SpaceOS: Operating System Services for Smart Computing Environments

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

***Abstract*—SpaceOS is a system software stack for smart computing environments. The major objective of SpaceOS is to make it easier to create applications and deploy them in smart computing environments. It provides several core services for managing smart devices in physical spaces and letting them interact with cloud backends and web-based users. SpaceOS also supports a programming language called Jade that leverages the facilities provided by the system software stack to provide a simpler programming model for the things.**

I. Introduction

The next expected frontier in the evolution of the Internet is the subsumption of physical objects and the spatial interactions with them into the Internet. Including physical objects into the Internet opens up new possibilities for remote monitoring and controlling of physical environments. For example, already smart bulbs can be controlled via Bluetooth-enabled smartphones without flipping a light switch. With appropriate enhancement of networking and security provisioning, the capability to control the smart bulb can be offered to a remote user on the Internet. Similarly, a remote but smart environment can be monitored using the readings streamed out by the sensors embedded in the environment.

Like other large-scale systems before it smart computing environments need a system software stack based on some organizing principle. For instance, cloud computing systems have many cloud stacks. The grid computing systems before them have their own toolkits and the Internet has its famous TCP/IP protocol stack. In the Internet, the TCP/IP stack is an implementation that provides the necessary functions for a device that is part of the Internet to communicate with another one. The cloud stacks on the other hand are more like control platforms that allow the orchestration of the activities among the physical nodes that are part of a cloud platform. A smart computing environment is bound to have nodes that fall into at least three types: things, cloud, and web. Unlike in other computing scenarios, in smart computing environments the capabilities of the nodes are asymmetrical. For example, a thing may not have sufficient capability to completely process the data it acquires. As a result, the thing needs to work with a cloud service to fulfill its processing requirements and use the web for user interaction purposes. Definitely, the task of facilitating these interactions falls onto the system software stack that is created for smart computing environments.

At McGill, we are developing a system software stack for smart computing environments called SpaceOS. In developing the SpaceOS, we have several unique requirements. First, we want to reuse existing technologies as much as possible to increase the developer familiarity. For instance, things are already programmable using C or C++ and the web/cloud is already programmable using Javascript. By reusing these languages, we reduce the barrier for entry. Another requirement for the SpaceOS is the single system image for the programmer. Because a thing does not have substantial computing capability, it cannot be programmed as a traditional computer; in most situations it runs a small portion of a distributed program. As SpaceOS evolves, we expect to provide a single system image and the distributed nature of the smart computing environment would be transparent to the programmer.

In Section 2 we present an example smart computing scenario to motivate the particular issues that are being dealt by SpaceOS. A system architecture for SpaceOS is explained in Section 3. In Section 4, we present the Jade language that is developed as part SpaceOS. Example applications that can benefit from SpaceOS are presented in Section 5. Finally, on section Section 6, we discuss work related to SpaceOS.

II. Example Smart Computing Scenario

Fig. 1 shows an example smart computing scenario that is motivating the SpaceOS design. A room with many smart devices including smartphones carried by people is shown in the figure. The smart devices are in the physical space. The simplest way of capturing the relation to the physical space of a device is to locate it in the physical space by determining the coordinates of it in a physical frame of reference. If the device is outside, Global positioning system (GPS) coordinates can be used. If the device is inside, indoor positioning system (IPS) coordinates can be used. The problem with using existing coordinate computation schemes is the error values associated with the coordinates.

There are two ways we could use location in interacting with a roomful of smart devices. First is to map the devices to a web portal associated with the physical location and control the devices using the web portal. The locationing problem is much easier in this case. We need to figure out whether the devices belong to a room or not. Determining the location of a device (that is in which room it is present) does not need computing the coordinates. It could be determined by merely evaluating the proximity of the device to known landmarks in the different locations.

Second is to map the devices onto a video frame. We assume that a video camera is streaming the physical scene containing the smart devices. The objective of this form of locationing problem is quite hard – it should be sufficiently accurate enough to position the smart device in the correct position in the frame of the video. For example, Alice and Bob

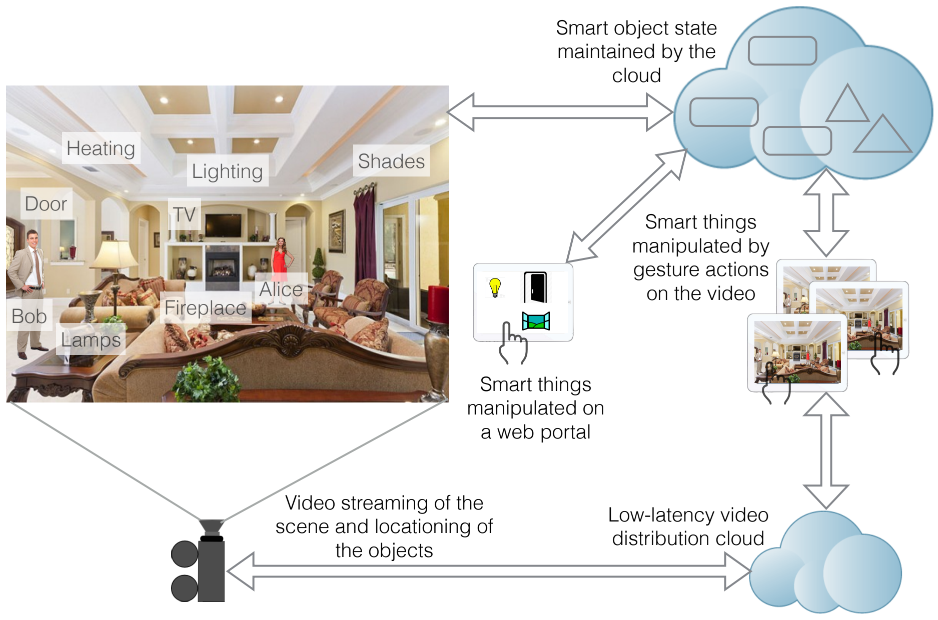
Smart object state maintained by the cloud
Heating
Lighting
Shades
Door
TV
Alice
Fireplace
Bob
Lamps
Smart things manipulated by gesture actions on the video
Smart things manipulated on a web portal
Video streaming of the scene and locationing of the objects
Low-latency video distribution cloud

Fig. 1: Example smart computing scenario

in Fig. 1 are carrying smartphones which should be mapped to the area occupied by Alice and Bob, in the video frame.

By using the video, we provide two channels for a physical object to communicate with the remote user. For example, a smart lamp could present a menu with control options and notifications for the user to control the lamp and know its status in addition to what the user can see in the video. The video acts as a redundant channel. With other smart devices, such as Alices smartphone the video could be an augmenting channel. For instance, a remote user can message the smartphone while observing Alices actions on the video. While the inclusion of the video makes the locationing problem harder because a higher level of precision is needed, it also introduces new ways for locationing. With the video, instead of solely relying on computational schemes for locating smart devices, user assisted locationing can be used as well.

There are several important challenges in realizing the smart computing scenario described in this section. First, developers should be able to create applications for the smart computing environment that will have portions running on the thing, cloud, and the web. The challenge is to keep the programming task simple while making the distributed application secure, lean, and responsive. Second, a smart computing environment consists of many smart devices that need to exchange data and coordinate their activities. In Fig. 1, Bob might want to control the lights using an application in his smartphone. For this activity, the smartphone should be interoperable with the control and monitoring functions exposed by the lighting system. Further, Fig. 1 shows two different ways of presenting the smart computing environment to the user. In either way, the interface should be user-friendly, correctly trigger the actions on the 'thing' and provide feedback to the user on her actions.

III. System Architecture

A. Overview

The services provided by the SpaceOS architecture are organized in a stack as shown in Fig. 2. The *Core* part has services that run on all elements: thing, web and cloud. These services are responsible for naming and discovery. The *Foundation* part has services that are specific for the different elements. The foundation services on the thing are responsible for functions such as locationing and service discovery. The foundation services run on the web and cloud as well. Their functions change depending on the element. The *Application* part has services that are created by the developer using the Jade language.

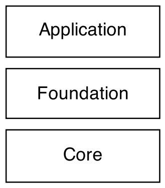

Fig. 2: SpaceOS software stack

SpaceOS has cloud-centric architecture, where the nodes in the network communicate through services hosted in a cloud platform. Therefore, it is important to have fault-tolerant communication services built into SpaceOS such that the smart computing environment can at least have partial capabilities even when the connectivity to the cloud is lost.

The SpaceOS lies on the application layer, abstracting physical objects connected to the network and creating a logical representation of them. All communication with 'things' and web users is done through sockets. For web users we are currently using web sockets and for 'things' we use TCP sockets. In the case where our smart environment has a video channel, such as a webcam, the SpaceOS cloud server assists the setup of direct connection between web users and the video stream. This approach minimizes possible latency issues which otherwise arise through an intermediate. To encapsulate the complexity of interacting with 'things' and cloud through the network we provide the Jade language. The Jade language is a mix of C and javascript which abstract complex network tasks to make more intuitive the usage of the cloud processing power and interaction with other things, see Section IV for details.

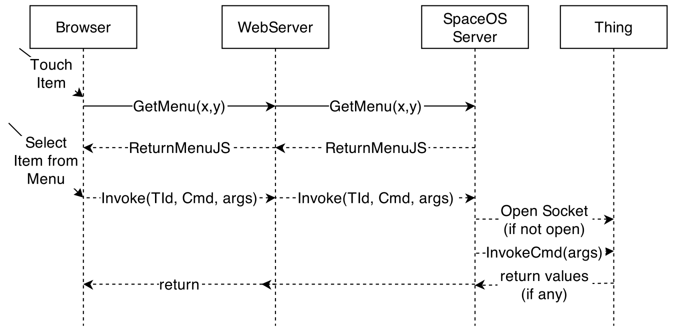

Fig. 3: User interacting with the Thing

Fig. 3 shows a sequence diagram of a simple user interaction through the browser. This example assumes that the environment has a video stream available and that the 'thing' was already mapped to its x,y coordinates in the video frame. First, the user selects the thing that he/she wants to interact with. That will trigger a request to get a list of available actions for that object. For example, if we were interacting with a smart bulb, the list would return TurnON() and TurnOFF(), assuming these are the only methods available. Once the user selects one of the actions the request is forwarded to the SpaceOS cloud server and then to the 'thing'.

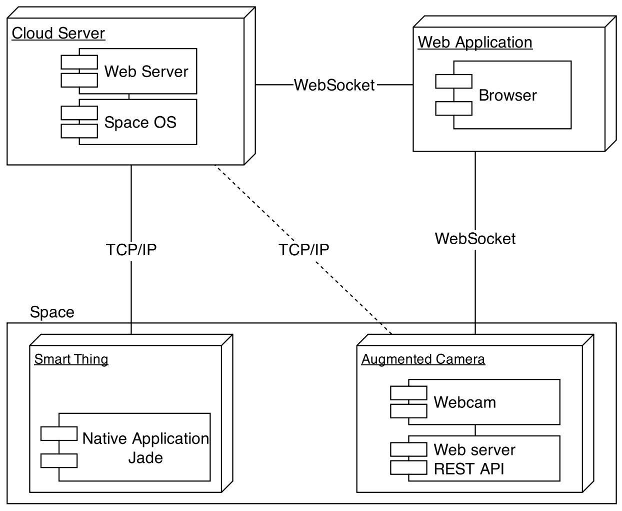

Fig. 4: Space OS components view

B. Components of Space OS

Fig. 4 shows a break down of the 3 main components of the SpaceOS framework, Spaces (Smart Environments), Web Application and the SpaceOS server.

1) Spaces: A Space is represented by the 'things' that it has and is identified by a unique id. The SpaceOS framework was designed to facilitate the process of adding objects to a smart environment. The act of bringing a smart object into a Space will trigger a chain of events responsible to registering that object with its current space and deploying necessary code to the SpaceOS server, see Fig. 5.

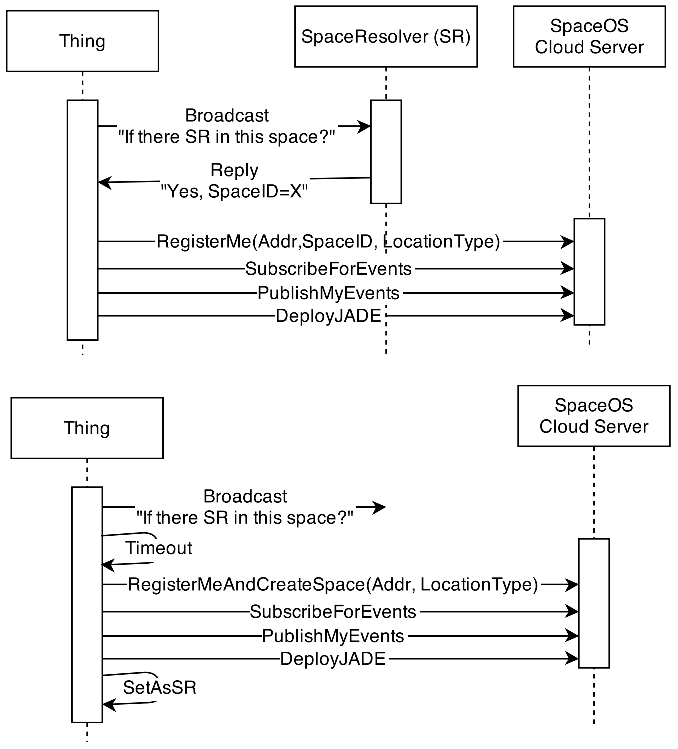

Fig. 5: Thing registration process

Each space has a Space Resolver. Space Resolvers assist the registration of a new object to a specific space in the network. For example, lets assume we want to make our house a smart environment. By bringing a smart object into the house it will connect to the local network and broadcast a message asking if there is any resolvers in the network. If yes, the resolver will reply by sending the unique id of their space. By receiving this unique id the new smart object can register itself with the SpaceOS server. In the case where a resolver is not found, the new smart object will register itself with the SpaceOS server and then becomes the resolver for the space. Once the registration is done the smart object and the SpaceOS server start the synchronization phase. In the synchronization phase the smart object will subscribe for events (see Section III-B3b) and publishes its metadata, live variables and server side code (see Section IV).

2) Web Application: The user browser is the interface at which the user will interact with the smart environment. It has two modes, the interaction mode and the programming mode. In the interaction mode the user is able to visualize smart objects in the environment, raise events from objects and interact with other users connected. In the programming mode users can create relationship between objects and events.

These relationships are used when smart objects notify the event manager that an event has happened. The act of building relationships between physical smart objects is what we call programming the physical world (see Section V).

3) Space OS server: The SpaceOS server is responsible for managing spaces and coordinating interactions between user-to-thing, thing-to-thing. It has 3 main modules which are explained below.

a) Space Management: The Space Management module is responsible for managing the logical representation of spaces, objects and users. Spaces, objects and users can be added, modified or deleted as necessary. It is also responsible for keeping track of which spaces and objects are still available (online). This is done by probing the smart objects and verifying that they are alive. If no object is available for a particular Space, that Space is considered offline, and is removed.

b) Event Management: Each smart object implements the observer pattern. Objects can subscribe for different types of notifications based on object type, content type or for an specific event raised by a specific object. Developers are able to expose their events at the synchronization phase (see Section III-B1). As events are raised, the event manager receives them through the network and is responsible for finding and notifying the subscribers with the assistance of the space manager. For example, if in our smart environment we have a switch that raises two events, SwithUp() and SwithDown(). A smart lamp can register for such events and at the same time it could specify a callback function, such as TurnOn() or TurnOff(). Therefore, relationships between events can be build between smart objects of the space.

c) View Management: The view management is responsible for handling all request and responses sent to the user browser. It communicates with the browser through web sockets. When requested, It will communicate with the space management to gather information about the smart objects available and their functionality. Its also responsible for forwarding event requests made by the user. For example, the user can request the lamp to TurnOn() through the web interface. The view management will receive that and forward that request to the Event manager for processing. Thus, all actions regarding user interface and user interactions are handled by the view manager.

IV. Jade Language

A. Objective of Jade design

There are several requirements while creating a new language to facilitate interactions in the world of Internet of Things. First challenge is to keep the language familiar. The new language should have minimal learning beyond existing languages and technologies such as JavaScript, C, C++, and HTML5. This way the language is most likely to be adopted by developers because the learning requirements will be low. Jade has been built on top of existing languages. Next, there is a need to simplify the programming model presented when developing for Internet of things. Instead of developing for potentially three different things, IoT, web, and cloud, we want to have a unified model. The unified model also presents a simpler security model.

Jade provides all of the above functionalities. The developer must write code in a Jade file (.ja extension) and then use the Jade compiler built in python to compile. The compiler creates two files, a C file for the 'thing', and a JavaScript file to allow the developer to insert part of the code into the nodeJS server, or the browser.

B. Functions

In Jade, 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 'jsdef', followed by a C prototype comprising of the function name and parameters. Inside the function, the code should be written in standard JavaScript syntax. This function is called from the 'thing' and runs in the cloud or the web.

```
jsdef C_function_prototype()
{
   // JavaScript body
}
```

Listing 1: jsdef function syntax

A simple jsdef function obeying the syntax is provided. Notice the C function declaration and JavaScript function body.

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

Listing 2: Example jsdef function

The JavaScript body in the jsdef function may require some external libraries, global variables or function calls which could be included inside jsrequire().

```
jsrequire()
{
    // external libraries, global variables,
        function calls ...
}
```

Listing 3: jsrequire syntax

In example below, the variable 'baz' is now globally scoped, which allows the jsdef function 'foo' in Listing 2 to access the value.

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

Listing 4: Example jsrequire

The function with keyword 'jscall' runs on the 'thing' and is callable from the cloud or the web.

```
jscall C_function()
{
    // jscall function calls to interact with other
        things through the exposed API
    // jsdef function calls to execute browser or
        server-side code
    // 'thing' application code ...
}
```

Listing 5: jscall function syntax

In the example jscall function below, 'foobar' calls the server-side jsdef function 'foo' in Listing 2.

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

Listing 6: Example jscall function

C. Live Variables

The browser, cloud, or 'things' are all capable of generating events. We need to have a method for updating the variable value, whenever theres a change in the state of the variable, possibly due to an event. To achieve this,we introduce the concept of 'live variables'.

1) Update triggered by the 'thing': In the application, the developer could indicate which variable could change, by using a 'live' keyword before the variables declaration, for example in Listing 6. 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.

On the JavaScript side, to create reference to a live variable in the 'thing', the developer needs to call the liveVar() function, which is defined inside a JavaScript file running on the browser.

```
    var x = liveVar( foobar.x  ,value);
```

The live variable creation should be inside the JavaScript body of the jsdef function. The first argument (e.g. 'foobar.x' above) of a liveVar() call needs to be a string containing the live variables scope and name to correctly refer to the live variable in the C code. The second argument bears the value of the live variable. The developer could add the event that should transpire when a live variable value update is received from the C side, using the syntax shown below.

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

The onupdate function allows the developer to insert any JavaScript operations; and it is invoked whenever the value of the live variable in the 'thing' gets modified. In the example function 'foo' in Listing 2, the onupdate function creates an alert message on the screen, whenever a new value of the live variable is received.

2) Update triggered by the web / cloud: In the jscall function, we set a timeout that checks for a difference in the value. Only if a new value is received, the C function updates it. Upon update in the variables value, the C function in the 'thing' is executed with this new value.

V. Example Applications

SpaceOS is suitable for hosting a wide array of applications that are already touted as candidates for the emerging paradigm of smart computing environments. Because SpaceOS supports a hybrid model that overlays the control and monitoring of smart objects on a video stream of the physical space, more application scenarios are made possible.

Home Automation: Smart computing is making steady inroads into home automation, where appliances and other items are getting an infusion of computing capabilities. The next step is for the smart objects to operate in concert to support a smarter environment in the home. Interoperability is one of the major challenges in achieving this objective. SpaceOS provides an enabling framework for a smart computing environment by providing a user-driven approach for tackling the interoperability problem. In addition, SpaceOS can also enable newer modes of interactions. In particular, users can employ video to obtain another view of the home environment while controlling the smart devices and interacting with the people in the home.

Remote Monitoring and Control for Elderly Care: The basic setup of this application is very similar to home automation. However, this problem can have stronger privacy issues because of a third party involvement. The hybrid model of overlaying the video stream with smart device information can help. We can decrypt portions of the video that are normally encrypted using keys provided by the smart devices. Depending on the contingency that might arise, the video becomes visible for the care provider.

Realistic Online Showrooms: SpaceOS is ideal for setting up interactive web-based virtual showrooms for advertising campaigns. Virtual showrooms could feature electronic gadgets such as televisions and computers. In interactive showrooms, users can observe the working of a gadget externally and interact with it in a selective manner. For example they could turn on/off selected features of a gadget and observe its behaviour. Even interconnecting smart gadgets to check their inter-networking capabilities could be part of the services offered by the virtual showrooms.

Physical Devices and Spaces as a Service: There are many valuable physical spaces and devices for which users have limited access. One example would be science laboratories for high school students. A smart computing environment like the one created by SpaceOS can be used to virtualize a laboratory and provide remote access to the facility to a large number of students. Further, using the facilities provided by SpaceOS it is possible for large number of users to collaborate in a physical space.

VI. Discussion and Related Work

SpaceOS emerges as a pragmatic approach to attempt to address some of the existing challenges in the areas of ubiquitous, physical computing, and ambient intelligence by leveraging the existing advancements in embedded systems and adapting to evolving technology trends in smart computing.

Analogous to SPREAD[1], in abstraction of resource representation, physical entity only exposes relevant information, if available, to the system through the independently SpaceOS API. Contrast to SPREAD, where a tuple-space representation, from LINDA[2], is used for each object, SpaceOS can link attributes, and state information associatively to better represent the object in logical space. In the ubiquitous computing model suggested by Plan B[3], instead of having a middleware, all properties of an object are exported to the central file system.

All machines are peers and export resources as a file system, with an associated name, and a set of attributes. SpaceOS also adopts this concept and allow 'things' to send their resources and attributes to the cloud.

In the vision of Cyber-Physical Systems[4], SpaceOS can participate by serving as the central communication core. By providing controls over the interconnected smart devices, SpaceOS enables telepresence by allowing safe, secure, and reliable manipulation of devices in the telepresence space. The challenges related to privacy and trust are dampened by the described unified model.

Spatial Programming (SP)[5], tries to implement a programming model for Networks of Embedded Systems, to coordinate actions in a decentralized manner. In contrast, SpaceOS has centralized components, for instance, the cloud, which enables better interactions between devices. In SP, space is divided into 'space1', which is a given geographic location and 'space2', which is another location detected dynamically once a motion sensor is triggered by an object. SpaceOS builds on this concept of dynamically detecting changes in state of a 'thing', by using the concept of 'Live' variables in the Jade language.

Creating software for Real Time Embedded Systems[6] to interact with physical devices poses some hard challenges, as the system at the time of writing the paper did not support the development of reliable and robust embedded systems. However, with the evolution of microprocessors, coupled with the ability to control them using SpaceOS, overcoming these challenges seem more realistic.

The Jade language, introduced in SpaceOS, allows developers to have first hand hardware interaction with smart devices, through the C component, while providing web based manipulations, through the Javascript component. Delivered in the unified model, Jade overcomes possible complications introduced by the requirement of CORBAs Object Request Broker to have an Interface Definition Language. In contrast to Remote Procedure Calls[7], Jade provides a lightweight execution environment which is suitable for smart devices, having the advantage of bidirectional communication, by accomplishing remoting over web sockets, as to REST[8].

The Semantic Oriented vision[9], involves 'things' interacting over the "internet". Semantic technologies are a means of creating machine interpretable representation which provides an efficient way of sharing and integrating information. SpaceOS shares the same notion of abstraction of different technologies as the paper, by using the cloud as a middleware between the 'things' and the browser. Interoperability, viewed as a key topic in Semantic technology, is resolved in SpaceOS by making the API of the 'things' accessible to the cloud. SpaceOS, through C programs running on the 'things', demonstrates lightweight ontology as noted by the author to provide a better likelihood of wide scale adoption, also addresses the outlined issue that most semantic tools/techniques are created mainly for web resources and often overlook the constraints of the IoT resources.

VII. Conclusions and Future Work

Internet is poised to undergo a major transformation due to the induction of a massive number of smart connected devices into its fold. There exists a huge potential for creating smart computing environments that interact with the physical world by leveraging the mass of interconnected devices. However, to tap the full potential we need application development frameworks that are easy to use for the developers. SpaceOS aims to do exactly that.

The SpaceOS offers many services to the developer. At the core it ties the three major types of elements in a smart computing environment: thing, cloud, and web. On top of the core, it provides foundational services that are element specific and enhances an element such that a developer can use the element in an application with least difficulty. At the top of the stack is the application services that are created by the developer using the facilities provided by the Jade language.

This paper presented the SpaceOS architecture with the details of the key components of the architecture. The important features of the Jade language were described. Some important applications that can leverage the features of SpaceOS are examined. In particular, SpaceOS enables a hybrid model that overlays the control and monitoring services for the smart devices on a video stream of the physical space. Using this hybrid model, SpaceOS can enable a new genre of applications such as virtualizing physical devices and spaces.

We are working on a proof-of-concept prototype of SpaceOS and a preprocessor and runtime for the Jade language. Many features of the Jade language have been implemented using the preprocessor and the runtime over the rudimentary SpaceOS framework. We are working on getting a full-fledged proof-of-concept prototype completed with an accompanying application such as smarter variation of telepresence.

References

[1] P. Couderc and M. Banatre, "Ambient computing applications: an experience with the spread approach," in *36th Hawaii Int'l Conf. on System Sciences*, Jan 2003, p. 9 pp.

[2] D. Gelernter, "Generative communication in linda," *ACM Trans. Program. Lang. Syst.*, vol. 7, no. 1, pp. 80–112, Jan. 1985.

[3] F. Ballesteros, E. Soriano, K. Leal, and G. Guardiola, "Plan B: An operating system for ubiquitous computing environments," in *4th IEEE Int'l Conf. Pervasive Computing and Communications*, Mar. 2006.

[4] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: The next computing revolution," in *47th ACM/IEEE Design Automation Conf.*, June 2010, pp. 731–736.

[5] J. Pauty, P. Couderc, and M. Banâtre, "Spatial programming: Using the physical world as a computing system," in *UbiPhysics workshop/Ubicomp*, 2005.

[6] J. A. Stankovic, I. Lee, A. Mok, and R. Rajkumar, "Opportunities and obligations for physical computing systems," *Computer*, vol. 38, no. 11, pp. 23–31, Nov. 2005.

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

[8] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, 2000.

[9] P. Balamuralidhara, P. Misra, and A. Pal, "Software platforms for internet of things and M2M," *Journal of the Indian Institute of Science*, vol. 93, no. 3, pp. 487–498, 2013.