Application Synchronisation on Public Displays based on PubSubHubbub

Manuel Pereira, Maria João Nicolau and Helena Rodrigues

Centro Algoritmi, Escola de Engenharia, Universidade do Minho, Campus de Azurém, 4800-058 Guimarães, Portugal
a50022@alunos.uminho.pt,{joao,helena}@dsi.uminho.pt

Abstract. Large-scale pervasive public displays networks are becoming an emerging paradigm and represent a radical transformation in the way we think about information dissemination in public spaces. One of the features of pervasive public display systems is their ability to create experiences that span across multiple displays in a coordinated fashion. Proprietary single site display solutions exist but these are not open to third-party developers.
On the other hand, scalable open systems that enable large-scale, synchronised and multi-screen experiences, spanning multiple networks domains will call for the definition of multiple administrative boundaries that accommodate function partitioning. In our research, we are studying the key requirements involved in this open application synchronisation and present our initial work on designing a synchronisation model and Application Programming Interface for public displays application developers that is built on top of the *PubSubHubbub* protocol, an open protocol for distributed publish/subscribe communication on the Internet.

Keywords: Public displays, synchronisation, publish-subscriber, PubSubHubbub

1 Introduction

Open, large-scale pervasive public displays networks are becoming an emerging paradigm and represent a radical transformation in the way we think about information dissemination in public spaces. In particular, open networks of public displays create the opportunity for third parties to create and publish content in the form of applications, promoting openness as a source of value for all the parties involved [7, 14]. In the future we believe that the interconnected nature of these displays will mean that new applications for public displays will promote not only enticing situated interactions but also meaningful synchronised interactions between independent, possibly remote, public displays installations, challenging radically the current stat-of-the-art in digital signage.

Application synchronisation on public displays networks has been mainly approached from the perspective of application scheduling. Storz et al [13] has developed, as part of the e-Campus project, a domain-independent API that supports the construction of domain-specific scheduling approaches. Using this approach, it is possible to support a combination of both statically scheduled content and interactive schedule content across multiple displays. A more recent approach for content scheduling in e-Campus has been described in [2]. The scheduling API has been replaced by the concept of Content Descriptor Set (CDS) that describes how single items of media should be rendered at the display. The system is opened to content from numerous sources with no centralised control. Although not specifically analysed in the paper, coordination between displays seems to be supported in the definition of CDSs.

Application synchronisation has also been investigated in the context of system support for smart spaces. The *Event Heap* is one of the main components of the iROS operating system for smart spaces. Users in a meeting room interact with different visualization applications that run on displays and other situated devices, such as on laptops or PDAs, and are able to control any device or application from their current location. The *Event Heap* [10], a publish-subscribe system, provides for dynamic application coordination and forms the underlying communication infrastructure for applications in the interactive workspace.

Our approach to application synchronisation on public display networks aims at evaluating a publish-subscriber protocol designed for the web space, the PubSubHubbub[1], as an architectural component of a public display network system with the objective to promote synchronised experiences across displays.

In section 2 we describe the main architectural concerns that support the execution of applications on open public display networks. In section 3 we describe a synchronisation scenario in public display networks and define the initial requirements for public displays synchronisation. In section 4 we introduce the PubSubHubbub protocol, describe a PHP API for application development and briefly describe our approach through the definition of a simple application. Finally, in section 5 we present our conclusions.

2 Public Displays Applications

Figure 1 presents the main functional blocks and interactions required to have a fully functional pervasive displays network. It can be instantiated in different forms and replicated as needed, depending on grow demands [5]. We will briefly describe the depicted architecture, emphasising the main architectural aspects that frame our work.

The *Application Developer* is the main actor in the *Application Domain*. He is responsible for application development, application description and application registration. The application model for public displays networks is still an open issue. In this work we share the application model provided by the Instant Places, a specific display infrastructure. Instant Places aggregates place-based screen media and explores new concepts for user-generated content [11]. A display application in Instant Places is a web application whose primary goal is to

[1] https://pubsubhubbub.appspot.com/

render content on a public display [14]. Display apps are based on web technologies and standards, e.g., HTML, JavaScript and CSS. They are developed on public servers from where they can be used in any public display. Displays apps are rendered in any standard web browser or other types of specially tailored web stacks and their model is optimised for the distinctive execution context and user experience of public displays. The InstantPlaces system facilitates seamless integration of third party web applications residing anywhere in the public Internet into a display, thus catering for a scalable and open architecture. Instant Places offers a model for content presentation that takes into consideration both the display environmental data, e.g., sensors and user interactions, and app specific configuration. This approach enables the content being shown to be highly personalized, thus reflecting the dynamic and situated behavior of displays global web apps [6].

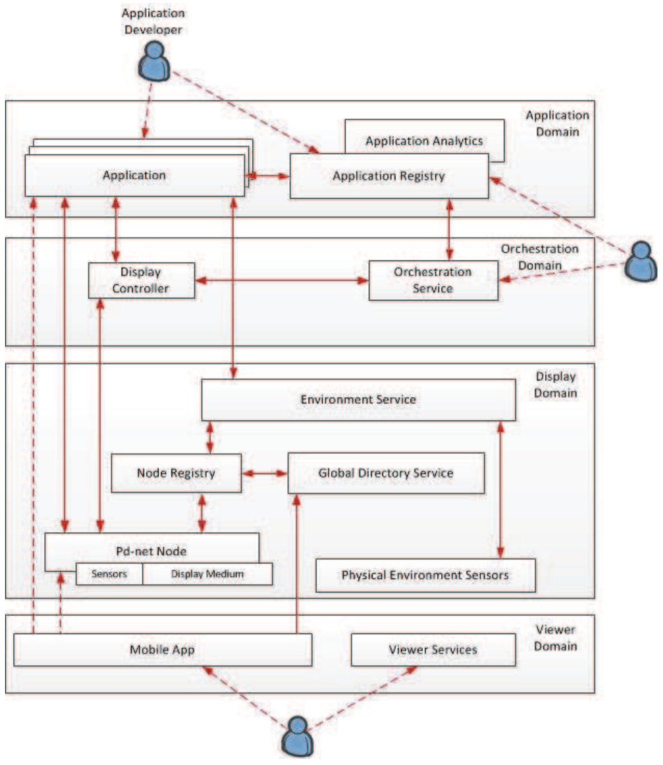


Fig. 1. Public Display Network architecture.

Application registration in the *Application Registry* is the basis for an application distribution model. Making an analogy with the mobile phone context, Clinch et al. present a set of design considerations for public displays applications stores [3]. Application stores for public displays have the potencial to promote collaboration and synchronisation in public spaces as innovative ideas are likely to appear in open innovation environments. Additionally, sophisticated application description systems that are likely to appear, will also have the potential for promoting application sharing and synchronisation as developers and displays owners will be offered shared environments for application behaviour description.

The *Display owner* accesses to one or more application stores, registers his display and installs applications on them. He interacts with the *Orchestration Service* to define how applications are rendered in the display. Application developers may also be able to define orchestration constraints and, ultimately, *User* may also influence display behaviour through situated interactions. Orchestration information are the basis for the *Display Controllers* to decide which content or application should be rendered, where in the display and when - scheduling model. Typical scheduling models for traditional public displays most commonly multiplex over a set of content items through time multiplexing, regularly changing the item visible on the display for another. Open public displays networks should accommodate new forms of highly dynamic scheduling that emerge from the dynamic nature of content itself. In particular, for the problematic of display application synchronisation, the need for dynamic scheduling may be associated with the need to react to application events running on a different display.

A final issue on public display networks systems is the question of where to physically locate the application the display is executing. The potentially large-scale characteristics of public displays networks with their inherent innovative nature, characterized by a continuously changing number of users, display owners, content producers, display nodes, application items, application hosts, content types, interaction modalities, sensors and connections, may lead to several scalability and performance problems. In this context, existing scalability techniques can be applied, in particular in the web environment. This may call for application replication techniques or for dynamic VM synthesis supported by cloudlets [4].

3 Synchronization scenario and requirements

To illustrate the synchronisation aspect of public displays applications, consider the following scenario: *International Action - World AIDS Day - It is the 1st of December - World AIDS Day. All around Europe there are initiatives exploring what AIDS means to different communities: including public information on how to avoid contracting AIDS; documentaries concerning the plight of many AIDS sufferers in Africa; and content from grass roots groups such as schools fund raising for AIDS charities. Citizens in Europe are united through common interactive experiences created by a team of international artists and displayed on every participating display in Europe.*

In this situation, displays are being used for improving communication between different communities and promoting awareness to a world-wide health problem. Synchronisation requirements are evident as different and, preferably, independent public displays installations need to coordinate their actions. Consider for example the situation where a photo of a greeting gesture of a group of people in front of a display may immediately be spread for all participating displays. Every participating installation has previously looked up and subscribed the "World AIDS Day application" (or one of the "World AIDS Day applications", if a few variations of the application may exist). The subscription process is out-of-scope of this document, but at the end every installation had been configured with the application URL (a public display opmtimised end-point) and a proper set of scheduling constraints would have been set by the respective installation's managers. Those would include, at the minimum, a rule for scheduling the application on the 1st of December and some indication about the event-based nature of the application. Situated-behaviour and interaction with local resources is accomplished through the integration of the environment service provided by the display node (possibly an Instant Places node). Applications are running on public servers, possibly replicated in different servers or running on local VM supported by cloudlets [4].

Public displays networks will be sharing the most important characteristics of large-scale distributed systems: they will be decentralised in many forms, developed, owned and used by a wide variety of stakeholders with conflicting needs and expanding or evolving continuously. One of the key aspect of this situation is the autonomy between the public displays installations. Every installation is likely to be managed by independent owners and do not share any functional component of the public display network system apart from the application store and, in some cases, the public server running the web application. Although some form of a scheduler component capable of managing a set of displays installations could also fulfill the above requirements [13], we believe that such an approach would "violate" the autonomy of public display installations. In our research we intend to approach the problem of synchronisation in public places by offering a web based, loosely-coupled synchronisation framework for public displays applications that allows for public displays synchronisation while preserving the autonomy of display installations. In this paper, we report our initial work on exploring the PubSubHubbub, a publish-subscribe protocol for the web space. In the next section we briefly introduce the protocol and describe an API for application synchronisation on public displays.

4 Coordination within PubSubHubbub

Publish-subscriber systems are known as having a particular relevance in communication models in the increasingly distributed and necessarily loosely coupled context of the modern Internet. They offer loosening coupling in space, time or synchronisation [8]. Moreover, publish-subscriber systems have been an important component in the web context, recently enforced by the increasing popularity of the "Real Time Web". In particular, approaches such as PuSH

(PubSubHubbub) [9] add realtime notifications on top of RSS provide a RESTful decentralised publish-subscribe service.

4.1 PubSubHubbub

PubSubHubbub is *a simple, open, server-to-server web-hook-based pubsub (publish/subscribe) protocol as an extension to Atom and RSS*[2]. PubSubHubbub is a topic-based subscription protocol, not a service. Application developers can run a PubSubHubbub or use an open server. Google is running, in scalable clouds, an open test server for anybody to use to help bootstrap the protocol[3].

The main architectural entities of the PubSubHubbub protocol are the "Hub", the "Publishers" and the "Subscribers". An Hub is the server which implements both sides of this protocol. It acts as a broker between publishers and subscribers. A feed publisher indicates in the feed document (Atom [12] or RSS [1]) its hub URL, to which a subscriber (a web server) can register the callback URL. The publisher notifies its hub whenever it generates a feed update. The hub than fetches the feed and sends it to all the registered subscribers. Subscribers provide endpoint URLs to which hubs can post updates. For a complete description of the PubSubHubbub protocol, please consult [9].

4.2 Synchronisation PHP API

We have design a basic PHP API for using the PubSubHubbub protocol. Our objective is to provide a very first application programming tool for providing synchronisation in public display networks. This will be the basis for the implementation of a few synchronisation scenarios in public displays networks. We intend to study performance issues as well as to define the requirements for a programming tool for developing synchronised applications for an open public display network. As we expect that, as display nodes, PubSuBHubbub will exhibit a federated behaviour, hud discovery and hub coordination is expected to be a crucial issue.

In our API, an *event* represents the information that is shared between publishers and subscribers. It has a type, a description, a creation time and a list of optional parameters, specific to the application domain. The API consists of four operations: *create_event()*, *publish_event()*, *subscribe_event()* and *read_event()*.

function create_event($parameters) This function receives an array that contains the parameters of an event. Each parameter is described by a pair ("key","value"). This function adds a xml representation of the event to the respective application feed. It returns true or false if success or failure respectively (HTTP error code is logged for analysis).

function publish_event($topic_urls,$hub_url) This function publishes an event in the hub with address $hub_url. The topic URL corresponds to the

[2] https://pubsubhubbub.appspot.com/

[3] https://code.google.com/p/pubsubhubbub/

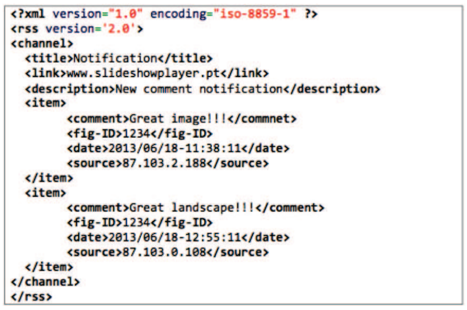
address of the application feed. It returns true or false if success or failure respectively (HTTP error code is logged for analysis).

function subscribe_event($hub_url,$callback_url,$topic_url,$lease_seconds) This function subscribes an event the application is interested in receiving. $hub_url defines the hub address, $callback defines the local endpoint URL that receives the hub notification when the event occurs. $topic_url corresponds to the address of the feed the application intends to subscribe for notifications. $lease_seconds defines the life time of the subscription (in seconds). The hub has to validate the subscriber application callback endpoint. To do that, an hub contacts the callback endpoint to test if it is alive. The application must to answer back to that contact. The API contains a class ("endpoint.php") that offers the mechanisms to implement this handshake.

function read_event($parameters) This function extracts from a notification the key values that are described in $parameters.

Example

We are developing a simple application that shows a photo slideshow, the description of the current photo and a set of user submitted comments about the photos. Additionally, the application provides two endpoints for submitting new photos and comments. The application is associated to a set of public displays and the intended behaviour is to provide a shared photo slideshow in every place. Whenever a new photo is uploaded, the new photo is displayed in every public display. Whenever a new comment is published, the list of comments is updated in every application.



```
<?xml version="1.0" encoding="iso-8859-1" ?>
<rss version='2.0'>
<channel>
  <title>Notification</title>
  <link>www.slideshowplayer.pt</link>
  <description>New comment notification</description>
  <item>
        <comment>Great image!!!</commnet>
        <fig-ID>1234</fig-ID>
        <date>2013/06/18-11:38:11</date>
        <source>87.103.2.188</source>
  </item>
  <item>
        <comment>Great landscape!!!</comment>
        <fig-ID>1234</fig-ID>
        <date>2013/06/18-12:55:11</date>
        <source>87.103.0.108</source>
  </item>
</channel>
</rss>
```

Fig. 2. Application feed containing two new comment events.

In our approach, different application components and/or different application components instances correspond to publishers and subscribers. Submission of a new photo or a new comment corresponds to application events which are published and subscribed accordingly. As an example, consider the figure 2 that describes a feed formed by two new comment events.

5 Conclusions and Future Work

In this paper we have described our initial steps on system support for application synchronisation in public displays. We have framed application synchronisation within the context of existent architectures for public display networks and described an initial set of application synchronisation requirements. Building on the characteristics of openness, autonomy and scalability, we have described our initial steps on investigating PubSubHubbub, a publish-subscribe protocol for the web space. We have design an initial API for application development using the PubSuHubbub.

More research work has to be done on evaluating performance issues and the programming model, possibly with external application developers. Also, integration with remaining architectural components such as the Display Controller or Scheduler or application subscription and application description models has to be further explored. Finally, as potentially in the future there will be many hubs and tons of publishers and subscribers, we also need to investigate the steps towards a federation of hubs that work cooperatively.

Acknowledgment

Research group supported by FEDER Funds through the COMPETE and National Funds through FCT Fundação para a Ciência e a Tecnologia under the Project FCOMP-01-FEDER-0124-022674.

References

1. Advisory Board RSS. RSS 2.0 specification, 2007.
2. S. Clinch, N. Davies, A. Friday, and G. Clinch. Yarely: a software player for open pervasive display networks. In *Proceedings of the 2nd ACM International Symposium on Pervasive Displays*, pages 25–30. ACM, 2013.
3. S. Clinch, N. Davies, T. Kubitza, and A. Schmidt. Designing application stores for public display networks. In *Proceedings of the 2012 International Symposium on Pervasive Displays*, page 10. ACM, 2012.
4. S. Clinch, J. Harkes, A. Friday, N. Davies, and M. Satyanarayanan. How close is close enough? understanding the role of cloudlets in supporting display appropriation by mobile users. In *Pervasive Computing and Communications (PerCom), 2012 IEEE International Conference on*, pages 122–127. IEEE, 2012.
5. P. Consortium. Deliverable D2.1 - scientific evaluation of pervasive display networkprototype, 2012.
6. H. R. Constantin Taivan, Rui Jos and B. Silva. Situatedness for global display web apps, 2013.
7. N. Davies, M. Langheinrich, R. José, and A. Schmidt. Open display networks: A communications medium for the 21st century. *Computer*, 45(5):58–64, 2012.
8. P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 35(2):114–131, 2003.

9. B. Fitzpatrick, B. Slatkin, and M. Atkins. Pubsubhubbub core 0.3–working draft. *Project Hosting on Google Code, available at http://pubsubhubbub. googlecode. com/svn/trunk/pubsubhubbub-core-0.3. html*, 2010.
10. B. Johanson and A. Fox. The Event Heap: A coordination infrastructure for interactive workspaces. In *WMCSA '02: Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, page 83, Washington, DC, USA, 2002. IEEE Computer Society.
11. R. José, H. Pinto, B. Silva, A. Melro, and H. Rodrigues. Beyond interaction: Tools and practices for situated publication in display networks. In *Proceedings of the 2012 International Symposium on Pervasive Displays*, page 8. ACM, 2012.
12. R. Sayre. Atom: The standard in syndication. *Internet Computing, IEEE*, 9(4):71–78, 2005.
13. O. Storz, A. Friday, and N. Davies. Supporting content scheduling on situated public displays. *Computers & Graphics*, 30(5):681–691, 2006.
14. C. Taivan and R. José. An application framework for open application development and distribution in pervasive display networks. In *On the Move to Meaningful Internet Systems: OTM 2011 Workshops*, pages 21–25. Springer, 2011.