Cross Domain Communication in the Web of Things A New Context for the Old Problem

Nam Giang, Minkeun Ha, Daeyoung Kim
Korea Advanced Institute of Science and Technology
924, N1 Building, KAIST
Daejeon, Korea
{zang, minkeun.ha, kimd}@kaist.ac.kr

ABSTRACT

Cross domain communication has been a long-discussed subject in the field of web-based application, especially for any sort of mashups where a single web app combines resources from different locations. This issue becomes more important in the Web of Things context, where every physical resources are exposed to the Web and mashed up by other web applications. In this paper we demonstrate a use case in which cross domain communication is applied in the Web of Things using the HTML5 Cross Document Messaging API (HTML5CDM). In addition, we contribute an advanced implementation of HTML5CDM that brings RESTful communication model to HTML5CDM and supports better concurrent message exchange, which we believe will be of much benefit to web developers. In addition, a time/space evaluation that measures CPU and Memory usage for the developed HTML5CDM library is carried out and the results has proved our implementation's practicability.

Categories and Subject Descriptors

H.5.3 [**Group and Organization Interfaces**]: Web-based interaction—*Web*

Keywords

Cross Domain Communication; HTML5 Cross Document Messaging; Web Apps; Mashups; Web of Things

1. BACKGROUND

Web-based applications (web apps) with web standard languages such as HTML and JavaScript are becoming a mainstream apps development beside other native programming languages. One of the best advantages that web apps offer is the ability to exploit client-side computing capability so that reduce any possible delay due to network overhead and servers' availability. In the Web of Things context, it is also preferable that the communication between users' terminal (i.e, web browsers) and physical objects should be low delay and more direct to ensure users' experiences. Thus web apps seem to be the most suitable mean for interacting with physical objects in the Web of Things.

However, web apps that mashup physical resources are suffered from Same Origin Policy (SOP), which prevents them to access physical resources that are located at different origins. For example, a mashup web application at `http://mythings.com` will have difficulties in accessing resources from `http://myroomlamp.com` or any IPv6 address, which is the physical object's address.

There are several solutions for the SOP to date, including 1) Cross Origin Resource Sharing [6], 2) JSON with Padding [5], 3) some old versions of web browsers provide a dialog to ask users for allowing cross domain requests and 4) HTML5 Cross Document Messaging API (HTML5CDM) [3]. The first solution tries to cooperate between web servers that want to access resources hosted by each other. To achieve this, a header *Access-control-allow-origin* is added to the response so that web browser can accept cross domain requests. For second solution, JSON with Padding is a temporary work-around that uses the script tag dynamically to make cross domain requests instead of using Ajax/XMLHttpRequest. The third solution creates a dialog that asks for user's permission to allow such restriction. Lastly, the HTML5CDM accomplishes the work by defining a standard set of rules to send and verify the exchanged messages' authenticity.

The HTML5CDM seems to be the most promising option, especially in the Web of Things context. This is because, 1) It is part of HTML5 and is going to be standardized, meaning officially accepted and widely spread, thus it ensures interoperability. Some temporary solutions can work at a time but are usually not favorable in long term. 2) It does not need to cooperate between embedded web servers and the web apps, which should be maximally avoided in embedded systems. 3) It does not require user's permission to perform cross-origins requests, which results in better user experiences. Nonetheless, it is relatively immature for two reasons. First, there is no limit for which the exchanged messages can be, they can range from the simplest form as a character sequence to a more sophisticated form such as a JSON object. Second, there is a lack of support for concurrent message exchange from different threads so that they need to wait for their turn to send the messages. Moreover, message responses are not automatically dispatched to the right destination. Thus, in order to exploit HTML5 CDM efficiently in web apps, an extra abstraction layer for HTML5CDM could be of benefit.

WWW'14 Companion, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2745-9/14/04.
http://dx.doi.org/10.1145/2567948.2577032.

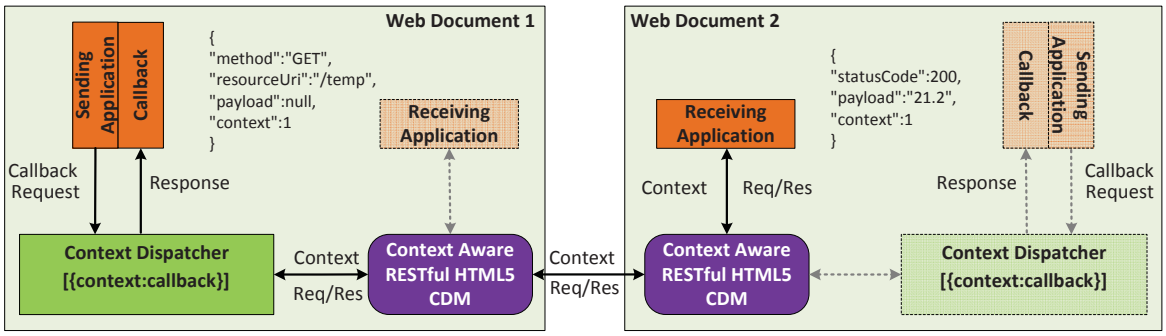

Figure 2: Context-aware RESTful HTML5CDM

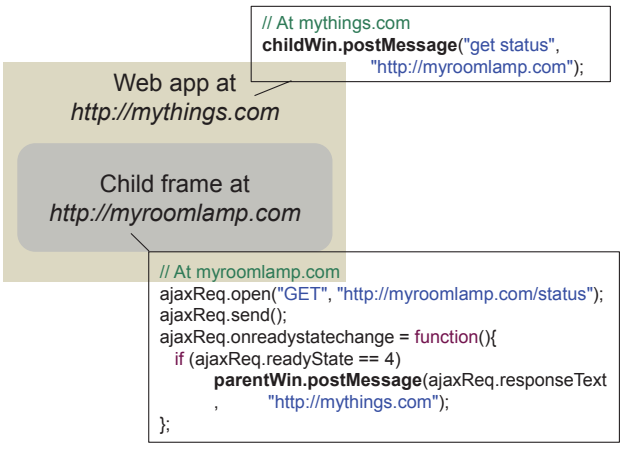

Figure 1: HTML5 Cross Document Messaging API

To this end, we propose an advanced implementation of HTML5CDM based on RESTful communication model [2]. The implementation helps web developers to easily get use of the HTML5CDM and supports concurrent message exchanges from different browser threads. The advanced implementation leverage the RESTful communication model in web services to abstract out the HTML5CDM communication. Thus, exchanging messages are divided into requests and responses with appropriate query verbs and status codes. By RESTifying the HTML5CDM, our advanced implementation features the transparent transition from HTML5CDM to Ajax so that the SOP is seamlessly overcome. Moreover, the advanced HTML5CDM library uses a context identifier to match responses with appropriate request originators so that multiple threads can participate in sending and receiving messages simultaneously. We demonstrate the application of cross domain communication, particularly the application of HTML5CDM in the Web of Things context using our proposed implementation.

The paper is organized as follows. Section 2 gives the discussion on cross domain communication in the Web of Things context and details of the advanced implementation of HTML5CDM in JavaScript. In section 3, we show the demonstration of the Web of Things web app and section 4 shows our evaluation results for the proposed solution. Lastly, section 5 concludes our paper.

2. CROSS DOMAIN COMMUNICATION IN THE WEB OF THINGS

2.1 The use case

Fig. 1 shows a general use case of cross domain communication and particularly the use of HTML5CDM in the Web of Things context. In this figure, the main web app from `http://mythings.com` wants to access a resource which is the lamp *status* from another location, `http://myroomlamp.com`. Due to different origins, the web app's client script cannot access the resource directly while a client script at the page `http://myroomlamp.com` is free to do so.

HTML5CDM suggests that the site `http://myroomlamp.com` could be embedded into the `http://mythings.com` web app as an inline frame to act as a proxy so that using HTML5CDM technique, the web app at `http://mythings.com` can access the *status* resource located at `http://myroomlamp.com`. In detail, the parent web app posts a HTML5CDM message to its child frame indicating that it wants to get the *status* resource. Sequentially, the child page retrieves the resource using Ajax communication and posts it back to the parent page.

This original version of HTML5CDM has two limitations as introduced in section 1. First, it lacks a message abstraction layer so that message format is defined by the individual web app developers. This poses difficulties for different web apps to interoperate properly with each other. Second, there is only one event listener for every exchanged message so that different threads need to wait until their turn to communicate. This is very troublesome for heavy event-driven programming languages like JavaScript.

2.2 RESTifying the HTML5CDM

To realize a standard message format, HTML5CDM messages are abstracted out following the RESTful communication model. We add some extra information to the exchanged message so that they are not just simple character sequences but a JSON object. To make the transition to Ajax communication transparent, the exchanged messages are divided into requests and responses. Similarly to Ajax, HTML5CDM requests consist of a method field (e.g, GET, PUT, POST, DELETE), a resource URI (e.g, /status) and an optional payload for any "write" requests. HTML5CDM responses include a status code (e.g, 200, 404, 403) and a payload for the actual response. This standard RESTful communication model helps ease the parsing tasks that are required to understanding requests and responses between

web apps. Thus, one web app can define the resources or services it wants to expose to other web apps and they will be consumed under the RESTful communication model.

Additionally, to support the concurrent message exchanges between different threads, a context identifier is added in both request and response in order to pair them together. This context identifier is supposed to be maintained in both sender and receiver and it is uniquely created for every thread/transaction so that responses can be correctly dispatched to the right origination. Thus, each time a thread wants to send a message to another web page, it defines a callback function that is used to process the response. Then the callback will be registered to the library and a unique context identifier will be generated for the transaction. A dictionary will be maintained to match the identifier with corresponding callback function so that once a response is retrieved, it is dispatched to the correct processor. Due to the use of context identifier and the RESTful messaging model, the developed library is also called context-aware RESTful HTML5CDM.

Fig. 2 fully illustrates the process. In this figure, there are two web documents are exchanging information with each other. They can be a normal web page and its child iframe or two iframes of one page. These two web documents are originated from different domains so that they cannot access each other's information directly. The sending application of the web document 1 sends a HTML5 CDM message to the receiving application of the web document 2. The context identifier-to-callback dictionary is maintained in a component called dispatcher. Whenever the dispatcher receives a request and a corresponding callback from a sending application, it generates a unique context identifier and stores it along with the callback. The dispatcher then attaches the context identifier in the HTML5CDM message and sends it out. The receiving application maintains that context identifier and send it back with the response. Using this context identifier, the sender's context dispatcher can dispatch the response to the appropriate callback of the sending application.

2.3 Application

This section details the application of the developed library. In order to use the library, web pages who want to communicate with each other using HTML5CDM will embed the library on their *<head>* element. A simple script element like *<script src="Ahtml5cdm.js"></script>* will simply do the task. Once the library is referenced, both web pages can begin with register their accepted peer origin to make sure only messages that are originated from the specified peer will be processed. The registration looks like the following code snippet:

```
Xdm5.registerPeer("http://foo.bar");
Xdm5.registerPeer("^https*://.+\.kaist\.ac\.kr$");
```

It is worth noting that the library also supports pattern matching for origin registration as shown in the second line. This registration process puts the registered origins into a whitelist so that the web app can accept them in later transaction. After registering the interested peer origin, the web app can proceed with registering all available resources that will be exposed. The process is illustrated as follow:

Figure 3: IP-WSN Testbed for the Demonstration

Figure 4: A web-based mashup application

```
Xdm5.registerResource("/test", testHandler);
function testHandler(event){
    var response =
        Xdm5.creatResponse(event, 200, "Some response");
    Xdm5.sendCdmMsg(event, response);
}
```

As seen in the code snippet, the process is very easy in development and thus, free web developers from the underlying complexity of handling HTML5CDM messages. The response is created using the library with the three input parameters. The first parameter is the event that was received from peer origin, this event is used to maintain the context of the transaction. The second and the third parameters are the status code and the response message, which are designed according to the RESTful messaging model. This response is then sent back to the sender using the received *event*.

3. DEMONSTRATION

3.1 The Demonstration Testbed

The demonstration testbed is shown in Fig. 3, which includes an Apps server that host the mashup web app and

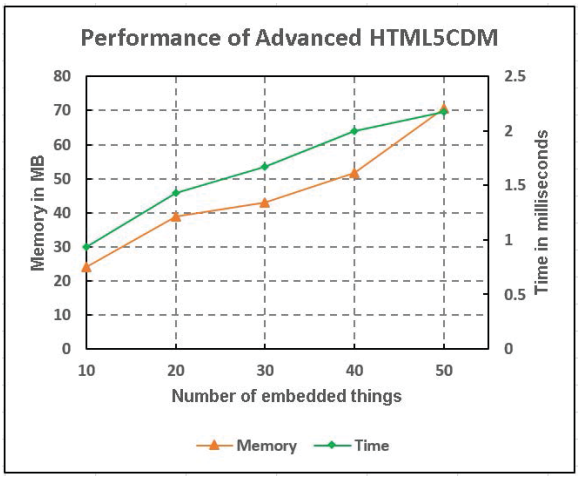

Figure 5: Time and memory analysis of Advanced HTML5CDM

an IP-based Wireless Sensor Network (IP-WSN) platform. Our IP-WSN platform called SNAIL [4] is used to represent physical objects in the Web of Things. SNAIL gateway is a 6LoWPAN gateway that brings Internet connection to every sensor, actuator mote. These sensor, actuator motes are equipped with embedded web servers and assigned globally unique IPv6 addresses so that they can expose their resources to the Web and can be accessed directly from web browsers. These motes are then attached onto everyday physical objects such as a flower pot or any heater to capture their contextual information that will eventually be exposed to the Web.

3.2 The Demonstration Scenario

The demonstrated scenario is about a web-based mashup application that collects data from a vast number of sensor web services from the SNAIL platform. Each sensor has its own web app that allows users to access its physical resources directly on the web browsers [1]. In addition, their web apps also support the advanced HTML5CDM library so that their resources can be exposed to the mashup web app. Thus, by embedding the individual objects' default page as inline frames, the mashup application can access its data regardless of the differences in their origins.

The snapshot of the demonstrated web app is shown in Fig. 4. From this web app interface, users can chose to add their own things' addresses, either IP addresses or domain names. Then the mashup application will automatically create an inline frame of zero size that points to the thing's web site. Through this frame, the mashup application can access the thing's data and iterate through all available resources to construct another thing's section on the web interface. Therefore, end users will have a dashboard on which they can directly interact with their physical things without having to go through any intermediate backend server.

4. EVALUATIONS

We evaluate the developed library using the Chrome browser's built-in CPU and Memory Profiler. The evaluation takes into account the number of concurrent exchanged messages between two web apps and evaluate how CPU and Memory usages are affected. The maximum number of embedded frames tested is 50 and result is shown in Fig. 5. The experiment is done ten times and average value is reported. From this figure, it is seen that the time and memory consumption grow when the number of embedded frames increases, which is obvious. However, the reported memory and time usage shows that the developed library and the proposed communication model is practical since only 70MB of memory is required for the worst case and all requests are completed within 2.25ms.

5. CONCLUSIONS

The demo abstract discusses the application of cross domain communication in the Web of Things context. We introduce a practical usage of HTML5 Cross Document Messaging API for the cross domain communication issue. While the original version of HTML5CDM has some limitations in terms of messaging model and concurrency, the advanced implementation has shown significant enhancements. The messaging model is carefully developed that follows the RESTful message formats. This message standard helps reduce the effort in parsing messages between different web apps and boosts the interoperability. Besides, the advanced implementation enables the concurrency in message exchanges so that multiple threads can send and receive messages simultaneously. Finally, the demonstrated web-based mashup application and necessary evaluation has proved the feasibility of the work.

6. ACKNOWLEDGMENTS

This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the CITRC (Convergence Information Technology Research Center) support program (NIPA-2013-H0401-13-2008) supervised by the NIPA(National IT Industry Promotion Agency), the IT R&D program of MSIP/KEIT (10041313, UX-oriented Mobile SW Platform) and the International Research & Development Program of the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT&Future Planning of Korea(2012-0008824).

7. REFERENCES

[1] S. Bae, D. Kim, M. Ha, and S. H. Kim. Browsing architecture with presentation metadata for the internet of things. In *IEEE International Conference on Parallel and Distributed Systems*, 2011.

[2] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. University of California, Irvine, 2000.

[3] I. Hickson. Html5 web messaging. w3c working draft. Online. http://dev.w3.org/html5/postmsg/.

[4] S. Hong, D. Kim, M. Ha, S. Bae, S. J. Park, W. Jung, and J.-E. Kim. Snail: An ip-based wireless sensor network approach toward the internet of things. *IEEE Wireless Communications*, 17(6):34–42, 2010.

[5] B. Ippolito. Json with padding. Online. http://bob.ippoli.to/archives/2005/12/05/remote-json-jsonp/.

[6] A. van Kesteren. Cross-origin resource sharing. w3c working draft. Online. http://www.w3.org/TR/cors/.