Web Services for Smart Objects - Part 3: A real-world web service system for smart objects

Jean-Philippe Vasseur and Adam Dunkels - July 02, 2010

Part 1 offers an overview of web services and how they can enable smart objects to be efficiently integrated into existing IT and enterprise business systems. Part 2 examines the question "Can smart objects maintain a good performance for web services?"

9.3 PACHUBE: A WEB SERVICE SYSTEM FOR SMART OBJECTS

To ground the discussion about web services and RESTful interfaces for smart objects, we investigate Pachube, one particular real-world instantiation of a RESTful interface for smart objects.

Pachube is a web site, as shown in Figure 9.9, to which users can submit sensor data from sensor networks and upload and store the data on the Pachube server. The sensor data can later be retrieved and processed from the Pachube servers. The Pachube web site lists several ideas for which Pachube can be used such as electrical usage monitoring and management, real-time pollution monitoring, and home automation.
FIGURE 9.9 The Pachube web site. Data sources are marked on a world map.

The developers of Pachube envision it as the fabric on which smart object systems and applications can be built. Smart objects, sensor networks, and telemetry systems submit data to the Pachube servers where the data are stored for later retrieval. The data can be retrieved by stand-alone applications that process the data to either visualize the data for human users, or to autonomously operate on the data.

Examples of applications that may want to work on these data without a human in the loop are building automation systems or electricity savings systems. The input to such a system consists of temperature data and electricity readings from sensors that submit their data to Pachube. An excerpt of the Pachube web site that illustrates this concept is shown in Figure 9.10.

FIGURE 9.10 Pachube is intended to be an intermediary for smart object applications.
We use Pachube as an example because of its status as an emerging service provider in the ecosystem that is about to form around smart objects. Pachube provides an open Application Program Interface (API) based on the RESTful architectural model and allows remote sensors to send their data to the Pachube servers over an HTTP connection. Where many of the applications for smart objects target industrial applications, Pachube illustrates the possibility of a consumer-oriented service for smart objects.

Figure 9.11 shows how data stored on the Pachube server can be displayed directly in a web browser window, but this is only one of the many available alternatives for accessing the data. Since the data stored on the servers are opaque, the data can be retrieved and processed independent of the API.

![A feed of sensor data from the Pachube web site.](image)

**FIGURE 9.11** A feed of sensor data from the Pachube web site.

Pachube provides an open API for accessing the sensor data stored on the servers over the Internet. It is intentionally simple and provides methods for uploading and downloading data. Application complexity is held outside of the system. Applications only query the Pachube servers for data, and any processing is performed on the application side.

Pachube data are divided into feeds. Roughly speaking, one feed corresponds to one instance of a particular application. For example, a building automation system built on Pachube may use one feed for the sensor data and control from one building.

Feeds are further subdivided into data streams. A data stream can come from one particular sensor or one particular physical location such as a room. Applications that work on the data can choose to collect data from one stream or many streams from the same feed, depending on the application.

Many of the sensors that serve their data to the Pachube servers are connected to the Internet via an external device such as a PC. As the field of smart objects continues to grow, we are likely to see IP-based smart objects that communicate directly with the Pachube server.

**Interaction Model**

**9.3.1 Interaction Model**
The interaction model of the Pachube API is simple. Clients, either smart objects or sensors connected to an IP network through a PC, connect to the Pachube server using HTTP and send their data using the HTTP request. The server responds with a status code and an amount of data. The request may either provide new data to be stored on the Pachube servers or a request for data to be delivered from the same servers. Both types of requests are sent using HTTP.

When a client performs an HTTP request to a Pachube server, the client first sets up a TCP connection to the server. Once the TCP connection has been successfully opened, the client sends its request using the normal HTTP mechanism where the first line of data sent from the client contains the HTTP request verb, followed by additional lines of text that contain additional HTTP headers. If the request contains additional data, they follow after the HTTP headers.

The server responds in standard HTTP by sending the status code as the first data over the TCP connection. The status code is followed by the server's HTTP headers. If the request caused any data to be sent back from the server to the client, these data are sent after the HTTP headers.

When the HTTP interaction is complete, the TCP connection can either be closed directly or kept open in anticipation of another request at some later time. Whether the connection is closed or not is determined via a negotiation through the HTTP headers. If either the client or the server sends the connection close HTTP header, the connection is closed after the request has been completed. Otherwise the TCP connection is kept open in anticipation of another request between the client and the server.

9.3.2 Pachube Data Formats
Pachube supports several data formats for exchanging data between clients and servers. Providing different forms of exchanging data allows integration of different types of systems with the Pachube servers. A simple sensor that only wants to submit data to Pachube may choose to send its data in a simple format that requires low effort to construct and transport, whereas a visualization system that processes sensed data from the Pachube server to visualize it needs meta-information about the data such as where the data were sampled and when.

In the Pachube system, the clients decide how they want their data to be formatted as part of the requests they pose to the Pachube servers. The responsibility for converting the data between the formats falls on the Pachube servers rather than the client, as the clients may be resource-constrained smart objects.

Not all data formats contain the same amount of information. The simplest formats contain sensor data values, whereas the more complex formats contain metadata such as where the sensor data were obtained and at what time the data were sampled. Pachube supports the following formats:

- Extended Environments Markup Language (EEML): A custom version of XML tailored to contain sensor data. The EEML format contains tags that specify the spatial location at which the sensor data were sampled, as well as meta-information about the sensor data such as the minimum and maximum values that the sensor data can reach, and the
default unit in which the data are to be represented. An example of an EEML document is shown in Figure 9.12.

- **JSON:** The JSON format contains the same amount of information as the EEML representation, but formatted in JSON rather than EEML. The JSON format is more compact than the EEML format, and is also easier to parse for programs implemented in JavaScript.

- **ATOM and RSS:** The ATOM and RSS formats contain less information than the EEML and JSON formats. The ATOM and RSS formats contain sensor data, but include only a limited form of metadata such as the spatial location of the sensors as well as tags and titles of the sensors.

- **Comma-separated value format (CSV):** This is the most basic format. It also contains the least amount of meta-information: the data stream contains only the sensor data. The CSV format is suitable for use on tiny units with limited processing power where creation or parsing of the more complex formats is not suitable.

```xml
<eeml
  <environment>
    <location exposure="indoor" domain="physical" disposition="fixed">
      <name>My Room</name>
      <lat>32.4</lat>
      <lon>22.7</lon>
      <ele>0.2</ele>
    </location>
    <data id="0">
      <tag>temperature</tag>
      <value minValue="23.0" maxValue="48.0">36.2</value>
      <unit symbol="C" type="derivedSI">Celsius</unit>
    </data>
  </environment>
</eeml>
```

**FIGURE 9.12** A document in EEML format that contains sensor data.

**HTTP Requests**

**9.3.3 HTTP Requests**

All Pachube requests between clients and servers are performed by using HTTP requests. As previously discussed, the REST architecture uses HTTP request types for different types of method invocations and Pachube is no different. Pachube uses four different HTTP request types for its operations: GET, PUT, POST, and DELETE. The different requests are used on different occasions:
GET: This request method is used to retrieve sensor data from a Pachube server. With a GET request, the URI provided as part of the request contains both the identity of the data feed and the client’s data format. The URI contains information about the type of data the client wants to receive, the identity of the feed, and what data format the client wants. The identity of the data feed is given as the directory part of the HTTP URI, whereas the data format is provided as a file extension.

PUT: This method is used when submitting new sensor data to the Pachube server. Data to be submitted from the client to the server are provided in the data portion of the HTTP request, which follows the HTTP header. As with the GET request, the feed identity and the data format are included in the HTTP URI that is sent together with the HTTP request.

POST: This request method is used to create a feed and to create a new data stream within a previously established feed. The body of the HTTP request sent by the client contains the definition of the feed or the data stream. The definition is provided in EEML format. When establishing a stream, the server creates a stream into which the client may use the PUT method to insert data.

DELETE: This request method is used to delete a data feed or a data stream. The URI provided with the request contains the identifier of the feed or stream. Once a feed or stream has been deleted, it cannot be restored.

9.3.4 HTTP Return Codes
On every HTTP request the server responds to the client with a return code. The return code provides information about the request, such as if the request was successful or erroneous. If there was an error, the return code contains information about the cause of the error. HTTP return codes are represented as three-digit numbers. The basic HTTP return codes are specified in the base HTTP specification [83], but many HTTP servers have added their own codes.

The Pachube API uses HTTP return codes to inform the client about the state of the request. The return codes are sent to successful transactions as well as failed ones. For failed transactions, the return code provides insight into what caused the problem, and the HTTP body contains an XML document containing an error message.

The HTTP return codes used by the Pachube API are

- 200 OK: This code is returned when a request is completed successfully. Unlike the other return codes, this one does not
indicate an error.

- **401 Not Authorized**: This return code is sent in response to a client request that needed authentication, but where the authentication key was invalid or not present.
- **403 Forbidden**: This error code is returned when the Pachube servers did not execute the request. The reason the server did not respond to the request is given in the body of the HTTP reply.
- **404 Not Found**: The requested URI was not found. Either the feed it requested could not be found, or the method that the client invoked did not exist.
- **422 Unprocessable Entity**: This return code is sent in response to client requests that contain EEML data. The return code tells the client that the EEML contained semantic errors, even if it was syntactically correct.
- **500 Internal Server Error**: This return code is sent when there is an internal error with the Pachube servers.
- **503 No Server Error**: This return code is sent when there are no Pachube servers available to complete a request.

**Authentication and Security**

**9.3.5 Authentication and Security**

To determine who can access what data, the Pachube system provides a simple form of authentication. The purpose of the Pachube authentication is to identify the client to the server, so that the server knows if it should trust requests from the client. Clients that are authenticated can insert data into a stream, create new streams within a feed, and retrieve data from a stream. Clients that cannot be authenticated are denied access to the data by the Pachube server.

The Pachube authentication mechanism is simple. With each HTTP request performed by the client, the client provides a secret key. The server checks the secret key with the pre-registered key for the feed that the client is trying to access. If the key supplied by the client matches the key stored on the server, the server allows the client access to the data.

The key for a particular data feed is created when the feed is created. When creating the feed, the client needs to remember the key that was created as part of the feed since the key is needed for future access to the feed.

The key is sent as part of each HTTP request performed by the client. The key can either be sent as part of the URI or as part of the HTTP headers. Since the authentication key is transmitted in clear text in every HTTP request, it is trivial for third parties to sniff the key as it traverses the network. The sniffed key can later be used not only to gain access to the data, but to delete
the entire data feed, including the data history.

To make it harder for third parties to gain unauthorized access to the authorization key, Pachube provides a way to encrypt the data stream using transport layer security through the Secure Sockets Layer (SSL). With SSL, the entire HTTP transaction is protected by encryption so that third parties sniffing on the data cannot read the key or the data transaction.

9.3.6 Triggers

The synchronous API provided by Pachube works well for applications that periodically submit data to Pachube and periodically poll the servers for new data, but it does not allow fully reactive applications. Reactive applications react instantly to incoming data. One example of a reactive application is a burglar alarm that directly alerts the owner when the sensors detect a break-in.

To allow reactive applications, Pachube provides a mechanism called a trigger. A trigger is a small function that clients can upload to the server. Trigger functions are extremely simple and are only able to perform a threshold comparison on a data stream. If the data values in the data stream become greater than or less than the threshold provided in the trigger function, the trigger is executed.

When a trigger is executed, the Pachube server performs an HTTP GET request to a preprogrammed URI. The URI, which is provided by the client when configuring the trigger, points to an application hosted by the user on an external web server. The HTTP request sent by the Pachube server contains information about what feed and data stream caused the trigger to execute, as well as the current data value from the data stream. This permits reactive applications that do not need to poll the Pachube servers for data. After the trigger, the application may use the synchronous Pachube API to retrieve further information about the event that caused the trigger to execute.

Trigger functions can be represented either in XML or in JSON notation. An example trigger is shown in Figure 9.13.
FIGURE 9.13 Pachube trigger function that triggers when the sensor data are greater than 20.0, expressed in JSON format.

This trigger function is programmed to react when the data stream value exceeds 20.0. The "trigger_type" keyword is set to ">" (abbreviation for greater than). The stream ID is 0, the environment ID is 1233, and the user name is "Pachube".

The threshold value is given by the "threshold_value" parameter and is set to 20.0. The "url" field contains the URI that the Pachube server will call when the trigger is executed. The URI must correspond to the RESTful API of the reactive application. The "notified_at" field is updated with the date and time the trigger was last executed. Finally, the "id" parameter contains the identity number of this particular trigger function.

9.4 CONCLUSIONS
Web services provide an established mechanism for exchanging data between disparate systems. They are widely used in general purpose IT systems and the integration benefits of running web services on smart objects are large. With web services for smart objects, smart object systems can be readily integrated in general purpose IT systems such as enterprise resource planning systems and business systems.

Web services can be implemented using the REST principles, which are an architectural model for distributed systems. The REST principles can be efficiently run on top of an HTTP connection, making it simple and compelling for the resource-constrained smart object devices.

Although the performance of web services has been criticized in the context of large-scale server systems, recent studies show that web services can be efficiently implemented on smart objects. Web services can be run over low-power radio networks with good results.
Taken together, the interoperability and integration benefits of web services for smart objects, combined with their low resource requirements and good performance, make them a compelling choice for smart object systems.

References:

Printed with permission from Morgan Kaufmann, a division of Elsevier. Copyright 2010. "Interconnecting Smart Objects with IP: The Next Internet" by Jean-Philippe Vasseur and Adam Dunkels. For more information about this title and other similar books, please visit www.elsevierdirect.com.

Related links:
Web services for smart objects - Part 1: Overview | Part 2: Performance considerations
Rise of the Embedded Internet
TCP/IP Embedded Internet Applications - The Network Layer,
Back to the basics: Improve TCP/IP performance in memory-constrained embedded apps
The basics of adding 802.11n wireless connectivity to your embedded design