On the Design and Development of *webinos*: a Distributed Mobile Application Middleware

John Lyle[1], Shamal Faily[1], Ivan Fléchais[1], André Paul[2], Ayşe Göker[34], Hans Myrhaug[3], Heiko Desruelle[5], and Andrew Martin[1]

[1] Department of Computer Science, University of Oxford
first.last@cs.ox.ac.uk
[2] Fraunhofer Institute for Open Communication Systems
andre.paul@fokus.fraunhofer.de
[3] AmbieSense Ltd, first@ambiesense.com
[4] Department of Information Science, City University London
[5] Ghent University – IBBT, Dept. of Information Technology, Ghent, Belgium
heiko.desruelle@intec.ugent.be

Abstract. As personal devices become smarter, opportunities arise for sharing services, applications and data between them. While web technologies hold the promise of being a unifying layer, browsers lack functionality for supporting inter-device communication, synchronization, and security. To address this, we designed *webinos*: a cross-device distributed middleware providing interoperability, compatibility and security for mobile web applications. In this paper we present a case study of the *webinos* project, showing how the architecture of *webinos* was specified, designed and implemented, and reflect on several lessons learned.

1 Introduction

The simple client-server online communication model is changing. Web applications in the form of web widgets [**?**] allow for offline usage and caching [**?**], which is particularly important for intermittent internet connections. In addition, mobile networks are becoming overloaded and there is an increasing desire to use local and peer-to-peer communication mechanisms wherever possible. At present browsers cannot switch to other local networks, such as Bluetooth or ZigBee, and so cannot take advantage of them for high-bandwidth local use cases such as file transfer or media streaming between friends. There is a strong case for the creation of middleware capable of supplying web applications with access to local networks and device resources in a standard and secure way.

This requirement for more sophisticated web application middleware motivated the creation of *webinos* [**?**]. The *webinos* platform aims to create a seamless experience for users of web applications on different devices, including PCs, in-car systems, set-top boxes and smartphones. This entails synchronising content and application state between devices, adapting to changes in user context, and providing standard JavaScript APIs to let web applications access device features. The platform also supports a medium-agnostic messaging system allowing

local, peer-to-peer communication channels such as Bluetooth and WiFi. Given that security and privacy are primary goals of the *webinos* project, a common distributed policy architecture and authentication model is supported.

In this paper we present the *webinos* platform as a case study in creating secure middleware for distributed web applications. We briefly describe its architecture and identify lessons learned during requirements capture, specification and implementation. We conclude with several recommendations for platforms with similar ambitions.

The paper is structured as follows. In Section **??**, we provide background information on web applications and interoperable middleware. Section **??** describes the *webinos* concepts and architecture. We reflect on lessons learned during development in Section **??** and make our recommendations in Section **??**. Finally, we present our conclusions in Section **??**.

2 Background: Mobile Web Applications and Middleware

The following sections describe related projects and technologies, including web applications, widgets and middleware.

2.1 Web Applications and Widgets

A web application in a mobile context is a web page or collection of web pages delivered over HTTP which uses server or client-side processing to produce an application-like experience within a web browser [**?**]. In comparison, web widgets are interactive applications for displaying and/or updating local or remote data, packaged to facilitate downloads to a workstation or mobile device [**?**]. As such, widgets are similar to web applications but are packaged for installation.

2.2 Middleware and Related Systems

When defining middleware services, Bernstein [**?**] states that they 'are generic across applications and industries, they run on multiple platforms, they are distributed, and they support standard interfaces and protocols'. In home environments, Raatikainen et al. [**?**] identify that middleware also needs to support *dynamic configurability*, *environment monitoring* – including device discovery and capability management – and a *mobile distributed information base* providing synchronised, consistent, reliable and easily available information across all personal devices.

Several cross-device application middleware systems have been developed previously. CORBA's initial motivation was to provide an environment that would allow distributed applications running on heterogeneous platforms to communicate with each other. Similarly, the Java Runtime Environment and the .Net Framework provide a set of common APIs and a platform supporting the 'write once run anywhere' concept. However, these are primarily aimed at local rather than web applications and have yet to fulfil their cross-device potential.

There are many related mobile technologies. Android and iOS supports *native* applications which run only on mobile and tablet platforms. The Chromium browser and O/S support web applications, and Nokia and Opera also provide cross-device widget runtimes. PhoneGap[6] takes an alternative approach: it compiles platform-independent applications into native packages for each target device, together with a runtime providing standard APIs. This wide variety of fragmented technologies motivates the need for interoperability.

The OMTP's BONDI project in 2008 [7] defined a common set of JavaScript APIs for mobile web applications. BONDI developed a standardized open source web and widget based application environment which has now been replaced by the Wholesale Applications Community (WAC)[8]. Similarly, the W3C Device APIs and Policy Working Group is currently developing device APIs with a similar scope. In addition, the W3C Web Application Working Group is working on several widget standards [**?**].

A common theme of related work is that it aims for *compatibility* and extended web application functionality [**?**] without the *interoperability* required by users with multiple devices. For example, WAC does not make it easy for a web application on a user's PC to access APIs hosted on their smartphone. Data control and privacy are other missing features: existing middleware does not help users to keep control of their personal data across multiple devices, despite the rise in web applications storing data in the cloud [**?**].

3 The *webinos* Platform

The *webinos* runtime has been implemented for three target platforms, chosen after an extensive analysis of device ecosystems. These are Android for smartphones and tablets, Windows for PC, and Linux variants for in-car systems and set-top boxes. The implementation uses NodeJS, a JavaScript runtime for distributed network applications [**?**]. NodeJS had to be ported to Android, but was then available on all target platforms. The rest of the platform was written in device-agnostic JavaScript as well as some native C++.

3.1 Architecture and Concepts

Personal Zones *webinos* is based around the concept of *personal zones*, as shown in Figure **??**. A user's personal zone is the set of all their devices. Each personal zone has a master, the *personal zone hub* (PZH), which coordinates communication, synchronises data and provides access to devices from the Internet. All other devices have a *web runtime* (WRT) (much like a browser) which displays web applications and process widgets. The web runtime has been extended with a *webinos* plug-in to connect it to a *personal zone proxy* (PZP) which implements APIs, provides local access control and communicates with the personal zone hub.

[6] `http://www.phonegap.com`
[7] `http://bondi.omtp.org`
[8] `http://www.wacapps.net/`

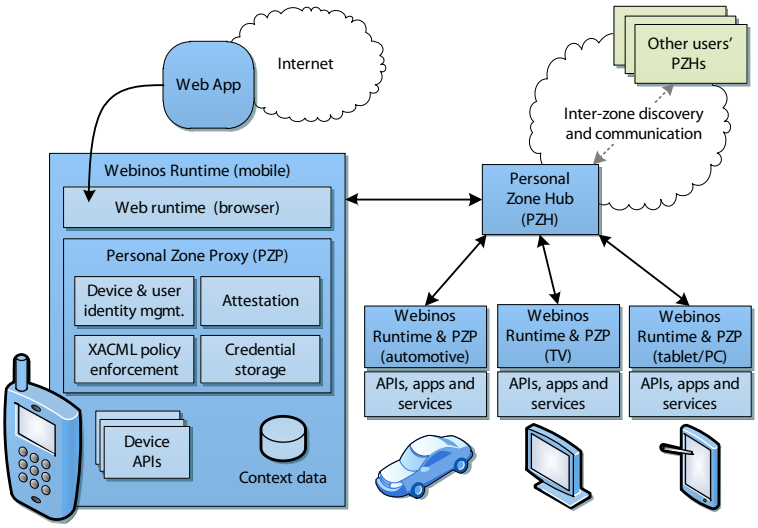

Fig. 1. Overview of *webinos*

Interoperability Interoperability between devices is provided through a common discovery service and eventing system. From an application developer's perspective, remote and local resources are accessible through exactly the same APIs, and no additional remote procedure calls or message handling is required.

The *webinos* runtime can use remote services over an internet connection or through local network technologies such as Bluetooth, WiFi and ZigBee. The list of available devices can be dynamically discovered at runtime. All services are identified through a service-type URI (e.g. `http://webinos.org/api/events`). These can be found and filtered using the service discovery API's `findService` method. Applications call this method and then receive a callback for every matching service found within a specified time limit. Found services are then *bound* to instantiate them and registered for any changes in their state, e.g. the service later becomes unavailable. The service can be used as a standard JavaScript object.

Inter-application communication is achieved through use of the *event* API. Each application can publish and subscribe to event messages of a certain type. The event API's implementation is constantly available on the personal zone hub, which is the logical choice for a publish/subscribe server. However, any proxy may also act as a server for the event API, allowing for ad-hoc collaborations when no internet connection is available.

Security and Privacy The platform has several security features including an access control policy system for APIs, a multi-level authentication model, and encrypted communication channels.

Authentication is available for devices, users and applications. Widgets are authenticated via signatures, and hosted web applications by HTTPS. Users authenticate to the personal zone hub through an OpenID-based login. This allows them to administer their personal zone, adding and removing devices. Devices authenticate to one-another through mutually-authenticated TLS sessions, with each device holding a certificate issued by the personal zone hub. Private keys are stored using operating system provided keyring facilities. Device authentication is sufficient in most scenarios, avoiding the need for users to login explicitly to *webinos*. The use of OpenID at the PZH means that no new identity or credentials are required.

4 Lessons Learned

4.1 Introducing new Concepts and Abstractions

The personal zone concept was developed during the design phase of the project. This turned out to be a useful simplifying assumption for platform developers to manage the complexity of a multi-device cross-platform application system. It was initially assumed that every device would exist only in one personal zone, and that each zone had a single owner. As a result, the zones could be considered small, mostly-isolated networks of devices where only one user was present at any time. This helped with architectural design and security – authentication could be based partly on device identities – and progressed the project greatly.

However, it became clear that this was an over-simplification, as many home devices are shared between people and do not have a single owner (e.g a family PC or television). Assuming that these were all part of one zone would cause problems for applications and services which require better user identities and authentication. The design was subsequently changed to allow devices to run several proxies. Unfortunately, this requires proxies on the same device to be isolated from each other in different user accounts. User accounts have been shown to have poor usability [**?**] in a home setting, and this requires every user to be logged-in for the proxies to run and accept remote requests. It also fails to account for systems which do not provide user accounts. While the solution of multiple proxies per device was acceptable in the short-term, a better implementation is required.

There are several lessons to take away from this experience. Firstly, the concept of a *user* and *user account* is changing, and is often not a useful abstraction or technical solution, particularly when dealing with many different types of device. Secondly, introducing a simple abstraction early on can be a curse as well as a blessing: it can help progress development, but can also limit the satisfaction of certain use cases in unpredictable ways. Finally, given that adapting the technical implementation of a personal zone proxy is a difficult problem for *webinos* platform developers, application developers (who are not as familiar with the platform) may have a difficult time in using and understanding its conceptual model and components. This corresponds to some of the findings from

focus groups [**?**] and suggests that even simple abstractions require considerable documentation.

4.2 Cross-Device Security and Privacy Challenges

While good security practice encourages reuse of existing solutions, our approach to adapt security architectures provided by WAC and BONDI was not straightforward. For example, the policy framework had to be adapted considerably to deal with *user* identity rather than just application and device identity. There were also more threats to consider: because each device is able to access others in the same zone, a malicious application on one (e.g. a PC) could compromise the security of data stored on another (e.g a smartphone). Another new threat is that storing and synchronising data between devices effectively downgrades the trustworthiness of stored data, as it could be affected by the least-trustworthy device. If developers assume that stored data is trusted, they might not check for malicious input, which could facilitate content-injection attacks. Both of these new threats are the consequence of one device in a zone being vulnerable, and used to attack devices which would be secure in isolation. This is compounded by the fact that some devices will update at different intervals and frequencies than others, and known vulnerabilities may persist.

One of the privacy-enhancing aspects of *webinos* ended up potentially conflicting with security goals. As an alternative to storing data on a web server, application developers can choose to store personal information locally in a device and synchronise with the personal zone hub to update other devices. This should increase user control and help satisfy privacy expectations, given that users control their own hub and devices. However, losing any device could result in the disclosure of all synchronised data, including potentially valuable intellectual property and credentials. Also, should the personal zone hub be hosted by an untrustworthy third party, information could be put at risk.

4.3 Developing in a Rapidly Changing Ecosystems

The cost of working in a rapidly-changing ecosystem had some unexpected consequences. Mobile web applications changed considerably during the course of the project. For example, the initial analysis of target platforms proposed developing for Meego, which has since been largely abandoned, as were several draft W3C APIs such as the Gallery API [9].

The up-front specification approach also had problems. Although specifications were supposed to support early application development, they were not useful without an implementation. Application developers had to create their own stubbed APIs, and this separated platform and application development effort. Our experience suggests that it is better to implement APIs early rather than just committing to adopting a standard in the future.

[9] `http://dev.w3.org/2009/dap/gallery/`

4.4 Scoping the Middleware and Specifications

Deciding on the scope and responsibilities of the middleware was a challenge. In particular, distinguishing the security and privacy responsibilities of the web applications from those of the middleware. Clearly the responsibility is shared, but identifying where each attack is mitigated and what is expected of the applications has proven difficult. Part of this issue is user perception: the *webinos* platform has security and privacy as a selling-point, but cannot guarantee this without secure applications. Furthermore, there is a temptation to offload functionality from the platform to applications to save cross-platform development. Keeping track of where features were implemented required more communication between developers of the platform and applications.

A related issue was defining the limits of the public specifications. The platform has undocumented features and design choices that do not affect the applications, but might have an impact on future extensions. With an open source project, these details tend to be shown only in the code, which may cause design problems later on. Therefore, the scope of documentation should have been better defined.

5 Recommendations

We make the following recommendations for other projects attempting to provide standard runtime environments on multiple platforms. First, the impact of cross-device interaction on a security architecture should not be underestimated. Even creating the threat models and identifying misuse cases is time consuming. It is easy to make mistakes where the ambient authority of the local device is confused with remote requests which require access control. The personal zone concept helps, as it assumes that any messages received has come from an authenticated source. We suggest devoting more time to security analysis of cross-device issues early on in the design process, and considering the security of scenarios rather than individual platforms or devices.

Second, any assumptions about cloud and application capabilities should be made early in the system design. The natural tendency will be to push functionality away from the middleware and on to either applications or cloud-based services to save cross-platform implementation effort. However, it should be decided in advance which features *must not* be implemented as cloud services or be delegated to applications. Part of the design should involve allocation of security responsibilities between the applications, middleware and cloud services. Any responsibilities which are placed on an application ought to have thorough documentation suitable for developers. Otherwise, calling a middleware 'secure' can give the impression that any application using the middleware is automatically secure and trustworthy.

Third, solutions respecting privacy can result in fragile endpoints with potential for data loss and insecurity. Middleware is not the place to judge the balance between privacy, user control and security, as this may be application-specific. Designing for a specific set of applications can help make progress, and

we suggest embracing this approach, even though this may fail to satisfy some later requirements

6 Conclusion and Future Work

Designing a cross-device interoperable application middleware is a challenge. We have described the approach taken during the *webinos* project and the insights gained from 18 months of design and development. For future work we intend to validate our findings through developing the platform further and building applications. This will include better modelling and analysis of the system and a deeper inspection of security and privacy. More applications will be developed, and we will also investigate how to apply the personal zone concept to corporate use cases.

7 Acknowledgements

The research described in this paper was funded by EU FP7 *webinos* Project (FP7-ICT-2009-5 Objective 1.2).

References

1. The W3C: Widget Packaging and XML Configuration (W3C Proposed Recommendation). `http://www.w3.org/TR/2011/PR-widgets-20110811/` (August 2011)
2. The W3C: Offline Web Applications (W3C Working Group Note). `http://www.w3.org/TR/offline-webapps/` (May 2008)
3. The webinos consortium: website. `http://webinos.org/` (February 2012)
4. The W3C: Mobile Web Application Best Practices. `http://www.w3.org/TR/2010/REC-mwabp-20101214/` (December 2010)
5. The Wholesale Application Community: Glossary of terms. `http://www.wacapps.net/glossary` (August 2011)
6. Bernstein, P.A.: Middleware: a model for distributed system services. Commun. ACM **39** (February 1996) 86–98
7. Raatikainen, K., Christensen, H.B., Nakajima, T.: Application requirements for middleware for mobile and pervasive systems. SIGMOBILE Mob. Comput. Commun. Rev. **6** (October 2002) 16–24
8. Mikkonen, T., Taivalsaari, A.: Reports of the web's death are greatly exaggerated. Computer **44**(5) (may 2011) 30 –36
9. Lin, D., Squicciarini, A.: Data protection models for service provisioning in the cloud. In: Proceedings of SACMAT, ACM (2010) 183–192
10. Joyent, Inc.: Nodejs website. `http://nodejs.org/` (February 2012)
11. Egelman, S., Brush, A.B., Inkpen, K.M.: Family accounts: a new paradigm for user accounts within the home environment. In: Proceedings of the 2008 ACM conference on Computer supported cooperative work. CSCW '08, New York, NY, USA, ACM (2008) 669–678
12. The webinos consortium: User exepctations of security and privacy phase 2. `http://webinos.org/blog/2011/11/01/webinos-repot-user-expectations-of-security-and-privacy-phase-2/` (November 2011)