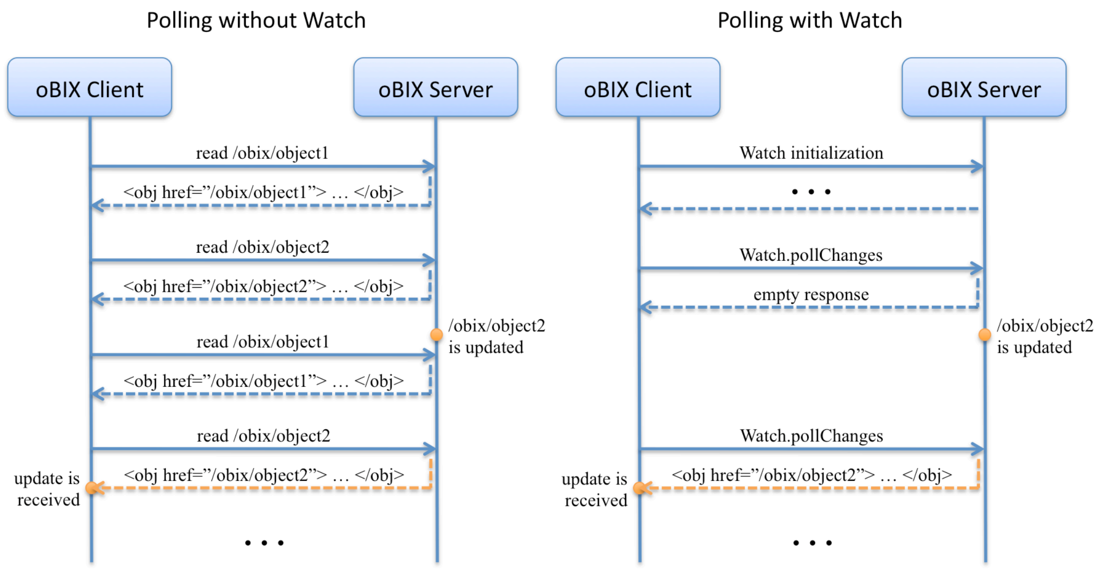

Figure 1 Polling data with and without oBIX Watch

Long polling

oBIX Watch engine improves the data propagation from server to client. Nevertheless, it cannot relieve completely from drawbacks of data polling. Client still receives data updates from the server with delay, which depends on how often the client is polling the Watch object. Developers of modern Ajax Web applications face the same drawbacks of classic polling approach. They have developed a set of techniques, referenced as Comet or Reverse Ajax, which allows a server to send data to a client on demand, without waiting for the client's poll request (Crane & McCarthy, 2008). One of those techniques, called *long polling*, can be easily adopted by oBIX Watch model.

With long polling a server doesn't answer to the client's poll immediately if there is no data to send. Instead, it blocks the request and waits for the data to be available. When new data comes, server uses the blocked poll request to send the data update. Otherwise, if no data appears during defined time period, server returns an empty answer. As soon as the client receives response from the server, it sends immediately a new poll request. Thus, the server always has a waiting poll request, which can be used to notify the client about some server event. Introducing this scheme to the oBIX Watch poll request, we achieve the communication model shown at the Figure 2.

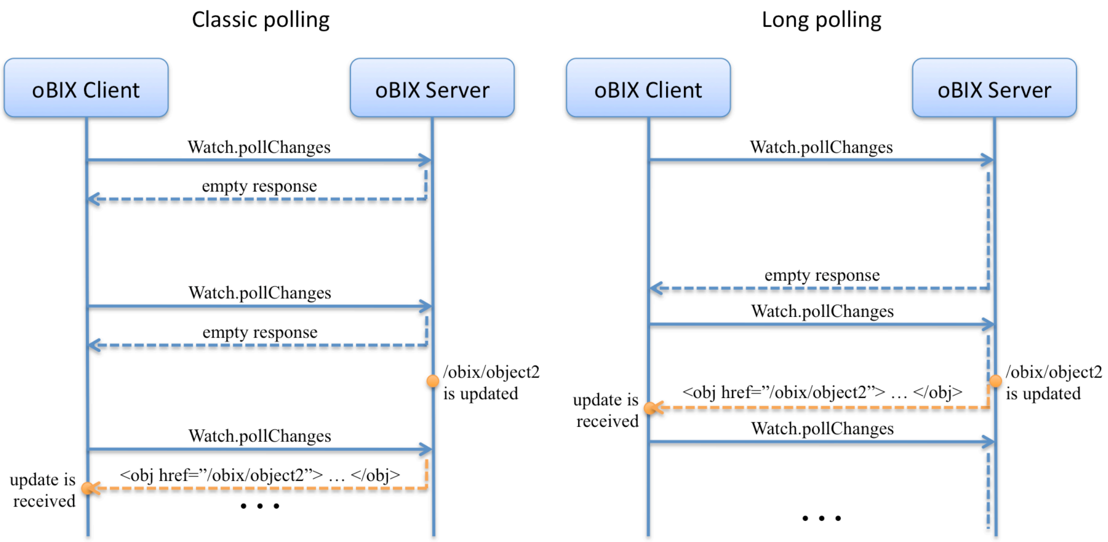

Figure 2 Long polling

Such solution has two advantages. First, long polling allows clients to receive updates of subscribed data immediately after the data is changed on the server. That is essential for systems with rapidly changing data and systems in which fast event handling is needed (e.g. security/fire alarms installations). The second advantage is that almost no changes should be made in oBIX client applications in order to work in this mode. From the client's point of view, a blocked poll request looks like a network delay. The only difference is that the client doesn't need to wait between sending poll requests.

On the other hand, making a server to hold requests increases server load. Usual web servers with synchronous request handling use one thread per each request. It means that waiting request blocks the thread by which it is handled. Thus 100 oBIX clients, which use long polling to get updates from the one server, will block 100 threads. Most of these threads will be in the waiting state, but every thread still uses considerable amount of resources, especially for servers running on embedded platforms. This problem is solved in Comet by asynchronous request processing, a technique allowing request blocking without blocking a thread, which is now supported by several web servers including Tomcat (Arcand, 2007; Arranz, 2008).

Actually, long polling is already used in Building Automation world. OPC XML-DA specification describes the similar technique for subscribing for data updates, called Advanced Polled Refresh Approach (OPC Foundation, 2004). This specification has the same ideas with oBIX: It describes Web Services interface for automation systems, based on previous OPC standards.

Implementation

C oBIX Tools project consist of two parts: the oBIX server and the client library, which simplifies development of oBIX client applications such as device adapters. Both parts now have long polling feature support.

At the server side the implementation is done with the idea to reduce the number of generated threads, because it is intended to run at embedded devices with limited resources. Initially the server is developed as FastCGI application. FastCGI is an interface for connecting external applications to web servers (Open Market, Inc., 1996). Traditional design of such applications assumes that a number of parallel processes are spawned and connected to the web server through FastCGI interface. Then server forwards to them incoming requests distributing the load between processes evenly. In case of C oBIX Server, it was decided to use *multiplexing* feature of FastCGI protocol, which allows one application to handle several requests simultaneously using one FastCGI channel. Inside the oBIX server all long poll requests are scheduled for delayed asynchronous handling using a dedicated thread (Figure 3). This architecture allows server to process normal requests while holding several long poll requests with help only one additional thread.

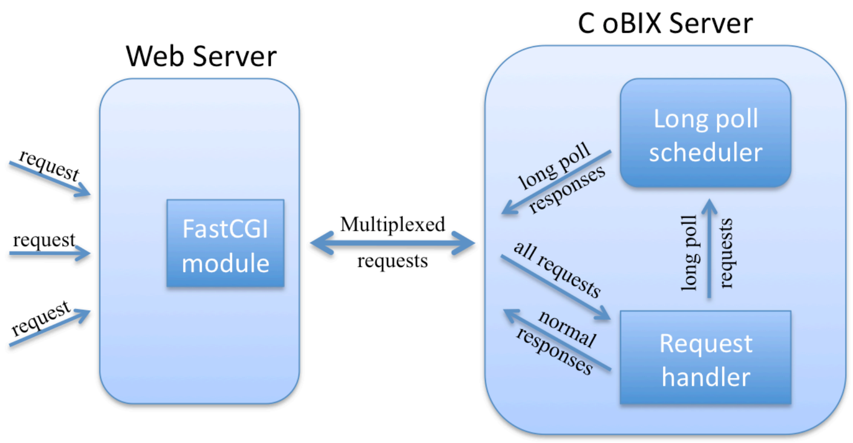

Figure 3 C oBIX Server architecture

Long polling is introduced to the oBIX interface in the following way. When a client creates a new Watch object by invoking *watchService.make*[2] operation at the oBIX server, the server returns an extended Watch object. This object implements the following contract:

```
<obj href="LongPollWatch" is="obix:Watch">
  <obj name="pollWaitInterval">
    <reltime name="min" min="PT0S" val="PT0S" writable="true" />
    <reltime name="max" min="PT0S" val="PT0S" writable="true" />
  </obj>
</obj>
```

So, the extended Watch contains one additional object named “pollWaitInterval”. This object defines the minimum and maximum waiting intervals for the long poll request. Maximum waiting time says for how long the server will hold the request if there is no data to send. Minimum interval specifies the time for which the server will wait even if a new data is available right after the request is received (Figure 4). Thus it guarantees the minimum interval for which a long poll request will be held. It is done for cases when a data at the server is updated so fast that it can overwhelm a network channel or a slow client.

[2] oBIX 1.0 Specification, page 41.

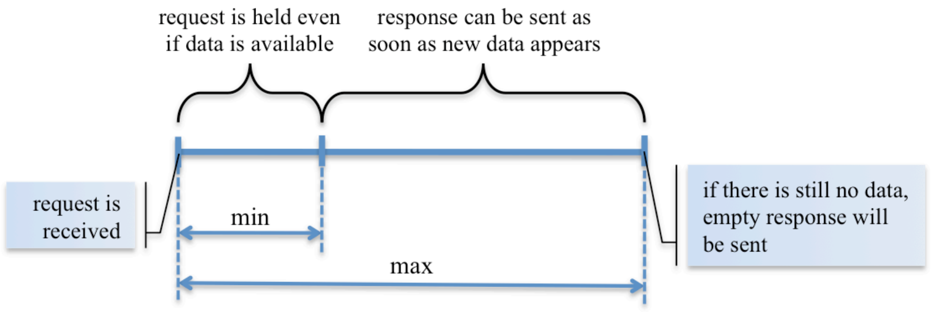
request is held even if data is available
response can be sent as soon as new data appears
request is received
min
max
if there is still no data, empty response will be sent

Figure 4 Long Poll wait interval

Changing these values will affect only one type of request: Invoking the *Watch.pollChanges* operation, which is used to poll updates from the server. Initially both intervals are set to zero and it means that server will answer immediately, i.e. will handle all poll requests as normal ones. Therefore, clients, which are not aware about long polling feature, will just ignore the additional values in the extended Watch object and proceed with traditional polling. Advanced clients should change waiting intervals during the initial Watch setup, and the next time when they will invoke Watch.pollChanges operation, server will hold the request (Figure 2).

The oBIX client library performs all operations with Watch objects including creation and polling. For a developer it provides a simple API, which allows subscribing listeners for the desired values at the oBIX server. The integrated long polling feature can be enabled during library initialization. Library configuration XML file should contain the following tags among other parameters of the oBIX connection:

```
<long-poll>
  <min-interval val="200" />
  <max-interval val="60000" />
</long-poll>
```

Minimum and maximum waiting intervals are provided in milliseconds. The library sets these values to the created Watch object at the server. In case when <long-poll> tag is omitted or the library detects that the server does not support long polling, the traditional polling is used.

Bibliography

- Arcand, J.-F. (2007, May 15). *New Adventures in Comet: polling, long polling or Http streaming with AJAX. Which one to choose?* From Java.net: http://weblogs.java.net/blog/jfarcand/archive/2007/05/new_adventures.html
- Arranz, J. M. (2008, November 21). *Is it Raining Comets and Threads... Again?* From Comet Daily: http://cometdaily.com/2008/11/21/are-raining-comets-and-threads/

- Crane, D., & McCarthy, P. (2008). *Comet and Reverse Ajax: The Next Generation Ajax 2.0.* Apress.
- OASIS oBIX TC. (2006, December 5). oBIX 1.0 Specification.
- OPC Foundation. (2004, December 18). OPC XML-DA Specification.
- Open Market, Inc. (1996, April). *FastCGI: A High-Performance Web Server Interface.* From FastCGI: http://www.fastcgi.com/drupal/node/6?q=node/15