MAGIC Broker 2

An Open and Extensible Platform for the Internet of Things

Michael Blackstock[1], Nima Kaviani[1], Rodger Lea[1], Adrian Friday[2]

[1] Media and Graphics Interdisciplinary Centre,
University of British Columbia
Vancouver, Canada
{mblackst@magic,nkaviani@cs,rlea@ece}.ubc.ca

[2] Computing Department,
Lancaster University
Lancaster, UK
adrian@comp.lancs.ac.uk

***Abstract*—One of the challenges of creating applications from confederations of Internet-enabled things is the complexity of having to deal with spontaneously interacting and partially available heterogeneous devices. In this paper we describe the features of the MAGIC Broker 2 (MB2) a platform designed to offer a simple and consistent programming interface for collections of things. We report on the key abstractions offered by the platform and report on its use for developing two IoT applications involving spontaneous device interaction: 1) mobile phones and public displays, and 2) a web-based sensor actuator network portal called Sense Tecnic (STS). We discuss how the MB2 abstractions and implementation have evolved over time to the current design. Finally we present a preliminary performance evaluation and report qualitatively on the developers' experience of using our platform.**

Keywords-Internet of Things; Modules and Interfaces; Software architectures; Pervasive computing;

I. INTRODUCTION

The concept of the Internet of Things (IoT) builds upon Weiser's vision of ubiquitous computing whereby physical objects have a representation in the on-line world. These on-line representations can identify themselves and interact with each other; they can also contain current and historical information about the object derived from sensors, or change the physical environment using associated actuators and controls. New applications engendered by the IoT include: supply chain management using RFIDs and sensor tags [1], facility management [2], spontaneous thing-to-thing interaction such as mobile phone to public displays [3-8] and emerging applications that use detailed real time information such as shared sensing [9] participative sensing for sustainability [10], social applications and search engines for things [11].

In such environments applications are comprised from confederations of spontaneously interacting devices that may be only available periodically due to mobility, communication failures and duty cycling (e.g. for power saving). Our aim is to provide a platform that offers a consistent model and interface for building IoT applications from cooperating things.

Our platform, called Magic Broker 2 (MB2), is novel in two important ways: firstly, it offers a simple, uniform web-based API for building IoT applications. Secondly, the platform offers developers three built-in programming abstractions: publish-subscribe event channels, persistent content and state storage, and brokerage of services via remote-procedure call. These programming abstractions are organised into a flexible and lightweight user-defined hierarchical namespace.

We have used this system to create a range of IoT applications involving spontaneous device interaction such as between mobile phones and public displays, and opportunistic, or shared sensing and control of devices using a web-based sensor actuator network called *Sense Tecnic* (STS). MB2 builds on our previous work in deploying ubicomp applications and supporting middleware called the REST Broker [5]. It has been re-implemented to meet new requirements uncovered during application development. These experiences have highlighted many of the challenges in crafting supporting systems for the IoT.

In this paper we describe the features of the MB2 platform and report on its use for developing our IoT applications. We explain how the abstractions and implementation have evolved over time to the current design. Finally we present a preliminary performance evaluation and report qualitatively on the developers' experience of using our platform, whose feedback suggests the abstractions are useful for building such applications and are easy to use and learn.

II. FLEXIBILITY AND SIMPLICITY: KEY TO SUPPORTING THE IOT

Our vision for the IoT encompasses a broad set of applications ranging from real world control using low-level sensors and actuators through to public display infrastructure that facilitates spontaneous interaction between groups of devices. In each application MB2 supports opportunistic interaction by providing a "well known" rendezvous point to broker local communications and bridge to the Internet.

Central to our vision is the notion of spontaneous interactions between the real world and virtual devices and the associated software that supports such interaction. To motivate our work, we have developed two IoT applications that have allowed us to define our core requirements, programming model and associated software abstractions. The first of these is a sensor application framework that supports rapid development of sensor/actuator 'mashups' designed to allow developers to quickly identify real-world objects and group and control them using high level tools and visualizations. Our second exemplar targets a different class of IoT applications and focuses on spontaneous device-to-device interaction enabled using a situated sensor point built around a public screen model.

Our experiences with these diverse IoT application classes have driven the development of a set of abstractions for building IoT applications, but perhaps more importantly have exposed the need for *flexibility* and generic support for *adaptability* and diversity of devices, protocols, applications and user models. We address these fundamental needs in two key ways.

- Flexibility: We utilize a publish/subscribe eventing system which support the flexibility in configuring and building assemblies of devices and services needed for emerging IoT applications. Our approach is built around a low-level event-based broker allowing us to decouple devices, data and users to support dynamic and robust assemblies of devices and services.
- Simplicity: Based on experience with generic platforms for UbiComp [14][15][16] and similar platforms for home networking [17], and in contrast to other IoT middlewares ([13],[12]), MB2 does not use techniques such as semantic modelling or generic device models [18]; we adopt a lightweight approach allowing the application developer and end-user to apply their own semantics to the collection of things that make up an application.

In this paper we provide evidence that MB2's basic programming model, built around channels containing events and state, is sufficiently expressive to meet the application needs described in this paper but does not burden the developer with complexity. OSGi [19] is used at the core of our new implementation to support run-time flexibility and adaptability of protocols and core system services as in [20].

A. Programming Models and Protocols

The MB2 programming model and protocol has evolved from our experiences with ubicomp middleware and more recently with support for a range of IoT applications such as mobile phone display interaction [3-6], and wide area sensor/actuator networks. The Sense Tecnic system (STS) is a sensor/actuator network portal that allows users to contribute information about sensors or actuators that send data from a sensor to the portal. Other users can find the sensor data they are interested in to create new applications and dashboards using the portal's processing and visualization features. Experience with these real-world IoT applications has allowed us to refine our abstractions and programming model to meet our twin core requirements for flexibility and simplicity. The MAGIC Broker 2 offers four core abstractions that may be directed or contained within a *channel*. These offer the key abstractions for constructing IoT: *events* and *state*, as mentioned earlier; augmented with two additional abstractions: support for synchronous remote procedure call (RPC)-style semantics called *services* and *content* storage (c.f. file system services).

Events. The backbone of the MB2 system is a publish-subscribe event broker that decouples event sources from sinks. Event-based systems offer two advantages: First, events naturally model the nature of many forms of interaction and changes in the real world. Interactive systems all involve the processing of events that are often user triggered and require timely handling - not well suited to underlying polling based schemes. Indeed, the use of events has been shown to be valuable in the construction of traditional single-user GUI applications, windowing systems, and multi-user synchronous groupware [21]. Similarly, we are often only interested in sensor data when it changes, in some cases trigged directly by user activity, the movement of RFID-tagged things between locations [11] or by changing conditions in the environment. Second, events provide a high level of flexibility: using events the control flow of communication between devices and applications can be changed dynamically; at development time the use of events allows components to be repurposed or extended, e.g. to accept input from more than one user at any time; and at debugging time, where communication between things is can be made visible on the event channel and component interactions can be simulated manually and/or using a test harness.

In the MB2 system, subscribing devices receive events sent to a channel from other devices. For example, mobile phones send events to designated channels to interact with public displays. In the Sense Tecnic system, sensors such as smart power meters send events to the channels assigned to them. Real time visualizations for any user subscribe to events on these channels.

State. MB2 supports a *state* abstraction that allows clients to request the last *n* events, as well as read and write name/value pairs in a channel. For example, in a public display channel we decided to store the last 16 messages in channel state. In our sustainability applications, channel state holds users' previous energy monitoring data. State can also be used to store contextual information such as the current location of a thing. Importantly, any application can interrogate this state without the need to wait for events containing updates to it.

Services. MB2 can also broker synchronous two-way request-response interactions called *services* with devices registered with the platform (analogous to a CORBA ORB). MB2 services are similar to those supported by SOAP web services and Java RMI. We have found, based on our own experience with event and tuple space based middleware [15],[22], the need to support RPC. Our analysis of other ubicomp systems [14] has also shown that support for synchronous calls to RPC-style services have often been used or retrofitted on non-RPC platforms partly, we suspect, as it remains a comfortable paradigm for developers. Even event-based systems such as iROS and one.world [23] support some notion of RPC-style services. Supporting synchronous, two-way interaction has obvious benefits for the developer; however, a disadvantage is that these interactions can limit the responsiveness of applications if they need to wait for responses [24].

Example uses of services in our applications include requesting directions from Google, performing a Flickr photo search, and sending MMS photo messages. Like events and state, the service abstraction is conceptually contained in a channel. External services interact with the MB2 in a similar

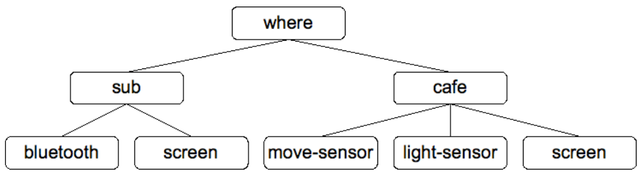

Figure 1. Example Channel Hierarchy

manner to event subscribers, except that they must also provide a response with a correlation identifier to the service caller.

Content. Finally, MB2 supports storage and retrieval of *content* such as images, videos, text, and HTML documents within a channel in a consistent way. This allows application developers to store content naturally associated with the people and things corresponding those channels. This can include non-interactive content such as images and video clips, or text, but also interactive content such as Flash applications or HTML pages containing Javascript. In our public display deployments, the channel-content facilities were used to store images and Flash-based interactive applications.

B. *Namespace: The Channel*

A *channel* is used as our namespace and conceptual container for other MB2 abstractions. It is used to name the on-line presence of things, and groups of things that comprise IoT applications. In interactive large public display applications, mobile phones spontaneously interact with groups of screens in various locations. We use the notion of a named channel to allow developers to name individual screens and groups of displays, mobile phones, and the interactive applications supported by these screens. In the Sense Tecnic sensor/actuator portal, channels correspond to sensors and actuators contributed by users to the system. In some cases we found it useful to name channels such that they represent a location-containment hierarchy, as illustrated by Figure 1. In this diagram, the channels are organized according to location in a *where* hierarchy (the 'sub' represents the Student Union Building). When using the middleware protocol, parent-child relationships in the namespace are indicated using a period "." notation. In the diagram the parent of channel *where.sub.screen* is *where.sub.*

Although we have found certain channel naming conventions useful, the organization of channels into locations or other groupings is arbitrary and up to the application developer, and in some cases, the end users of the application. In this manner, we rely on a "folksonomy" of names for things, rather than defining an ontology or comprehensive set of device profiles. By providing a system that allows flexible naming, our aim is to avoid imposing ways of classifying things to enable developers and end users to collaborate on device classifications for greater interoperability as the IoT evolves over time.

C. *Protocols*

To ensure support for diversity MB2 supports several protocols. Our primary protocol is HTTP/XML based and suitable for heterogeneous clients, cross-domain interaction and web-oriented application design. However we also support TCP/IP, SOAP, SMS and higher-level protocols such as Twitter. Using OSGi, protocols and other features can be added or removed from the system at run time.

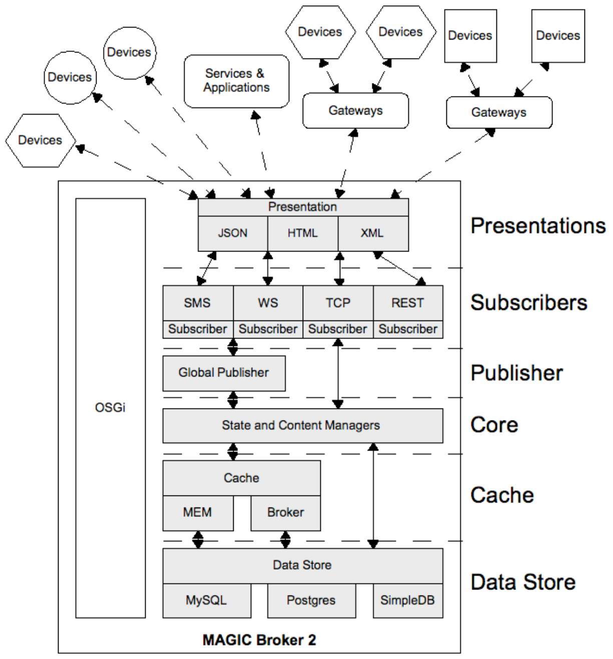

Figure 2. MAGIC Broker 2 Architecture.

III. DESIGN AND IMPLEMENTATION

MB2 is a pub/sub middleware coded with simplicity in mind. It implements the model discussed above as well as diverse communication protocols to facilitate developing IoT applications such as interactive public displays and web-based sensor actuator network applications. In MB2 each communication modality is implemented as an OSGi service registered with OSGi's service registry, discoverable by dependent services, and in turn capable of discovering required services. In our original work [5] we focused mainly on using a RESTful interface over HTTP. Although powerful, the diversity of the IoT has required us to extend our communications protocol support. MB2 supports a variety of protocols (e.g., HTTP, TCP, Web Services, and SMS) to connect clients to the core without the need for an intermediary. Each protocol implementation's lifecycle is completely independent of others during runtime, allowing them to be started, stopped and replaced without affecting core services or communications using other protocols.

MB2 consists of 86 classes and almost 5000 LOC. It provides an abstraction layer on top of OSGi by implementing eight OSGi bundles. Referring to Figure 2, which illustrates the MB2 architecture, the set of bundles and services implemented for the broker fall under one of the following categories from top to bottom: *Presentations, Subscribers, Publisher, Cache,* and *DataStore*; from which only the core is strictly required; others are optional.

The *Core* modules contain all the APIs and interfaces for the broker, from the generic data structures and containers to service interfaces for subscribers, publishers, clients, data persistence, caching, etc. It also provides some abstract

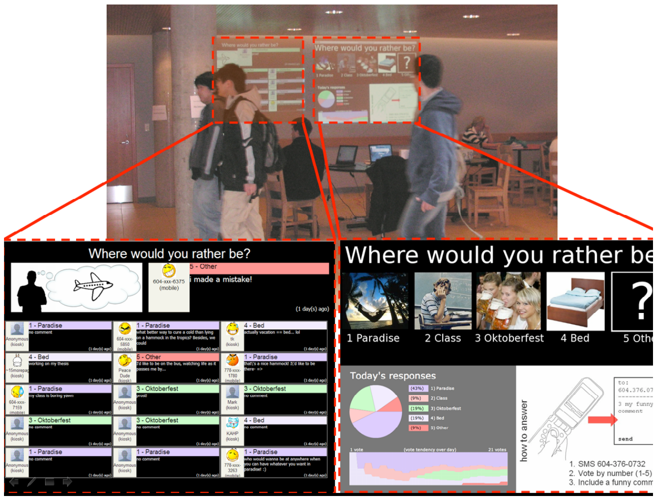

Figure 3. MAGIC Board and Voting App solicit responses from the public on thought provoking issues.

implementations for the key components of the broker (e.g., publisher, subscriber, etc.) to ease development of these components. Our *Global Publisher* bundle provides a single hook for the publishers to post to all subscribers regardless of the communication protocols. The *Cache* provides an in-memory key-value store for small but frequently used chunks of data on exchanged events, subscribed clients, content, and state information. The *DataStore* bundles implement the data persistence service interface for the broker, connecting the broker to a backend database to persist state, events, registered clients and existing channels. We have provided implementations of the DataStore bundles for MySQL and Amazon SimpleDB's NoSQL cloud database.

The *Subscriber* bundles allow for implementing variants of the communication modalities by extending the subscriber interface and corresponding abstract subscriber classes supplied by the *Core* bundles. Each subscriber bundle offers a distinct communication protocol and implements a wrapper to utilize the Global Publisher. Requests to publish events cause the Publisher to relay the events to all of the existing subscribers. The subscriber employs the client's preferred presentation service (XML, JSON, or HTML) to decode the message and pass it to the target client through their established communication protocol (e.g., HTTP, TCP, Web Services, or SMS). *Subscriber* bundles are flexible in that we can install and run one or more as needed providing the fine-grained level of flexibility and extensibility we desire in our supporting system to deal with heterogeneity of devices, communication protocols, use cases, users, etc. Requests for *state* and *content* are passed to the DataStore and cached locally to speedup future requests.

The real strengths of the new design of MB2 are three-fold:

- IoT device types are often resource or communication constrained. By supporting a range of communication protocols we can choose the best protocols to use for communicating with a range of devices, from small resource-constrained sensor devices to fully resourced server nodes and gateways. Furthermore, our model enables resource-constrained devices of various types to seamlessly connect to the broker and offload their processing load to the broker.
- The inherent characteristics of the MB2 abstractions and their implementation using OSGi allow for scalability and proper lifecycle management of new and old modules. Consequently, the system can be kept backward compatible while providing new features and functionalities with minimal to no disturbance to devices operating under the current system settings.
- Reliability: the ability to hot-swap modules improves the middleware reliability supporting fail-over and upgrade to improve system reliability.

IV. Application Development

We have developed a number of prototype applications using evolutions of the MAGIC Broker 2 platform. Some were deployed as part of the Lancaster e-Campus deployment [22] and others as part of the UBC Ubicomp test bed [25]. In this paper we present two case studies of IoT applications involving spontaneous device interaction between screens, sensors and mobile phones and Internet enabled embedded sensing.

A. *Interactive Public Display Applications*

A typical interactive public display deployment incorporates mobile phones, situated sensors and computers hosting public displays connected directly to the MB2. Mobile phones communicate with MB2 using SMS, HTTP or TCP depending on the capabilities of the phone. The software hosted on displays are typically browser-based and communicate directly with MB2 using HTTP. In some of our deployments, a voice XML gateway was used to post events to MB2 allowing mobile phone users to interact using voice commands. Situated sensors such as RFID readers, Bluetooth scanners and movement sensors may also post events to MB2-subscribed public displays.

MAGIC Board [3] illustrated in Figure 3 is a polling/voting application deployed to solicit responses to ‘thought-provoking or semi-contentious issues’. Members of the public contribute their opinions by commenting or voting either privately using SMS-enabled cell phones, or publicly at the display itself using a kiosk. We deployed MAGIC Board in a busy campus location to gain experience with integrating a diverse set of things (kiosks, cell phones, and displays) and protocols (HTTP, WiFi, and SMS) and to study how people can be encouraged to use public displays. MB2 was also used not only to relay messages between devices, but also to store state such as the current vote question and content including the images related to the questions.

Other screen-device interaction applications we have developed include an implementation of the popular Desktop Tower Defence game[1], an SMS based message board, a music video player suitable for a shared public space and a photo search and display application[2].

1 http://www.handdrawngames.com/DesktopTD/Game.asp

2 http://www.magic.ubc.ca/PSPI/pmwiki/pmwiki.php

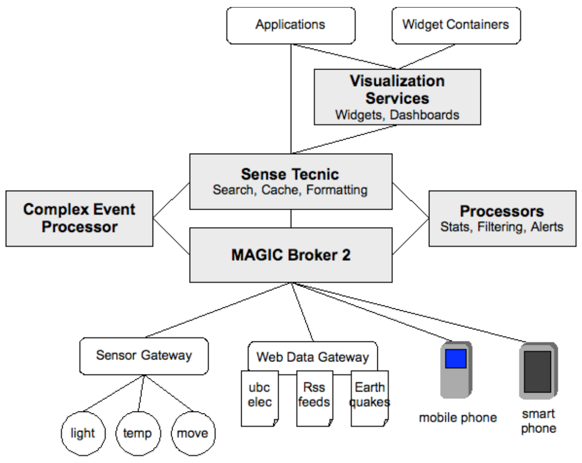

Figure 4. Sense Tecnic Architecture

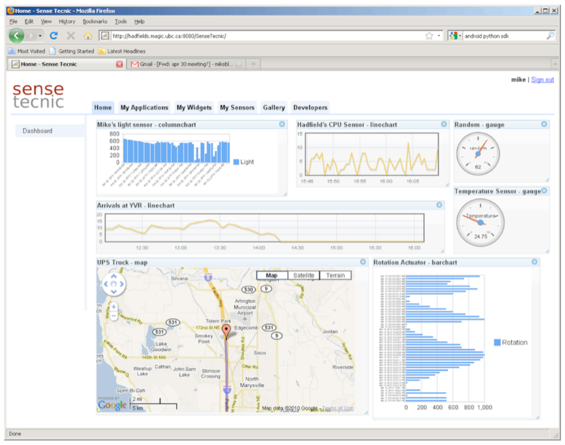
sense tecnic
Home
My Applications
My Widgets
My Sensors
Gallery
Developers
Dashboard
Mike's light sensor - columnchart
Hadfield's CPU Sensor - linechart
Random - gauge
Temperature Sensor - gauge
Arrivals at YVR - linechart
UPS Truck - map
Rotation Actuator - barchart
Map
Satellite
Terrain

Figure 5. Example Sense Tecnic dashboard

B. *Sensor Mashups*

The Sense Tecnic System (STS) is a web-based platform as a service for wide area sensor and actuator networks. The service makes streams of information on the Internet such as weather, air quality, location of ferries and buses, and physical devices, sensors and actuators such as mobile phones, cars, RFID readers, light and temperature sensors, all available to applications and end users in a uniform manner. Using Sense Tecnic, developers can quickly and easily publish new streams of data; consumers of data can easily find the sensor data they need, and deploy applications that coordinate, observe and control a variety of computing resources in the physical environment.

The distributed architecture of STS is illustrated in Figure 4. The Sense Tecnic service maps the sensors it manages to MB2 channels. In this way it uses MB2 to store information about a sensor (or actuator) and to broker sensor data feeds from sensors to applications and visualizations managed by Sense Tecnic. Publishers of sensor data register a sensor with the system using a web form. Sensor either send data to MB2 directly or using a sensor gateway as shown. Once a sensor is registered with the system and sending data, an STS user can monitor the sensor feed using one of the visualizations supported as illustrated in Figure 5.

Consumers of sensor information can then search for sensors of interest in the STS sensor gallery. Once they find a sensor, they can subscribe to it, adding it to their personal sensor palette called the *My Sensors* page. By clicking on a sensor in My Sensors, they can then monitor the sensor using a number of pluggable visualizations such as gauges, line charts, and maps. When they find a visualization they like, they can create a sensor widget that can be installed on an STS dashboard (Figure 5) or another widget container such as iGoogle.

When an application requests events with real time updates to populate visualization, the STS platform subscribes to the corresponding MB2 channel. To improve performance and reduce load on the MB2, STS system keeps a cache of events in a ring buffer in case more than one end user visualization or application requests the same data stream. The STS visualization services reformat data for presentation to create dashboards, widgets or custom applications as shown in Figure 5.

The STS platform also includes facilities to process sensor data, effectively creating higher-level sensors. Platform users create *pipes* similar in concept to Yahoo Pipes[3]. This configures a complex event-processing engine [26] to process lower-level sensor events sent back into MB2 for output to higher-level derived sensor feeds that can be used by applications and visuals.

STS is also capable of controlling actuators or requesting sensor configuration information. An STS user can create a “control visualization” that uses an MB2 output channel to change the state of an actuator, such as turn on lights in a building, or change the position of a servomotor. We expect to migrate this functionality to use the channel services abstraction in a future implementation.

C. *Current work: Sensors and Sustainability.*

The STS system and MB2 will become the core of a 'green tech' application focused on household sustainability. Using MB2 and our sensor infrastructure we are working to enable households and end users to monitor their activities e.g. home energy use, transportation choices, consumption choices related to recycling, etc. via a network of physical sensors, derived sensors and user-in-the-loop sensing.

In some cases, where smart meters are available, a home gateway will periodically send power meter readings from the household directly to the system without user intervention. In other cases, users may manually input meter readings using a cell phone. Transportation choices can be recorded by combining user input with GPS information from their cell phone. Information on products purchased and recycled can be entered by sending SMS messages with details to the system. Sense Tecnic will process this information to obtain meaningful metrics for users to understand and measure their sustainability performance.

Recognizing that sustainability choices are not primarily driven by technology, but rather by social drivers, we are combining this real world data with a social networking application to allow users to share advice and tips on

3 http://pipes.yahoo.com/pipes/

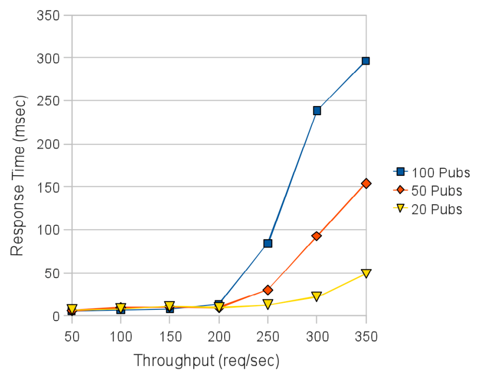

Figure 6: Response-time to throughput in MB2 as the number of publishing clients increases

sustainability issues and to compare themselves with friends and neighbours to gauge their impact and effect.

V. Evaluation

To evaluate the MB2's suitability for device-to-device interaction, we measured the MB2's performance for brokering events and retrieving state. Note that our tests do not measure the bandwidth and latency in a typical MB2 deployment using a wireless network, rather they provide a baseline for understanding the MB2 contribution to these measures. We then gathered information from application developers experience to gain an understanding of the suitability of system for building IoT applications.

A. Performance

To assess the upper limits of throughput and understand potential bottlenecks in our MB2 implementation, we conducted several experiments with the broker. We used three PCs connected in a local area network with 1 GB/s Ethernet cables. Two of the PCs were used to launch clients and had the following specifications: Intel dual core 3 GHz CPU, with 3GB of RAM under Ubuntu Karmic 9.1 and kernel 2.6.31. The third PC, used to launch MB2 and the MySQL database server, had the following specifications: Intel dual core 3.4 GHz with 2GB of RAM under Ubuntu Jaunty 9.04 and kernel 2.6.28.

For the first experiment, we measured the throughput for MB2. Our experiment follows a typical use case for MB2 in IoT applications in which tens of devices in a constrained area (e.g. within a room) publish events to the broker while only a few devices subscribe to events. This is generally the case with applications consisting of assemblies of devices (e.g., cell phones and screens). Phones and sensors publish events to the broker for situated displays. In Sense Tecnic, many sensors publish to the broker for fewer visualizations and the processing engine. In our first experiment we launched 10, 25, and 50 publishing clients on each client machine (a total of 20, 50, and 100 clients) to publish events to the broker at rates of 50 to 350 events per second. In all cases, events are also written to the MySQL database on the same host as MB2. One client on each client machine subscribes to the channel that publishing clients were sending events to (a total of 2 subscribed clients). Each experiment was conducted for a total of 25000 published events and was repeated three times to assure consistent

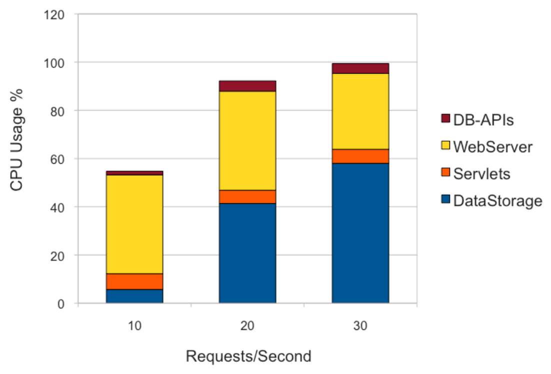

Figure 7: MB2 bundle CPU usage with ten publishers and one subscriber.

results. Figure 6 shows the response time for the broker as the number of publishing clients increases. For 20 publishing clients, the overall response time from MB2 for 350 events/sec was below 50ms. As expected, the response time increases with the number of publishing clients. We account some of the delay to the client machines where threads on those machines compete for CPU and bandwidth in communicating with the broker.

To better understand MB2 throughput bottlenecks, we measured the proportion of allocated CPU use for each of the MB2 bundles using a profiler. We decreased the number of concurrent threads sending requests to the profiler since the profiler adds an order of magnitude of overhead to MB2's performance. We tested the performance of MB2 with 10 publishers each sending requests to the one subscriber at 10, 20, and 30 requests/second. The results are shown in Figure 7. From this graph, it can be seen that the MB2 storage module becomes the dominating bundle between 20 and 30 requests per second. Note also that the overall CPU use hits 100% at these rates due to the overhead introduced by the profiler. Overall throughput diminishes at this point; the test server and profiler can no longer keep up with the rate of requests and the web server component actually does less work while waiting for storage. At 30 requests per second the storage module uses more than 60% of the allocated CPU. Based on this, we can account for most of the overhead measured in the previous experiments (Figure 8) at higher throughput rates (> 200 requests per second) to the data storage module, a new feature in MB2 over the previous implementation.

In a third experiment using the same setup we measured the time it takes for clients to retrieve historical events from channel state. This facility is used by the Sense Tecnic platform to populate graphs on end-user dashboards with past sensor readings. We measured the latency of requesting the last *n* events in a channel (15, 100, 500, and 1000) as the number of client requests for unique channels increases (10, 50, and 100 channels) as shown in Figure 8.

We copied the MB2 database used by our current STS deployment to our test server machine for this experiment. The events in each channel corresponded to the latest temperature in different cities around North America. Each event contained 7 key/value pairs containing a timestamp, the name of location, latitude, longitude, temperature, the ID of the sensor and channel name. When clients request the last 15 events, MB2 responds quickly (36ms is the maximum

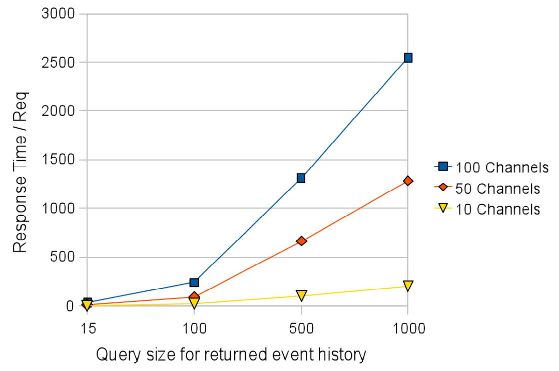

Figure 8. Event history performance.

response time when requesting history from 100 channels concurrently). As the number of returned events increases from 100 to 1000 events, we see a linear increase in the response time. In the worst case, it takes about 2.5s for a request for 1000 events to be fulfilled with 100 concurrent clients/unique channels.

Based on these results, communication delays on a wireless network would be minimal compared to the latency introduced by MB2 under high load.

B. *Developer Experience*

To evaluate how difficult it has been to learn and use the MB2 abstractions we created a questionnaire and interviewed seven developers from our lab that were not involved in the development of our platform. Participants were asked to report their impressions of the system and were asked several open questions about learning to use MB2 for IoT applications. We asked the developers how easy or difficult was it to learn/use the REST web service API/Protocol on a scale from 1 – very difficult, to 5 – very easy. Most developers found it fair to easy to learn (mean 3.42). Once they were familiar with the concepts, developers also found the APIs and protocol easy to use (mean 3.71). The developers we spoke to took between a few days to at most 3 weeks to develop the first working version of their application. Languages used for applications included Python, Java, Flash, PHP, Javascript and Java, and combinations of these.

"Once I learned how everything worked (what needed to be sent and what *I should receive back) it was very easy".*

Application developers all made use of the channel and event abstraction. The next common abstraction was the use of *state* to store information on the server such as received votes, past sensor data or game high scores. More complex applications used most of the MB2 capabilities including content storage. We noted that a typical development strategy is to use MB2 first as a simple event relay and then make use of additional facilities for state persistence, content, and the use of outside services as the need arose.

Informal discussions and code reviews indicated that developers who used the state abstractions typically found the storage of simple name/value pairs in a channel too limiting. Some developers wanted to use the channel to save and query *lists* of objects, not just simple name/value pairs. We also noted that the type of data a developer saved in channel state corresponded to either an event received in a channel (e.g. sensor data or vote) or the result of an event received (e.g. game high score). This indicated a closer affinity between event and state abstractions than we initially realized. While we still see the need for the storage of name value pairs in a channel (e.g. for information about the named channel such as device settings the location of a device), we also saw the recurring need to persist events in the channel for some period of time, for example to populate a message board, or a sensor visualization. We implemented an event history mechanism where applications can request the last N events sent to the channel. To support more complex data structures we allowed state values to contain XML fragments, and suggested the use of state names that follow a sequence such as *score0*, *score1*, *score2*. In some cases, such as in the Sense Tecnic application, the developer abandoned state storage other than event history, and stored information about sensors in a database.

Our own experience with building Sense Tecnic has confirmed for us that MB2 abstractions are not only suitable for application development but also for extending the platform with distributed services. For example, while MB2 provides a simple name/value pair caching system for state and content, STS added a ring buffer cache for real time sensor events for better visualization performance. Similarly, STS has extended the platform with complex event processing, the use of meta-data and typed attributes. Since we expect these facilities to have general applicability, we expect to add these capabilities to the core MB2 platform in future releases.

The way that an application makes use of the MB2 abstractions is specific to its assembly of things. Based on the application experience highlighted here, and previous research work [14][27] we expect that channel, state, and event naming conventions will be required to that take into consideration the different semantics between collections of things.

Two key implementation decisions have served us well: the support for simple HTTP-based protocols, and OSGi. HTTP is supported universally, means that many devices such as browser-based displays and simple gateways can be written without the need for an immediate MB2 protocol updates. The use of OSGi means that we can introduce support for the protocol removing the need to maintain these gateways simplifying deployments.

VI. Related Work

The MAGIC Broker 2 is related to middleware for pervasive and ubiquitous computing as well as platforms targeted specifically for the Internet of Things. In the former category, events as in the iROS system [16] are a key abstraction. Like MB2 events, events in the iROS event heap can persist, but rather than subscribing to named channels as in MB2, iROS events are retrieved using content-based filtering. Other ubicomp systems such as Gaia [28], and one.world [23] also expose events as a key abstraction. Gaia and one.world extended the basic event abstraction with support for shared persistent interaction state and services. MB2 contains events, state, content and services in a named *channel* similar to one.world's *environment* container.

Like other platforms for IoT applications (e.g. [20]), we leverage OSGi to communicate directly to devices without the need for gateways. Unlike OSGi alone, we provide a set of higher-level reusable abstractions, rather than arbitrary services and support for streams. Like Hydra [13] MB2 hides the heterogeneity of devices and their underlying protocols. In contrast to Hydra and others (e.g. [12]), we do not attempt to provide an ontology or standard profiles for classes of things [18]. We allow application developers to decide on the naming and organization of things that comprise their applications.

Instant messaging platforms have been used for the IoT [8], [30]. Like IM systems, MB2 provides a simple unified model for things on the Internet; however, in IM systems the state (or presence) of end points such as devices or users is brokered by the system, not stored in the system as it is in MB2 channel state. Other abstractions such as content and services are either not supported, or layered on top.

VII. Conclusions

In this paper, we have presented a lightweight middleware platform called MAGIC Broker 2 for the Internet of Things. MB2 was used to develop IoT applications that support spontaneous interaction between things such as mobile phones, public displays and shared sensor/actuator networks. We have highlighted flexibility and adaptability as key requirements for supporting these classes of applications. The MB2 platform addresses these fundamental requirements by providing a set of simple abstractions: *events*, *state*, *services*, and *content* organized into collections of IoT application defined *channels* all running on a run-time extensible platform.

The current implementation of a single-server MB2 has served us well for our target deployments where throughput has not been critical; that said, we see opportunities for improvements in addressing security and scalability for larger scale deployments, neither of which have been a focus of our work to date. We plan to evolve the MB2 functionality in these directions to meet the growing needs of end users and applications as more things are connected to the Internet.

VIII. Acknowledgements

We thank Aiman Erbad, Matt Finke, and Vincent Tsao for their work on MAGIC Broker and applications. Financial support was provided by NSERC.

[1] A. Dada and F. Thiesse, "Sensor Applications in the Supply Chain: The Example of Quality-Based Issuing of Perishables," IoT, 2008.

[2] S. Krishnamurthy, et al., "Automation of Facility Management Processes Using Machine-to-Machine Technologies," IoT, 2008.

[3] A. Tang, M. Finke, M. Blackstock, R. Leung, M. Deutscher, and R. Lea, "Designing for bystanders: reflections on building a public digital forum," CHI 2008, Florence, Italy: ACM, 2008, pp. 879-882.

[4] M. Finke, A. Tang, R. Leung, and M. Blackstock, "Lessons learned: game design for large public displays," *Digital Interactive Media in Entertainment and Arts*, Athens, Greece: ACM, 2008, pp. 26-33.

[5] A. Erbad, M. Blackstock, A. Friday, R. Lea, and J. Al-Muhtadi, "MAGIC Broker: A Middleware Toolkit for Interactive Public Displays," PerWare 2008 Workshop at IEEE PerCom 2008.

[6] N. Kaviani, M. Finke, S. Fels, R. Lea, and H. Wang, "What goes where?: designing interactive large public display applications for mobile device interaction," *Internet Multimedia Computing and Services*, Kunming, Yunnan, China: ACM, 2009, pp. 129-138.

[7] T. Paek, et al., "Toward universal mobile interaction for shared displays," CSCW 2004, Chicago, Illinois, USA: ACM, 2004.

[8] E.M. Huang, D.M. Russell, and A.E. Sue, "IM here: public instant messaging on large, shared displays for workgroup interactions," CHI 2004, Vienna, Austria: ACM, 2004, pp. 279-286.

[9] A. Kansal, S. Nath, J. Liu, and F. Zhao, "SenseWeb: An Infrastructure for Shared Sensing," IEEE MultiMedia, 2007.

[10] M. Mun, et al., "PEIR, the personal environmental impact report, as a platform for participatory sensing systems research," MobiSys 2009.

[11] E. Welbourne, et al., "Building the Internet of Things Using RFID: The RFID Ecosystem Experience," IEEE Internet Computing, vol. 13, 2009, pp. 48-55.

[12] A. Katasonov, O. Kaykova, O. Khriyenko, S. Nikitin, and V. Terziyan, "Smart semantic middleware for the internet of things," Proceedings of the 5-th International Conference on Informatics in Control, Automation and Robotics, 2008, pp. 11-15.

[13] M. Eisenhauer, P. Rosengren, and P. Antolin, "HYDRA: A Development Platform for Integrating Wireless Devices and Sensors into Ambient Intelligence Systems," IoT Workshop on Digital Communications, 2010, pp. 367-373.

[14] M. Blackstock, R. Lea, and C. Krasic, "Evaluation and Analysis of a Common Model for Ubiquitous Systems Interoperability," Pervasive 2008.

[15] N. Davies, S.P. Wade, A. Friday, and G.S. Blair, "Limbo: a tuple space based platform for adaptive mobile applications," C*onference on Open distributed processing and distributed platforms*, 1997.

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

[17] Lea, R., Gibbs, S., Dara-Abrams, A., and Eytchison, E. 2000. Networking Home Entertainment Devices with HAVi. Computer 33, 9 (Sep. 2000), 35-43.

[18] UPnP Forum, "UPnP Device Architecture version 1.1," Oct. 2008.

[19] OSGi Alliance, "OSGi Service Platform Release 4," 2007.

[20] J. Rellermeyer, M. Duller, K. Gilmer, D. Maragkos, D. Papageorgiou, and G. Alonso, "The Software Fabric for the Internet of Things," IoT, 2008.

[21] S. Greenberg and M. Rounding, "The notification collage: posting information to public and personal displays," CHI 2001.

[22] O. Storz, A. Friday, N. Davies, J. Finney, C. Sas, and J. Sheridan, "Public Ubiquitous Computing Systems: Lessons from the e-Campus Display Deployments," IEEE PerCom 2006, vol. 5, 2006, pp. 40-47.

[23] R. Grimm, "One.world: Experiences with a Pervasive Computing Architecture," IEEE PerCom 2004, vol. 3, 2004, pp. 22-30.

[24] L. Arnstein, et al., "Systems support for ubiquitous computing: A case study of two implementations of Labscape," Pervasive 2002.

[25] M. Finke, M. Blackstock, and R. Lea, "Deployment Experience Toward Core Abstractions for Context Aware Applications," Smart Sensing and Context 2007.

[26] "Esper - Complex Event Processing.", http://esper.codehaus.org/ retrieved September 10, 2010.

[27] M. Blackstock, R. Lea, and C. Krasic, "Managing an Integrated Ubicomp Environment Using Ontologies and Reasoning," CoMoRea Workshop at IEEE PerCom 2007

[28] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R.H. Campbell, and K. Nahrstedt, "Gaia: a middleware platform for active spaces," MobiCom 2002, pp. 65-67.

[29] J. Choi and C. Yoo, "Connect with Things through Instant Messaging," The Internet of Things, 2008, pp. 276-288.