Thing Broker: A Twitter for Things

Ricardo A. P. Almeida
Computer Department.
Federal University of São Carlos, SP, Brazil
ricardoalmeida@dc.ufscar.br

Michael Blackstock
Media and Graphics Interdisciplinary Center.
The University of British Columbia, BC, Canada.
mblackst@magic.ubc.ca

Rodger Lea
Media and Graphics Interdisciplinary Center.
The University of British Columbia, BC, Canada.
rodgerl@ece.ubc.ca

Roberto Calderon
Media and Graphics Interdisciplinary Center.
The University of British Columbia, BC, Canada.
roberto@alumni.ubc.ca

Antonio F. Prado
Computer Department.
Federal University of São Carlos, SP, Brazil
prado@dc.ufscar.br

Hélio Crestana Guardia
Computer Department.
Federal University of São Carlos, SP, Brazil
helio@dc.ufscar.br

UbiComp'13 Adjunct, September 8–12, 2013, Zurich, Switzerland.

Abstract
In the Web of Things, standard web technologies and protocols are used to represent and communicate with physical and virtual things. One challenge toward this vision is integrating things with different characteristics, protocols, interfaces and constraints while maintaining the simplicity and flexibility required for a variety of applications. In this paper we present the Thing Broker, a core platform for Web of Things that provides RESTFul interfaces to things using a Twitter-based set of abstractions and communication model. We present the key abstractions, a reference implementation and explain how a typical WoT application can be created using Thing Broker. We finish with a preliminary evaluation and draw some lessons from our experiences.

Author Keywords
Web of Things, Internet of Things, Pervasive Computing.

ACM Classification Keywords
D.2.11. Data abstraction
D.2.11. Domain-specific architectures
D.2.11. Patterns

Introduction

The "Internet of Things" refers to the technology and implementation of a global network of uniquely identifiable objects (things) and their online representations [6].

The Web of Things (WoT) leverages the Web to support this vision and relies on standard web protocols to name, access, find and use Things, in particular the use of HTTP, RESTful interfaces and asynchronous mechanisms based on web standards for communication [7]. In this way, the WoT expands the notion of things to not only include smart objects with embedded computing capabilities like sensors, controllers, and home appliances, but any uniquely identifiable person, place or thing. This includes users with mobile phones, active sensors, controllers, and passive every day objects and environments.

One of the challenges toward achieving the WoT is that it can be difficult to quickly build applications capable of integrating things with different characteristics and constraints. Part of the reason is the wide variety and scope of possible WoT applications, such as:

- Large screen to mobile phone interaction – requires event-brokering facilities to move interaction events between things.
- Map visualizations of the current value of sensors distributed geographically – requires supporting geographic information and other metadata about sensors, maintaining the latest value of sensors, and a capability for on-demand queries and data aggregation.
- Personal sustainability applications such as smart meter monitoring – requires historical data storage of the value of smart meters and electricity use in the home.
- Emergency response applications – requires event processing to determine whether an emergency condition has occurred, and the ability to send alerts.
- Participative sensing or crowd sourcing applications – Citizen-powered data collection requires associating mobile phones, their capabilities and users. Security and privacy issues must be considered avoiding unauthorized access and preventing from exposing user data to applications with no need.
- Context aware applications - Location based or profile based apps rely on infrastructure to infer higher-level context (information about a thing). Based on the context, applications can determine relationships between things, format data according to user preferences, and complement data about things after processing context information.

A key characteristic of these scenarios, and one of the main issues with the WoT is the variety of things that there are in the physical and virtual world that may be needed by applications. Some things are already web enabled (e.g. earthquake sensors on the web, the GPS location of ferries), while others need gateways and other software components to adapt or proxy things to the web. In short, the sources of data from "things" and the way of controlling "things" vary depending on their capabilities and the gateway chosen to expose these things. Even when things are web-enabled, there are different ways of representing them using standards like HTML, XML, JSON, RSS, Atom, KML or OWL. Fundamentally, there isn't a widely used standard to represent things, their content and their relationships.

In this paper we describe the Think Broker platform, which we have developed and which supports a core approach and a uniform set of abstractions to easily develop WoT applications in a variety of domains. Two of the key contributions of the Thing Broker are:

- A common *thing* abstraction: the Thing Broker exposes a notion of things, their relationships and the events and content that they generate.
- A Twitter like model, using notions of "following" and "followed" links, for representing thing relationships.

Background

The Thing Broker has evolved from our research on ubiquitous computing and IoT platforms, most recently the Magic Broker 2 [4] and WoTKit [5]. The Magic Broker models things using *channels* that can be organized in relatively static hierarchical relationships using its namespace. Thing Broker thing-relationships are dynamic as things follow and unfollow other things.

Like other web of things platforms such as Xively [12], Open Sens.se [8], Thing Speak [11] and the WoTKit, the Thing Broker can be used as a data hub, these systems often focus on providing facilities for processing and visualizing data streams; The Thing Broker aims on providing an underlying platform for application development, not a cloud based IoT service with a UI for managing and visualizing sensors and their data.

Like the publish/subscribe communication model used Spacebrew [3], and ubiquitous computing systems such as the iROS EventHeap [9] the Thing Broker allows clients to publish and subscribe to data associated with things, however, in the Thing Broker there is no distinction between publishers and any "thing" represented in the system can be a publisher or a subscriber at the same time. In all of the aforementioned systems there is limited or no clear notion of a thing, i.e. the developer is left to manage thing entities. Equally, although several of the systems have a channel or pub/sub notion, they lack support for relationships as expressed by following/followed and leave the application to support these thing relationships.

Thing Broker

Based on the experience with these platforms and their applications, we found that it is useful to consider the world as being composed of "things" that produce time-related events and are related to each other. For example, when building cross-display applications, hardware sensors (e.g. mobile phone accelerometers), screens (e.g. a large screen, and a small screen) and the content itself (e.g. an image or text) can all be modelled as related 'things'. Using this concept of things and their relationships, we've found that developers can easily conceptualize, design and create applications by assembling together key elements of an application.

Key Abstractions

In the Thing Broker, the main abstractions provided are "things" and "events". A "thing" holds information about any type of resource and its relationships with other "things". Relationships are represented using a model similar to what is observed in Twitter, as a following/followed link. "Events" are associated with things and encapsulate any data they produce, from

simple text information to multimedia content. Each new event generated by a thing will cause its associated data to be made available to the followers of this thing. Figure 2 illustrates the representation of a thing in the Thing Broker.

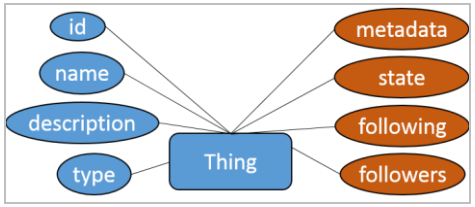

Figure 2. Thing abstraction.

As seen in Figure 2, a thing has a globally unique id, which can be a human-readable name or a universally unique identifier (UUID). A name, a simple description and a type can also be defined to better describe and identify a thing at the application level. Thing-attributes and thing-relationships are represented using complementary data stored in the metadata, state, and the following/followers fields.

The metadata field is used to hold attributes and their corresponding values associated with a thing, basically a set of typed name/value attributes. The type of a metadata attribute can be a number, a string or a multimedia resource. For example, a "smartphone" thing can have a "wifi-mac-address" attribute and a "screen-resolution" attributes for use by an application.

The state field consists of a set of dynamic attributes of a thing that can change their value with time, based on new events that occur and that are related to a thing. The state of a "thing" holds dynamic data which can be generated by processing contextual information such as the current physical location of a thing, its current IP address, the battery level, the established transmission rate, etc. This information can be used by clients of the Thing Broker to process a thing's context. Like the metadata field, it is also a set of typed name/value attributes. Neither the metadata nor the state fields have a set schema and can contain any attributes an Thing Broker client needs.

The *following* and *followers* fields represent relationships between things. For instance, a phone follows its sensors and a user may also follow a phone. The former case shows a dependency and the later a simple association. These fields consist of a set of thing ids and provide access points to the data produced by or associated to the corresponding thing. The following and the follower fields are used to retrieve information from other things using, respectively, a pull and a push communication models.

Any data created by a thing is represented in the Thing Broker as an "event". Events can represent actions, control messages or contents that are further processed. The event abstraction provides a generic structure that can represent from simple data such as strings to complex data structures or multimedia content. Based on the data of an event, applications running on the top of the Thing Broker are able to determine if they need to consume or forward content, change the state of things or trigger specific routines. In the Thing Broker, events are modelled as shown in Figure 3.

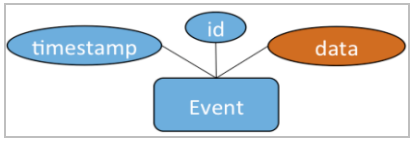

Figure 3. Event abstraction.

An event is composed of a universally unique identifier (UUID), a timestamp field of its arrival in the platform and a "data" field, which is used to store information associated with the event. The "data" field is a set of typed name/value attributes structured as a document using a specific syntax, such as the XML or JSON syntax. Data field attributes can also contain references (URIs) to binary content stored in a local repository or in a remote server, providing support to multimedia content.

Any event created by a thing can be stored in the Thing Broker and accessed by other things when the appropriate security constraints defined to the platform are met. A thing can only have access to other thing's events if they share a following/followed relationship.

By focusing on the entities that communicate to produce and consume contents, the abstractions proposed by the Thing Broker provide a simple way of modelling WoT applications. In this sense, applications can be defined by their things and associated events.

A Twitter Based Model

Typical WoT applications will manipulate information related to several things/objects often requiring access to information from those things in order to perform some processing which establishes a dependency relationship. Other kinds of relationships can also exist, such as an association between two things to produce some content that will be accessed by a third thing.

Considering this common characteristic of WoT applications, Thing Broker provides a simple relationship model to integrate things and provide access to data created by them. This approach is based on the Twitter communication model, which uses the notions of following/followed to connect people with similar interests.

In the Twitter model, if a person wants to keep tracking the tweets from another person, she starts *following* that person. At the same time, the person that will be *followed* has a new follower added to her account. Once the relationship is established, the follower receives tweets posted by those she follows ordered by time. A follower may process the information received and, as a result, post comments or share content.

The same concept can be used to create relationships between things in applications. If thing-1 wants to have access to data from thing-2, a following/followed relationship is established.

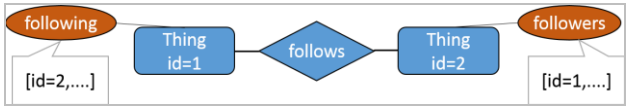

Figure 4. The following/followed relationship.

In Figure 4, thing-1 follows thing-2. This relationship is defined by adding the *id* of thing-2 in the *following* field of thing-1 and the *id* of thing-1 in the *followers* field of thing-2. Once the relationship is established, every new event produced by thing-2 causes thing-1 to be notified and allow it to access the information associated to the event.

The following/followed relationship also provides a hierarchical namespace to address things at different levels. For example, the user profile "john" may be associated to a "smartphone" which exchanges data with a location sensor in a room. The user profile manager (also a thing) in that case will need to access sensor data and specific information about the

smartphone, such as its network status. For this scenario, two following/followed relationships can be established: one between the thing "john" and the "smartphone" and another between the thing "smartphone" and the "location sensor". The second relationship creates a transitive relation between the user profile and the location sensor, so the "user profile" can access data from the smartphone *and* the sensor.

Security and Privacy

Security and privacy mechanisms are optional in the Thing Broker as clients of the platform may not need or want them for simplicity or performance reasons. In simpler deployments, the Thing Broker is seen as just a backend service for inter-thing communications. In larger deployments where security is critical, Thing Broker may also provide authenticated, access restricted, and encrypted message delivery.

Secure transmission protocols are optionally supported by the Thing Broker. HTTPs and HTTP basic authentication can be easily integrated to the set of Thing Broker services, providing a first level of security and privacy.

Furthermore, considering the following/followed relationship supported by the platform, it would be possible to establish a group access control mechanism, which provides access to information of a set of related things at the same time. For example, if the thing "home" follows a "living room" thing and the latter follows the "light sensor" and "presence sensor" things, providing access to data from the "home" thing also provides access to data from its related things.

This basic level of security balances the need for a generic mechanism against the overhead imposed on things with no need of software security. However, Thing Broker is designed to make it possible to easily add security layers on the top of its architecture in order to provide the desired security and privacy levels needed for WoT applications.

Supporting Complex Queries

Thing Broker supports basic and advanced queries to retrieve data about things and their associated events. Basic queries usually require general parameters such as timestamp or maximum number of records, resulting in a set of unfiltered data. Advanced queries may be created by specifying attributes of a thing or event, to filter query results. The architecture supports an SQL-like query language to find appropriate sources of data and perform simple aggregation.

Architecture and Implementation

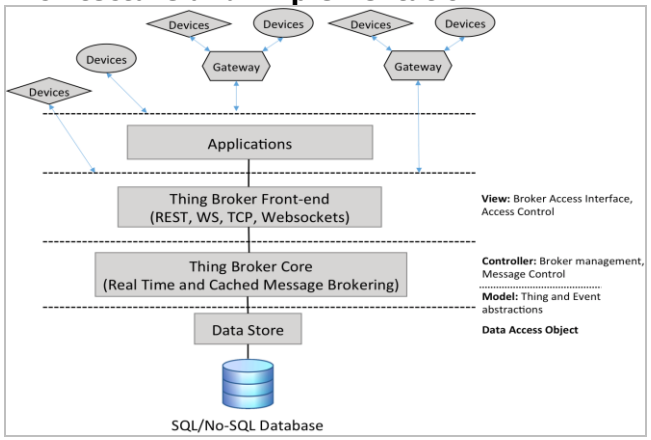

Figure 1. Thing Broker architecture.

The Thing Broker architecture was designed to provide a clear separation by layers as illustrated in Figure 1. The architecture is composed of three layers: DAO

(Data Access Object), core and front-end layers, structured in a MVC (Model-View-Controller) architectural model.

- The DAO layer provides interfaces for data manipulation and storage using a SQL or No/SQL database.
- The core layer provides abstractions of things and events, which are the main structures used to manipulate information about things and their content. This layer also provides real-time and cached message brokering, with a set of basic interfaces to control the message flow between things and to control changes in the state of things.
- The front-end supports different protocols and communication models, such as RESTful and traditional Web services, and can also communicate via explicit message exchange on top of TCP sockets or websockets. Using a variety of protocols, the front-end layer is designed to work as a third-party application module, running locally and providing thing and event data manipulation APIs to the applications.

Above the front-end layer, different applications can be built allowing the developer to focus on the application logic, leaving data manipulation to the Thing Broker. Applications can provide interfaces to different kinds of resources, ranging from simple sensors to more complex services. The Thing Broker can also be directly accessed by resources using the protocols supported by the platform.

A reference Java implementation of Thing Broker architecture was created for validation purposes. The Java Spring Framework [10] was used to create RESTful interfaces to the core functionalities of Thing Broker, such as thing registration, event posting and refined queries when searching for specific events.

The implementation also provides a cross-domain solution to allow web applications to access Thing Broker through JavaScript calls.

Things and events are structured using standardized JavaScript Object Notation (JSON) documents and their data (thing description and events) are stored in a MongoDB [2] instance, a NoSQL database. Event persistence is optional, aimed at applications that require historical data. All data transmitted through the Thing Broker Restful services are also structured as JSON documents.

Real-time event communication is provided using a message broker system based on Java Message Service API [1] and a HTTP long polling approach. Historical events data can be accessed by providing URL parameters that are used in specific queries to retrieve a set of events.

The Thing Broker source code[1] and its API documentation are available on GitHub. A native Android client API[2] that encapsulates the communication with Thing Broker is also publicly available.

[1] Thing Broker source code link: https://github.com/ubc-magic/thingbroker.git

[2] Thing Broker Android client API link: https://github.com/ricardoalmeida123/thingbroker-android-api.git

Validation

A set of applications has been developed with the Thing Broker implementation to validate the proposed abstractions and API. These applications consist in a collaborative picture gallery, a digital message board and a meeting alert system. The applications include code developed for the Android platform, using a Thing Broker Android client API, and JavaScript code running on standard web browsers. All of the applications use HTTP long polling for real-time event communication, between things.

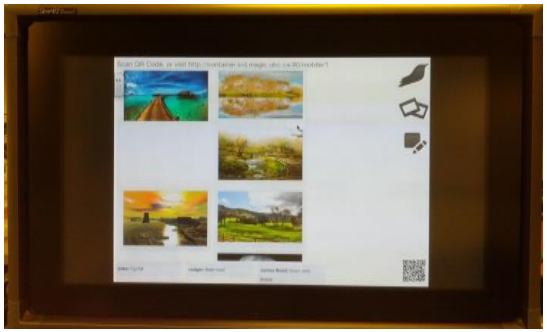

Figure 5. Collaborative Picture Gallery.

Figure 6. Digital Message Board.

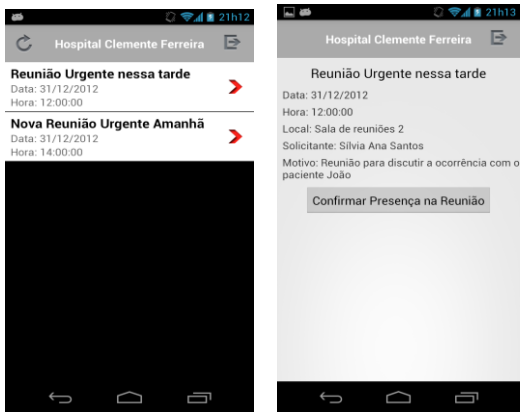

Figure 7. Meeting notifier for a Healthcare Scenario.

The collaborative picture gallery (Figure 5) is a web application developed for use in public spaces. It allows users to post pictures to be displayed on a display, sharing content they consider interesting to other people. Photos can be uploaded using either a mobile application or a web browser. This application involves two types of things: "photo-gallery-service" and a "display". The "photo-gallery-service" consists in a distributed service that runs in a set of mobile devices and browsers allowing users to post photos to Thing Broker, acting as the same "thing" in the platform. The thing "photo-gallery-service" is followed by a set of displays represented as things "display-1", "display-2", "display-n". Every time a new photo is posted by "photo-gallery-service", all the displays simultaneously receives the content and presents it on the collaborative picture gallery.

The digital message board, shown in Figure 6, is another example of collaborative application developed for public places. In this application users are modelled as individual things followed by the thing "message-

board". When a user posts a new event containing a message, the thing "message-board" receives the content and presents it on the screen as a post-it.

The meeting notifier application shown in Figure 7 was developed to allow the staff in a real healthcare facility to notify/invite people about upcoming meetings. After receiving a notification, a user can confirm if she will attend the meeting. A web interface is provided to schedule the meetings. In this application, the Thing Broker is used as follows:

- The meeting scheduler service is represented in Thing Broker as a thing called "meeting-scheduler".
- Each staff member is also represented as a thing that follows the "meeting-scheduler".
- When a new meeting is scheduled using the developed web interface, a new event is created and posted to the Thing Broker. Events are stored in the Thing Broker that containing details about the meeting, such as date, hour, local and subject. An array of thing ids is also provided as part of an event to model the invited staff members.
- The Android client application uses long polling mechanism to periodically check for newly scheduled meetings. Long polling requests are performed in background to send the staff member id as a query parameter to the Thing Broker. This id is used by the Thing Broker to retrieve the data about all new events that represent meetings to which that staff member was invited.
- Once a new event is retrieved to the mobile application, a notification shown on the Android notification bar informs that the user was invited to a new meeting.
- On notification, the user can open the mobile application to check details about the meeting. This details view also contains a confirm button that allows the user to confirm that she will attend the meeting. Once the confirmation button is pressed, the event containing data about the meeting is updated with the confirmation and posted back to Thing Broker, which will merge it with the event already registered.
- Using the web interface again, a user can check all the staff members that confirmed their presence at the meeting.

As the examples illustrate, the thing model can be applied to different entities, from services to physical objects.

Conclusions and Future Work

The Thing Broker abstractions and communications model provides a more uniform access interface to different Web of Things entities. Using a single abstraction to represent things, each with its own set of configurable attributes, Thing Broker allows all sorts of objects, from physical sensors to high-level services to be addressed in a WoT application. The following/follows relationship model provides an abstraction for publish/subscribe style asynchronous communication between things. Push and pull based transmissions may be used for receiving event notifications.

Based on application development and experiments to date, the Thing Broker has proven to be a simple, and flexible, yet powerful platform for functional integration. These are keys aspects in a world of heterogeneous entities that need to be accessed and possibly

controlled by a varying number of distributed processing units.

For our future work we plan to further investigate the authentication and encryption needed to accommodate the communication between things of different complexities and in particular try to support things that have their own (possibly physical) security mechanisms in larger scale WoT applications. In addition, we intend to improve support for the mobility of things as well as the production and consumption of continuous data flows.

Acknowledgements

Special thanks to professors Junia Coutinho Anacleto and Sidney Fels, for their advice, support and direction. This work was partially supported by the CAPES/DFAIT Brazil-Canada collaborative research program with additional support from AeroInfo Systems Inc.

References

[1] About Java Message Service. http://docs.oracle.com/javaee/6/tutorial/doc/bncdr.html

[2] About MongoDB http://www.mongodb.org/about/introduction/.

[3] About Spacebrew. http://docs.spacebrew.cc/about/.

[4] Blackstock, M., Kaviani, N., Lea, R. and Friday, A. MAGIC Broker 2: An Open and Extensible Platform for the Internet of Things. Internet of Things 2010 International Conference (IoT 2010), 1–8.

[5] Blackstock, M. and Lea, R. IoT Mashups with the WoTKit. 3rd International Conference on The Internet of Things (IOT 2012), 159 –166.

[6] Gubbi, J., Buyya, R., Slaven, M. and Marimuthu, P. Internet of Things (IoT): A vision, architectural elements, and future directions. http://dx.doi.org/10.1016/j.future.2013.01.010..

[7] Guinard, D. A Web of Things Application Architecture. PhD Thesis. ETH Zurich, Zurich, Switzerland, 2011. http://www.webofthings.org/dom/thesis.pdf.

[8] Open Sen.se Feel, Act, Make sense, Feel, Act, Make sense. http://open.sen.se/

[9] Ponnekanti, S.R., Johanson, B., Kiciman, E. and Fox, A. "Portability, Extensibility and Robustness in iROS," IEEE PerCom, 2003, p. 11.

[10] Spring Framework. http://www.springsource.org/spring-framework.

[11] The Internet of Things – ThingSpeak. https://thingspeak.com/.

[12] Xively - The Internet of Things is Open for Business" https://www.xively.com