JADE: A Unified Programming Framework for Things, Web, and Cloud

Debashish Ghosh, Fan Jin and Muthucumaru Maheswaran
School of Computer Science, McGill University, Montreal, Canada
Email: {debashish.ghosh, fan.jin}@mail.mcgill.ca, maheswar@cs.mcgill.ca

***Abstract*—In this paper, we present JADE, a framework that allows a developer to mix C and JavaScript constructs with JADE supplied keywords to construct a complete program to solve a particular computing activity in a 'thing'. We describe the language constructs introduced by JADE and explain how they can be used to implement different software interaction patterns among the thing, web, and cloud. We perform an evaluation of the JADE framework over Intel Galileo and cloud. The results from the experiments are described in the paper.**

I. Introduction

A recent forecast made by International Data Corporation (IDC) projected Internet of Things (IoTs) and the associated ecosystem to be an $8.9 trillion market by 2020 and include 212 billion connected things. To realize such a scale of expansion, it is not sufficient to just create the devices and deploy them, the associated ecosystem needs to be developed and users need to embrace IoT. Users will start embracing IoT only when compelling applications that solve real problems are available for IoT. Therefore, it is important to focus on creating programming frameworks that will allow developers to easily develop high-quality applications for IoT.

Due to various factors, programming IoT is different from normal computers. One difficulty is that things may not be capable of processing a complete computing activity by themselves. They may need the computational capabilities of cloud-based backend to complete the processing tasks and a web-based frontend to interact with the user. Therefore, programming the thing can be quite complicated by the fact the programmer needs to deal with disparate elements to complete an activity. Another difficulty is the mobility of things or other factors that can make things less available. Yet another difficulty and perhaps the hardest is the interoperability of the things.

In this paper, we present a unified programming framework called JADE for IoT. JADE is a hybrid language that uses few keywords to mix C/C++ with JavaScript. For instance, we can have JavaScript implementation of a C function. The JADE preprocessor separates the JavaScript and C/C++ code and makes the necessary linking between the corresponding functions. Using JADE, we can write a program for a thing that partly runs on the thing and partly runs on the web or cloud. The execution of all components are coordinated by the portion of the program that runs on the thing. The C/C++ portion augmented with glue code is compiled to run on the thing. Similarly, the JavaScript portion and the corresponding glue code is injected into the web or cloud. With the rapid ascension of NodeJS in cloud computing, it is conceivable that cloud-based backends can run JavaScript code that is injected by the JADE program.

JADE makes it simpler to develop many IoT applications by bringing the well known "single node view" to IoT programming. It fuses the thing, web, and cloud such that cloud and web can be accessed via remote procedure calls implemented by the JADE runtime. For instance, an intelligent thermostat application can be developed in a single set of source files written in JADE. The C/C++ portion does sensing and control of the thermostat functions. The JavaScript portion is responsible for storing the sensed values in the clouds, running the prediction algorithms on the cloud, and realizing the user interface on the web. Using JADE, the intelligent thermostat developer would not be restricted by any preconceived service interfaces. She is free to program all elements according to the needs of the application at the same time reusing existing libraries and services.

In Section II we motivate the design of the unified programming framework for IoT. Section III presents the design of the JADE language along with brief examples of using important constructs in JADE. In Section IV we evaluate the JADE framework under a typical IoT setting consists of sensor node and the cloud. Related work is explained in Section V.

II. Motivation

Programming IoT presents many unique challenges. One of the key problems for programming is the heterogeneous nature of IoT with heavily constrained sensors in one extreme and fully featured smart devices in the other extreme. While the resource-constrained sensors can be dedicated to a specific computing activity, the fully featured smart devices could be hosting apps that allow them to perform different tasks at users' discretion. Therefore, the debate rages on regarding the programming models that should be supported by IoT.

There are three major ways for approaching the programmability problem in a heterogeneous distributed system such as the IoT: use service-oriented computing, develop a new programming language, and develop a library that exposes a uniform interface on all devices. The Web of Things (WoT) [1] is a variation of IoT that implements the service-oriented computing paradigm. In service-oriented computing, each "service endpoint" is maintained by a service provider to expose services that are used frequently by other applications. It is not an ideal paradigm for exposing functions that are not used by many applications. If a function is just used by a single application, the authentication and authorization processes can be an undue burden on the developer.

Over the years, many programming languages have been developed for IoT like environments [2]. Although a programming language that is designed for IoT can bring the optimal set of features, it will also have many drawbacks. The most important among them is the resistance from the developers to learn an entirely new language. Also, implementing the new language on all constituent devices is another major challenge. Due to these reasons, most proposals for new languages for heterogeneous distributed systems have remained academic curiosities.

Another powerful approach to programming IoT is to develop libraries that run on all constituent devices and support a uniform interface for communications. The AllJoyn [3] framework from Qualcomm is one such example. The biggest advantage of this approach is that the developer need not learn a new language; instead, she needs to learn the library APIs. The drawback is the effort needed in maintaining the library on all devices. Because the libraries need to be packaged with the operating system and often require hardware support, the device manufacturers need to engaged in the development of the libraries. Another significant drawback of this approach is its incompatibility with the web. The web browsers have very restricted ways of interacting with other components. Therefore, a program running in a browser cannot utilize such libraries.

The framework we propose here is quite novel and includes elements of all of the above approaches. Following are some of the key objectives we wanted to achieve in designing the framework.

1) *Leverage existing trends:* IoT is in a highly dynamic technology sector. Therefore, it is necessary to leverage existing trends in developing the programming framework. In particular, the programming framework should reuse technologies that are heavily used by the developers in the sector.
2) *Lightweight framework:* The framework should have small resource footprints so that it could be ported to highly resource-constrained devices.
3) *Flexible function partitioning:* A programmer of heterogeneous devices such as IoT can immensely benefit from a flexible function partitioning mechanism that allows her to easily deploy custom code at the web or cloud. Using such a functionality the programmer can easily create computing activities that incorporate the thing, web, and cloud.
4) *Gentle learning curve:* We can expect many programmers to be highly proficient on well known languages such as JavaScript, C, C++, and Java. By creating a framework that combine snippets of existing programming languages such as these, we can provide a gentle learning curve to the IoT programmer.
5) *Support for heterogeneous systems:* We can expect the IoT ecosystem to contain a highly heterogeneous collection of devices. These devices should interoperate among themselves and cloud and web resident services.

III. The JADE Programming Language

A. Overview

In JADE, the developer must write code in a JADE file (`.ja` extension) and then use the JADE preprocessor coded in python. The JADE preprocessor creates a `.c` file for the 'thing', and JavaScript files that can be injected into a NodeJS server or a HTML5 capable browser.

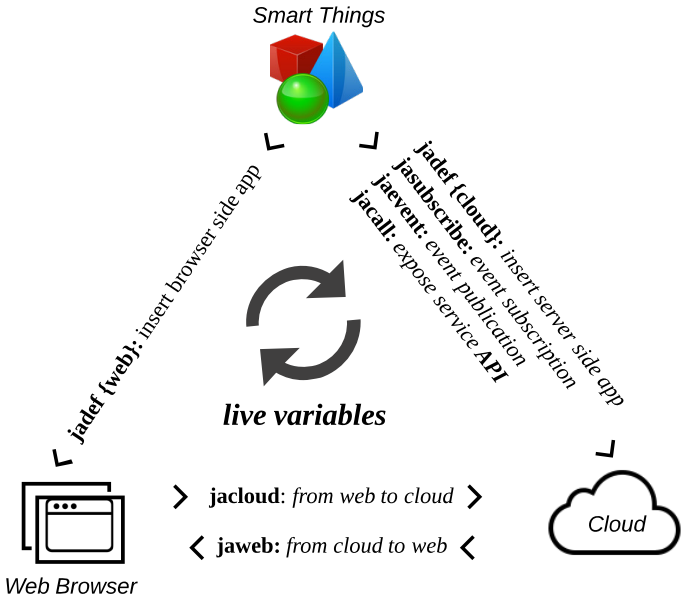

Fig. 1: JADE functions among thing, web, and cloud

B. Functions

1) jadef C function: JADE language allows the developer to code for both the server and the web. The code that is to be passed to the server or the browser, needs to be enclosed within a function. Such a function should begin with the keyword `jadef`, followed by either the keyword `web` or `cloud` enclosed within parentheses depending on where the function will be executed. The function name and parameters follow the ANSI C syntax. Inside the function's body, the code should be written in standard JavaScript syntax. This function is called from the 'thing' and runs in the cloud or the web. Below is the syntax for writing `jadef` function for the web:

```
jadef {web} C_function_prototype()
{
    // JavaScript body
}
```

Listing 1: jadef web function syntax

A simple example `jadef` function obeying the syntax is provided. Notice the C function declaration and JavaScript function body.

```
jadef {web} foo(int xValue)
{
    var x = liveVar("foobar.x", xValue);
    x.onupdate = function(){
        alert("new value of x " + x.value +
            "baz: " + baz);
    };
```

```
}
```

Listing 2: jadef web function example

The cloud server for JADE will be implemented in NodeJS, which runs the JavaScript code from the 'thing'. The 'thing' should also provide its name, so that the functions in the cloud can be associated with the corresponding 'thing'. For sending the JavaScript code to the cloud, the developer should use an additional keyword `cloud` enclosed within parentheses in front of the function name, as shown below:

```
jadef {cloud} C_function_prototype()
{
    // JavaScript body
}
```

Listing 3: jadef server function syntax

2) jarequire: The JavaScript body in the `jadef` function may require some external libraries, global variables or function definitions which could be included inside `jarequire()`. If the user wants to insert HTML5 templates or link CSS files into the HTML5 templates, `jarequire` could be used for this purpose too.

```
jarequire()
{
    // external libraries
    // global variables
    // global functions
    // HTML5 templates.
    ...
}
```

Listing 4: jarequire syntax

In example below, the variable `baz` is defined in the global scope, which allows the `jadef` function `foo` in Listing 2 to access the value.

```
jarequire()
{
    var baz = 10;
}
```

Listing 5: jarequire example

3) jasubscribe Function: Allows the developer to write a function in C/C++ syntax in the 'thing' that gets executed when the event is triggered by another 'thing', browser, or the cloud server. Below is an example of the prototype of the `jasubscribe` function:

```
jasubscribe {event name} c_function()
{
    // code to be executed when the event occurs
}
```

Listing 6: jasubscribe function syntax

4) jacall Function: Using the `jacall`, the developer can mark a function as callable from the cloud or another 'thing'. A JavaScript function signature that corresponds to the C function will be registered at the cloud server as an API supported by the 'thing'. This API can be discovered by other cloud resident functions and invoked, which will result in a callback to the 'thing'.

```
jacall C_function( )
{
    // jacall function calls to interact with
    // other 'thing's through their exposed API
    // 'thing' application code ...
}
```

Listing 7: jacall function syntax

5) jaevent Function: As we are using a publisher-subscriber model for event handling, there needs to be a mechanism to publish events to the server so that `jasubscribe` will receive the events. The `jaevent` function allows the developer to define new events or use predefined events in the system. Below is the syntax for `jaevent`:

```
jaevent {event_name} {
    // declare event parameters
}

void c_function(){
    // update event parameters
    raise_jaevent(event_name, event_parameters);
}
```

Listing 8: jaevent and raise_jaevent function syntax

The call to `raise_jaevent` transfers the event to the server. The server notifies the subscribers of the event when it gets published. At the system level, a data structure is maintained for storing the events and its parameters. A hierarchical naming convention is followed underneath to help identify different events. For instance `event.user.eventButton` could be used to identify an event called `eventButton` which is created by the developer. While `event.system.mouseMoveEvent` could identify a system event that is used by the developer.

C. Live Variable

JADE allows the programmer to *tie* a variable in the 'thing' to a variable in the cloud or web. This provides a much simpler way of obtaining the latest value of a variable without generating another event for it. To achieve this, we introduce the concept of "live variables."

1) Update triggered by the 'thing': Variables whose value could change are referred to as 'live' variables. They are denoted by the storage class `live` we introduce into the C side.

```
void foobar(){
    live int x;
    x = 5;
    foo(x);
}
```

Listing 9: An example of live variable

The 'live' here is similar to the `static` or `extern` keyword in C. When a variable is declared 'live', if the value of the variable changes elsewhere, the new value will be reflected locally, and vice-versa. While writing the code in JADE, any portion of the code written in C syntax, could include a 'live' variable, except the JavaScript part which is inside the `jarequire` and the `jadef` function for the cloud and the web.

Inside the `jadef` function for the web/cloud, the developer could add the event that should transpire in the web/cloud, when a live variable value update is received from the 'thing', using the syntax shown below.

```
x.onupdate = function() {
    // event upon value update
};
```

Listing 10: onupdate JavaScript function

The `onupdate` function allows the programmer to execute arbitrary JavaScript code when the value of the live variable gets modified in the 'thing'.

2) Update triggered by the web/cloud: The live variable can be updated in either end. Above, we described how the updates propagate from the 'thing' to the web or cloud. Below we describe how updates propagate from the web or cloud to the 'thing'. In the C side, JADE provides a library function called `checkVal()` that allows the programmer to read the current value of the live variable. If there is no updates for the live variable, the `checkVal()` routine will timeout after the given interval. The C function uses the return value to indicate the presence of an updated value.

```
void updateThing()
{
    live int x;
    while(1) {
        int timeout = 1000; //1000 ms
        int xflag = checkVal("x", timeout);
        if(xflag) {
            // new value found for x
        } else {
            // timeout occurred
        }
    }
}
```

Listing 11: checkVal() example usage

D. Implementing JADE

The JADE implementation has two major parts. First is the JADE preprocessor that converts JADE source file to the C/C++, and JavaScript. Second is the JADE runtime that is necessary to integrate all the components together. This part is mainly responsible for message communications and invoking the necessary functions.

1) The JADE Preprocessor: The JADE Preprocessor is written in python. It takes a JADE file with `.ja` extension as input and generates a `.c` and JavaScript files as output. The preprocessor parses the JADE file one line at a time and adds the code to the JavaScript file or the `.c` file determined by the presence of keywords like `jadef`, `jasubscribe`, `jacall`, `jaevent` and `jarequire`.

2) The JADE Runtime: The JADE runtime is made of a library in C, library in JavaScript, and a NodeJS server (currently server is implemented in python). The C library consists of important functions that could be used by the developer while writing the C/C++ part of the code in the JADE language. The C and JavaScript libraries are responsible for implementing the remote procedure call (RPC) [4] functionality so that C functions can call their JavaScript implementations and vice-versa. In addition to the RPC functionality, the libraries also support the live variable updates through a similar but simpler mechanism. In live variable updates, we need to keep track of the functions getting in scope and leaving the scope. Further, the updates for the live variables need to be routed to the appropriate variables when many variables are active at the same time.

One of the features of the JADE runtime is its ability to work with web browsers. That is, the Javascript programs running inside web browsers can use the JADE runtime to communicate with programs running in 'things' and cloud.

IV. Design Evaluation

To evaluate the design of the system architecture in practice, we measured the memory footprint, network transmission efficiency, and network transmission performance between a cloud server and a thing prototype. For the measurements, we used a PC with UBuntu 14.04 running a python server and tools like network analyzer, and an Intel Galileo [5] with JADE runtime deployed as the thing prototype. The program on the thing has been compiled with i586-poky-linux-uclibc-gcc (GCC) version 4.7.2. With memory footprint, device of interest is the Intel Galileo as other devices are not typically resource constrained. The link between thing and cloud is Ethernet, with all devices reside in LAN. The message exchanging protocol at the transport layer is TCP. In terms of M2M communication, JADE resides between the application layer and transport layer in the Internet stack. For transmission efficiency, we measure the efficiency of using JADE as service for communication between IoT devices. The efficiency is measured for different payloads from 0 to 512 bytes passed by the application layer. For transmission performance, we measure the latency of 10,000 messages for different payloads from 0 to 512 bytes. During the evaluation, payload is pushed to cloud in the form of RPC, with payload being the parameter.

A. Memory Footprint

Intel Galileo board runs Yocto based Poky Linux distribution, which supports native Linux based applications. *smem* is used to analyze the memory footprint, and **PSS** is measured. The simulation program on the thing uses JADE for event management and RPC. The goal is to examine memory usage under different loads. The simulation on the thing resembles sensor activities in typical IoT settings, where a sensor node collects data and pushes to the cloud, while periodically receives data from the cloud. The sending rate from thing to cloud varies from $1Hz$ to $100Hz$, and from cloud to thing stays constant at $0.2Hz$. The sending rate will be referred as activity, and is normalized against $100Hz$ in column 1 below. The payload size is fixed to 100 bytes both ways.

Activity	Scenario A	Scenario B	JADE API Overhead
1%	144.0 kB	144.0 kB	∼0 kB
10%	144.0 kB	144.0 kB	∼0 kB
20%	144.0 kB	144.0 kB	∼0 kB
50%	144.0 kB	148.0 kB	∼4 kB
100%	148.0 kB	152.0 kB	∼4 kB

TABLE I: Memory usage for simulation program under different loads

The memory overhead incurred by making JADE library calls at different rates is examined. In both scenario, simulation program collects data after receiving data from cloud, and simultaneously, pushes data to cloud in the form of RPC at different rates. The *difference* is, in **Scenario A**, the JADE library function used to push data to cloud is *void*; but is *valid* in **Scenario B**. The measurement shows *proportional set size* measured for the simulation process. Under light activity, JADE library calls incur insignificant memory overhead. Whereas, under moderate and heavy activity, overhead is 4 kB.

B. Network Transmission Efficiency

The simulation program on the thing pushes payload of different sizes to cloud through JADE. The goal is to examine data transfer efficiency using JADE as service for communication. For each message, the size of payload passed by the application layer to JADE is compared against size of payload JADE passed to the transport layer.

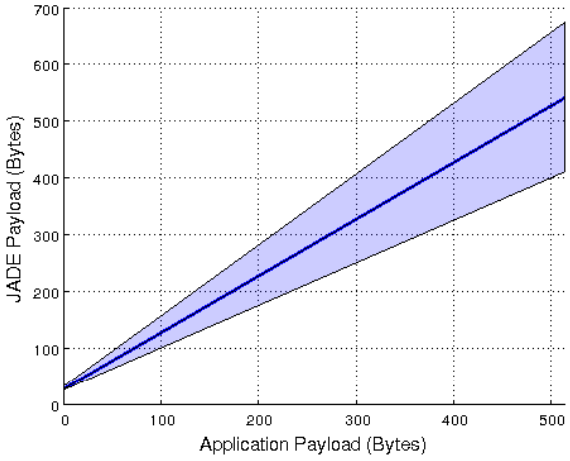

Fig. 2: Data Transmission Efficiency

JADE encodes each message to be passed in the JSON format [6], which incurs a slight transmission overhead. The JSON object is transferred over the network as a string and is evaluated at the receiving end.

C. Network Transmission Performance

The latency is measured in terms of TCP round-trip time for different payloads from thing to cloud. During the experiment, sending rate from thing to cloud remains at $20Hz$ to prevent buffer overflow and network congestion. The goal is to examine network transmission performance using JADE as service for communication. For payloads until 256 bytes, mean latency remains under $1ms$. However, mean latency increases to $37.4ms$ for payloads of 512 bytes. This suggests, to achieve optimal transmission performance, application should use data compression scheme as necessary when using JADE as service for communication.

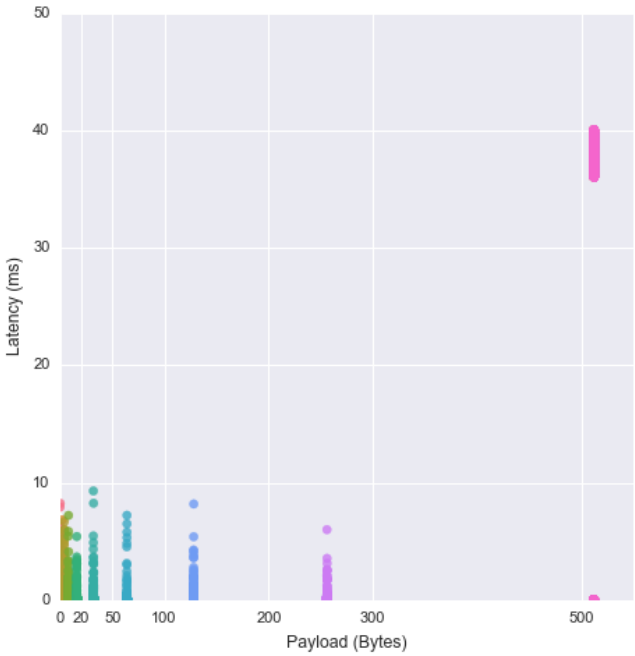

Fig. 3: Network Transmission Latency

V. Related Work

JADE has been created to aid in the process of developing an efficient framework for IoT. This section compares and contrasts JADE with other services which have made similar attempts.

REST [7] enables component interactions in a layered client server style architecture with the added constraint of a generic resource interface. REST typically uses a client server architecture with a stateless protocol, usually HTTP. In an analogous manner to REST [8], JADE also allows component interactions through the functions defined in the built-in library. Through the `jadef {web}` and `jadef {cloud}` functions, a web/cloud based function could be called directly from the 'thing' without using any third party API.

JADE has a similar approach to SOAP [9], but instead of using the envelope syntax for sending XML messages, JADE wraps up the message to be sent in a JSON object, which is parsed at the server or the web, and the appropriate function is called. Using this approach, the message is less convoluted and there is lesser content to be transmitted compare to SOAP. The disadvantage of using JSON is that it is limited in functionality, whereas XML is generic, unconstrained and much more flexible.

It is important in the vision of IoT, to allow adding sensors to existing smart infrastructures. For example, tiny web services [10] use existing web services in a lightweight manner. A sensor node reports its interface using the Web Service Description Language (WSDL). Applications that want to use the sensor, sends the sensor the message specified. Advantage of this technique is that application developers only need to know the semantics of the sensor, while the WSDL handles the task of generating method calls in a high

level language (easy to use). Requirement of sleep for devices to save power could be achieved by using Web Services eventing [11]. Similar to Tiny Web Services, JADE application developers only need to know the semantic of the sensors or the interface for the 'things' that they will interact with, while the underneath architecture takes care of generating method call using internal libraries.

SpaceBrew [12] is an open software toolkit which helps in creating an IoT environment through a publisher-subscriber system. In contrast to SpaceBrew, where the publisher-subscriber system is user driven, the system in JADE is developer driven. Instead of the user who must specify the semantic relationship between events, the developer can do so. The advantage of JADE is that by allowing access through the web, JADE provides remote access and telepresence in a more effective manner.

The vision of End User Programming follows three major steps. Firstly, user should be involved in the design process. The platform should be human centric and only moderately abstract. Secondly, provide an extensible platform for matching conditions to actions. Performing actions depends on the availability of hardware. Each smart environment is distinct. The paper [13] suggests to use a single software framework, which could be extended to accommodate additional software or hardware functionality. Lastly, to build on powerful, widely deployed web protocol. The platform server could be written on any of the existing web languages. Implementing ubiquitous programming through JADE framework could achieve most of the desired objectives of End User Programming. For a space which allows Ubiquitous Computing, using the JADE framework, the developer could include the end user during the design process. An extensible platform could be then created, where any new device could be added. Also, JADE uses JavaScript which is the most widely used scripting language for the web and would fit in well with the proposed vision for end user programming to widely deploy web protocols.

VI. Conclusions and Future Work

Internet is poised to undergo a major transformation in terms of number of devices and services due to the induction of Internet of things (IoT). To fully realize the promise of IoT, we need programming frameworks that will allow developers to work on IoT with familiar tools. To achieve this, we propose JADE that constructs a programming framework using C/C++ and JavaScript. The ideas used in this synthesis could be applied to used to create a version that mixes Java with JavaScript as well. The approach of using a preprocessing step and leaving the actual compilation to the "native" compiler has its benefits in long term maintenance of the tools and the application code.

To further simplify the application development for IoT, we introduced the ideas of 'live' variables. Although the functionality offered by the live variables can be implemented by the event processing scheme in JADE, the live variables provided a much easier usage pattern for the programmers.

We presented an evaluation of the framework over 'thing' prototype and the cloud, detailing memory footprint, network transmission efficiency, and network transmission performance.

As part of the future work, we will implement and deploy the JADE framework on other IoT platforms, e.g. based on Contiki and TinyOS. Another interesting future direction is to use JADE in newer operating systems such as Tizen that have the "web app stack" and "native app stack." Using JADE, we could develop "hybrid apps" that use the web and native features in a single device or across multiple devices.

References

[1] D. Guinard, "A Web of Things Application Architecture – Integrating the Real-World into the Web," Ph.D., ETH Zurich, 2011.

[2] R. Grimm, J. Davis, E. Lemar, A. Macbeth, S. Swanson, S. Gribble, T. Anderson, B. Bershad, G. Borriello, and D. Wetherall, "Programming for pervasive computing environments," University of Washington, Tech. Rep., 2001.

[3] R.-C. Marin, "Hybrid contextual cloud in ubiquitous platforms comprising of smartphones," *Int. J. Intell. Syst. Technol. Appl.*, vol. 12, no. 1, pp. 4–17, 2013.

[4] A. Birrell and B. J. Nelson, "Implementing remote procedure calls," *ACM Trans. Comput. Syst.*, vol. 2, no. 1, pp. 39–59, 1984.

[5] Introducing the intel galileo development board. [Online]. Available: http://www.intel.com/content/www/us/en/do-it-yourself/galileo-maker-quark-board.html

[6] D. Crockford, "Javascript object notation." see http://www.json.org/.

[7] R. T. Fielding, "REST: architectural styles and the design of network-based software architectures," Ph.D., University of California, Irvine, 2000.

[8] H. Hamad, M. Saad, and R. Abed, "Performance evaluation of restful web services for mobile devices." *Int. Arab J. e-Technol.*, vol. 1, no. 3, pp. 72–78, 2010.

[9] K. A. Kadouh and K. A. Albashiri, "Improvement of data transfer over simple object access protocol (soap)," *International Journal of Computer, Information Science and Engineering*, vol. 8, no. 2, pp. 16 – 19, 2014.

[10] B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao, "Tiny web services for sensor device interoperability." in *IPSN*, 2008, pp. 567–568.

[11] "BEA, microsoft, and tibco release web services eventing (WS-eventing) specification." 2004.

[12] About spacebrew. [Online]. Available: http://docs.spacebrew.cc/about/

[13] S. Holloway and C. Julien, "The case for end-user programming of ubiquitous computing environments," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*, ser. FoSER '10. ACM, 2010, pp. 167–172.