8-1-2014

Using Smart Water Meters In (Near) Real-Time On The iWIDGET System

Michael G. Barry

Mark E. Purcell

Bradley J. Eck

USING SMART WATER METERS IN (NEAR) REAL-TIME ON THE IWIDGET SYSTEM

MARK E. PURCELL, MICHAEL G. BARRY, BRADLEY J. ECK
IBM Research Ireland, Mulhuddart, Dublin 15, Ireland.

Devices and technologies to measure and report water consumption at sub-daily intervals are growing in popularity. Data from these devices are creating new opportunities to manage the supply and demand of water in near real-time. To this end, the EU FP7 iWIDGET (**I**mproved **W**ater efficiency through **I**CT for integrated supply-**D**emand side mana**GE**men**T**) project is developing a state-of-the art analytics platform for the integrated management of urban water. Key challenges include extracting useful insights from high-resolution consumption data and exploring a range of decision-support tools for water utilities and consumers. To overcome these challenges, iWIDGET is developing a distributed, open, robust, collaborative architecture that allows partners and utilities to collect and process data from a large number of sensors in parallel and analyze data on demand.

We present a distributed system that enables flexible, near real-time monitoring of water networks by providing four critical mechanisms. First, a means to regularly poll water utility raw data systems. Second, assimilation of fresh data into a purposely designed, high-performance database. Third, geographically local or remote analytic systems poll the database to incorporate the latest consumption information in their analysis. Lastly, an online portal based platform is used to trigger analysis and review results.

A key architectural feature of this system is a loose coupling between central storage and analytic systems. Communication between the central storage and processing components utilizes standard techniques, including WaterML, over RESTful web services. This arrangement avoids restrictions on the underlying technologies in analytical components and allows analytic systems to execute on different operating systems and run-times. The system is under active development and will enable a wide variety of tools for water utilities and individual consumers.

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 318272.

INTRODUCTION

Measurements of water consumption at sub-daily intervals create new opportunities for urban water management. Many types of analysis including time-series forecasting, optimization, and agent based modeling can benefit from water consumption data. The relevant techniques are implemented in a wide range of analysis tools and new tools and techniques are under active development. The most valuable insights will likely come from a collaborative analysis approach, whereby the results of one technique become part of the input data set for another technique. In this way, successive algorithms may iteratively learn more from the underlying data, creating new and unique insights.

Realizing the potential of a collaborative and near real-time analytics platform is not straightforward. The underlying technologies vary considerably and depend not only on the analysis task, but also the preferences of different organizations. Some techniques employ licensed tools such as Matlab, others are written in Java, Python or C/C++. For example, a high performance analysis routine written in C++ cannot easily interact with an R analytic, especially if they operate on separate local area networks. This problem is compounded by varying run-time environments and hardware requirements. The variety of approaches and tool kits complicates the potential for interoperability.

This paper presents the architecture of a system for collaborative and near-real time analysis of water consumption data. The goals of the system include flexibility for research and commercial use as well as ease of interoperability between disparate systems and tools. Achieving these goals will result in improved water efficiency for utilities and consumers.

ARCHITECTURE

The motivation behind the iWIDGET architecture is to promote interoperability between heterogeneous technologies, specifically proprietary analysis engines, storage and visualization. As such, all participating analysis systems are loosely coupled: there is no direct link between systems. An iWIDGET application programming interface (API) provides access to water consumption data, in effect sharing this data across participating remote systems. An analysis system can request data from this API, perform its analysis, and return a result through the API. These results can be disseminated across the other participating systems and be used in subsequent analysis.

Figure 1 shows the iWIDGET system, which is comprised of a centralized server and database, with remote client systems for data acquisition and analysis. Clients in the iWIDGET system interact with the central iWIDGET server through an abstraction layer: the iWIDGET API.

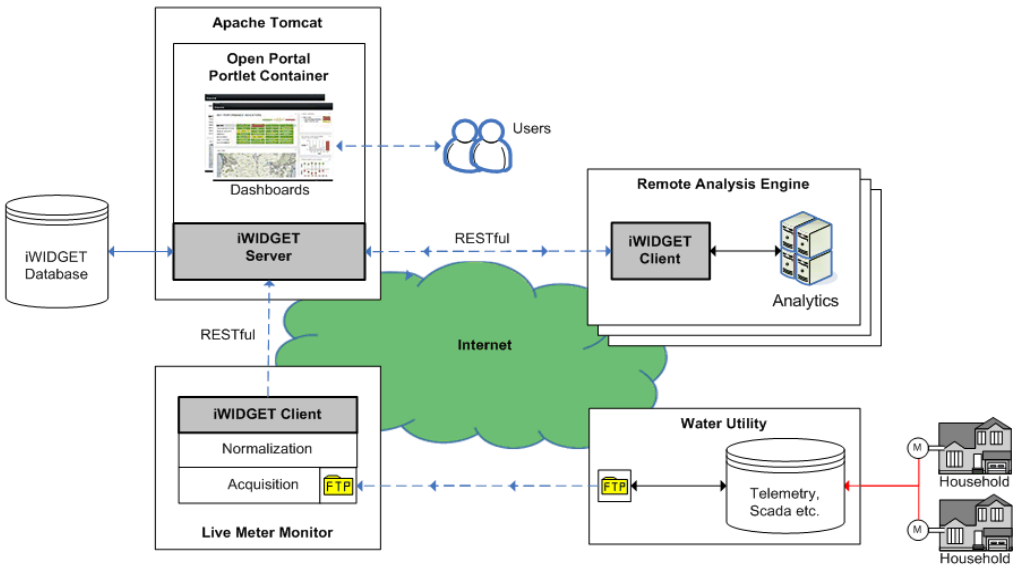

Figure 1. iWIDGET Distributed Architecture

A water utility acquires readings from individual consumption points and makes the readings available by file transfer protocol (FTP). An iWIDGET monitoring client polls the utility FTP server on a frequent schedule, normalizes the readings, and populates the database through the iWIDGET API and server. Remote iWIDGET analysis clients obtain these readings from the iWIDGET server, through the iWIDGET API, perform an analysis, and return results.

All requests for data are made through the iWIDGET API, not directly to the database, thereby insulating the requestor from the specific details of the database implementation. This abstraction effectively virtualizes the database, allowing different database technologies to be employed. For example in a commercial scenario, IBM® DB2® enterprise edition may be installed, whereas for smaller projects, open source or license free solutions, such as MySQL can be used. It is even possible to use a file system directly. For iWIDGET clients and servers, the API provides the abstraction layer which enables the use of many different strategies and underlying techniques.

In order to provide configuration details for remote analysis engines, as well as view analysis results, an online portal platform is used. Two flavors of platform are included in the architecture. Firstly, at a water utility level, the data requested by analysis engines is of water network scale. With this large volume of data, analysis may take some time and needs to be

scheduled to execute on an appropriate, remote system. This is the standard iWIDGET loosely-coupled model. Secondly, for smaller amounts of data, for example at a household level, analysis can be performed quite quickly. In this case, analysis and display can be more tightly-coupled, with analysis running whilst the display is constructed. This is an on-demand variation of the platform, where no remote analysis engine is required.

INTERACTIONS

In the iWIDGET system, components for storage, analysis and visualization interact through the iWIDGET API, which is built as a RESTful service. Representational state transfer (REST) [1], defines a set of architectural principles by which web services are designed to focus on a system's resources. REST describes how resource states are addressed and transferred over the hyper-text transfer protocol (HTTP) to a wide range of clients written in different programming languages. A RESTful service performs tasks by answering requests over HTTP. Requests and responses are representations of resources capturing their current state and a resource is any piece of information/data that can be globally identified with an HTTP uniform resource identifier (URI), for example, *http://MyServer/iWidget/MyService*.

From Figure 1 above, the underlying database technology is masked by the iWIDGET API RESTful abstraction layer. Analytic components only require knowledge of this API and its inputs and outputs. This arrangement leaves the designers of analytic components free to use whatever underlying technology they wish.

One of the main reasons for proposing REST as the technique for accessing remote services, is its ease of client-side implementation. Alternative mechanisms such as the simple object access protocol (SOAP) [2] or remote procedure calls (RPC) [3], require every client application (analysis engine) to implement a functional end-point to the communications channel. With REST however, most of the complexity of the communication resides on the server. To operate on the iWIDGET platform, the client simply accesses a URI, minimizing the integration effort on the part of existing analysis engines.

This client-side ease of implementation strategy only necessitates access to the iWIDGET server using HTTP verbs. The verbs are: PUT to create a server side entry; GET to retrieve an existing entry; POST to update an entry; and DELETE to remove a server side entry. All interactions between requests are stateless, reflecting current server-side state. It is even possible to construct an iWIDGET client without writing any code. For example, the *cURL* [4] application can be used:

curl -X GET http://iwidgetserver/iWidget/devices

This RESTful query returns a device listing, or list of water meters, that can be used as input to an analysis engine. Other queries retrieve meter readings for a given meter over a specific time period:

curl -X GET http://iwidgetserver/iWidget/timeseries/00062620
?startTime=2009-03-01 03:00:00
&endTime=2009-03-01 04:00:00

The iWIDGET API, based on RESTful services, is similar to the Open Geospatial Consortium's KVP Protocol (key-value pair) [5]. iWIDGET extends the HTTP GET semantics of KVP with the PUT, POST and DELETE features described above. Although still under active development, the iWIDGET API should eventually converge with the OGC protocols, perhaps proposing extensions to the standard.

DATA FORMAT

Water consumption data transported through the iWIDGET API, must adhere to a known structure. The structure of this data must be comprehensible by all participating analysis engines. Two schemes are currently used to format this data, JSON and WaterML.

An example measurement time series is shown in Figure 2, for both WaterML and JSON. This is the data that is returned to an iWIDGET client in response to a query (see the *curl* examples earlier). WaterML is significantly more verbose than JSON, and this has implications in terms of network bandwidth and processing requirements, especially when multiple remote systems are operating on the iWIDGET platform.

JavaScript Object Notation (JSON) [6], is a lightweight, human-readable, data-interchange format. It is programming language agnostic, and is built on a collection of name-value pairs, and ordered lists of values, or arrays. JSON is very easy to parse, and can be used directly in many programming languages, including Python and Javascript. This makes it particularly suitable as a data structure for online web portals.

WaterML [7], is an information model for water data, specified by the Open Geospatial Consortium. WaterML is based on XML and was designed to promote interoperability for hydrologic time series data across many adopters. As an XML format, the data must first be parsed by a SAX [8] based XML parser (such as Xerces), before consumption and analysis by a client application. This places an additional programming effort on clients that wish to use the WaterML variant of iWIDGET data.

```
<wml2:Collection xsi:schemaLocation="http://opengis.net/waterML/2.0
      http://schemas.opengis.net/waterml/2.0/waterml2.xsd">
  <wml2:observationMember>
    <om:OM_Observation>
      <om:featureOfInterest>
        <wml2:MonitoringPoint>
          <gml:identifier>00060986</gml:identifier>
        </wml2:MonitoringPoint>
      </om:featureOfInterest>
      <om:result>
        <wml2:MeasurementTimeseries>
          <wml2:metadata>
            <wml2:MeasurementTimeseriesMetadata>
              <wml2:cumulative>false</wml2:cumulative>
            </wml2:MeasurementTimeseriesMetadata>
          </wml2:metadata
          <wml2:defaultPointMetadata>
            <wml2:DefaultTVPMeasurementMetadata>
              <wml2:quality xlink:href="http://www.opengis.net/def/WaterML/2.0/quality/good"
                            xlink:title="Good"/>
              <wml2:uom code="l/min"/>
              <wml2:interpolationType xlink:href="http://www.opengis.net/def/waterml/2.0/Interpolation"
                                      xlink:title="Average In Preceeding Interval"/>
              <wml2:aggregationDuration>P1D</wml2:aggregationDuration>
            </wml2:DefaultTVPMeasurementMetadata>
          </wml2:defaultPointMetadata>
          <wml2:point>
            <wml2:MeasurementTVP>
              <wml2:time>2009-03-11 04:00:00.0</wml2:time>
              <wml2:value>0.0</wml2:value>
            </wml2:MeasurementTVP>
          </wml2:point>
          <wml2:point>
            <wml2:MeasurementTVP>
              <wml2:time>2009-03-11 05:00:00.0</wml2:time>
              <wml2:value>1.6666667E-5</wml2:value>
            </wml2:MeasurementTVP>
          </wml2:point>
        </wml2:MeasurementTimeseries>
      </om:result>
    </om:OM_Observation>
  </wml2:observationMember>
</wml2:Collection>
```

```
[{
"deviceID":"00060986",
"units":"l/min,
"timeSeries":[
{“date":"2009-03-11 04:00:00",
"value":0.0},
{"date":"2009-03-11 05:00:00",
"value":1.6666667E-5}]
}]
```

Figure 2. iWIDGET WaterML and JSON encodings for a measurement time series

DEPLOYMENT

The iWIDGET architecture supports a wide range of deployment scenarios. The current development deployment provides user interfaces through a portal server. This is composed of a number of sub-components called portlets. Each portlet provides an interaction layer to specific analytics, establishing a configuration and presenting results. For example, a portlet could define a configuration for a remote analysis engine to execute at midnight, using as input, all water consumption data for the previous twenty-four hours.

Although the portal based user interface is depicted on the same physical system as the iWIDGET server, in Figure 1, as with analysis engines, individual portlets are in fact acting as a remote clients. They use the same techniques as remote analysis engines to access water

consumption data: the iWIDGET API. This enables the user interface and iWIDGET server to be deployed on geographically separate systems. As the portal based user interface platform is effectively an iWIDGET client, it is similarly possible to provide a tablet/smartphone based user interface platform, also acting as an iWIDGET client.

The database can also operate remote to the iWIDGET server making the iWIDGET platform ready for cloud deployments. No system adaptation is required for a fully virtual environment. Multi-tenant use is also supported, whereby several water utilities can operate simultaneously on the iWIDGET platform, with appropriate data isolation and protection.

To facilitate the use of the iWIDGET system in research environments, the architecture is realizable using fully open source, or license free software. The active development platform was built in this manner. Recent meter data is acquired in near real-time, via FTP, from a remote water utility and passed through one or more normalization routines, which produce a standardized data set for use in analysis calculations. For example, volumetric meter readings are normalized into a series of hourly flow rates, denominated in m^3/s.

This data is then stored in a purposely designed iWIDGET database which is backed by IBM® DB2® Express-C. This is the community edition of the database, with no licensing restrictions. The database schema is built with standard SQL constructs allowing for transition to other database technologies. In fact, as part of the development process, this schema was also successfully deployed on an IBM® Informix database.

The iWIDGET server, which hosts the iWIDGET API, is built upon Apache Tomcat, an open source product. The user interface is portlet based and requires a portal server, for example, the open source Open Portal Portlet Container.

Future commercial installations of the iWIDGET system can replace all of these free products with enterprise scale alternatives, such as IBM® DB2® Enterprise Server Edition and IBM® WebSphere® Portal Server.

CONCLUSIONS

The iWIDGET project is primarily concerned with improving water efficiency through the analysis of water usage from sub-daily measurements of water consumption. The iWIDGET architecture, system design, storage mechanism, and communication techniques support this objective by providing data that is normalized in a consistent way. The design minimizes complexity for analytics, ensuring easy integration of newly designed analytic routines. Existing analytical tools may operate on the iWIDGET platform by instantiating an iWIDGET client, which interacts with the system.

An iWIDGET system holds benefits for many stakeholders. Universities, companies, and utilities wishing to analyze high resolution water consumption data can access a common set of data and share results. Individual water users gain visibility into their own resource usage. Utilities can provide an iWIDGET interface to their data to gain access to a variety of analysis

tools. The results of these analytics on water consumption data can lead to significant advancements across the whole network, in the form of improved leak detection, optimized flow rates and reduced energy consumption.

REFERENCES

[1] Roy T. Fielding and Richard N. Taylor., "Principled design of the modern Web architecture", *ACM Transactions on Internet Technology 2*, May 2002, 115–150.

[2] W3C, "SOAP Version 1.2 Part 0: Primer", Second Edition, *in Nilo Mitra & Yves Lafon*, W3C Recommendation, 2007.

[3] Andrew D. Birrell , Bruce Jay Nelson, "Implementing remote procedure calls", ACM Transactions on Computer Systems (TOCS), v.2 n.1, p.39-59, February 1984.

[4] Jes Fraser, "Command-Line Application Roundup", Linux Journal, Issue 198, October 2010.

[5] Peter Baumann, "Web Coverage Service 2.0 Interface Standard - KVP Protocol Binding Extension", Open GeoSpatial Consortium, Ref: 09-147**,** 2013.

[6] Tim Bray, "The JavaScript Object Notation (JSON) Data Interchange Format", Internet Engineering Task Force RFC7159, ISSN: 2070-1721, 2014.

[7] Peter Taylor, "WaterML 2.0: Part 1 – Timeseries", Open GeoSpatial Consortium, Ref: 10-126r4, 2014.

[8] David Brownell, *Sax2,* O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.