webinos whitepaper

June 2011

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

This report is a public deliverable of the webinos project. The project members will review any feedback received; updates will be incorporated as applicable. The webinos project reserves the right to disregard your feedback without explanation. Later in the year, update to the report may be published on www.webinos.org as well as being made available as a live and community maintainable wiki.

If you want to comment or contribute on the content of the webinos project and its deliverables you shall agree to make available any Essential Claims related to the work of webinos under the conditions of section 5 of the W3C Patent Policy; the exact Royalty Free Terms can be found at: http://www.w3.org/Consortium/Patent-Policy-20040205/.

This report is for personal use only. Other individuals who are interested to receive a copy, need to register to http://webinos.org/downloads. For feedback or further questions, contact: **editors@webinos.org**

Vision

webinos is: a collective project to make the web work for applications. webinos has a vision to build a multi-device, applications platform based on web technology that:

- allows web apps to run seamlessly across multiple devices and to use resources across devices
- allows web applications to communicate with other web applications and (non web components) over multiple device
- links the application experience with the social network
- achieves all of the above in a security preserving manner
- explicitly targets the four distinct “screens”: the mobile, the PC, the in-car (automotive) and the home media (TV) devices

The intent in webinos is to translate the success of the web as a distributed document publishing system into a successful, distributed applications platform.

The webinos platform should be built upon and move forward the required open standards. This platform should have a concrete implementation that is accessible to all as an open source asset.

Technically, all of this should be achieved reusing the core development technologies that have already proven themselves on the Web (HTML and JavaScript), affording the benefits of speed of development and access to a large developer talent pool.

The innovation webinos brings shall not just be technical; by embracing an open web culture, we hope to create an application framework that does not favour any particular corporation, and on which may parties can collaborate, and from which many companies benefit.

Background

Web applications are everywhere: Chrome, WAC, HP WebOS, Nokia, WAC, PhoneGap, Titanium, and Rhomobile etc etc. All these newly created application platforms and tools are based on the premise that application developers use web technologies first and foremost. But what are web applications and why are they proving so popular?

What is a web application?

The Google Docs or Microsoft’s web based outlook interface, look, to most people's eyes, like fully functioning applications. The typical end user does not know, and does not care whether this is a native (C++ built) application, or running magically on a web browser. These application are “vanilla” Web 2.0 apps, in other words they do everything you see, using basic web mark-up, a dynamic object model (DOM), a scripting language (JavaScript) and a communication mechanism (XHR – XMLHttpRequest). However, these are just the tip

of the iceberg. The next generation web applications have some interesting new possibilities. Within the webinos community we have a vision, where a fully fledged web application can:

- take full control of the devices UI – and are not rendered with the pre-configured chrome of another application
- effectively run background processes and present a good user experience to the end user
- make full use of the device capabilities and APIs available on the device
- insulate the inherent risks of API access with robust security model
- run when not connected to the web
- be packaged and distributed, again without assumed connection to the web
- make full use of the device capabilities and APIs available on the device
- take full control of the devices UI – and are not rendered with the pre-configured chrome of another application
- effectively run background processes and present a good user experience to the end user
- may run remotely from their user interface
- insulate the inherent risks of API access with robust security model
- be combined (mashed up) in creative ways not envisaged by the original application author
- demonstrate a high degree of portability, running essentially the same code on many devices

Why are they proving so popular?

There are a number of possible reasons that so many companies (and individuals) are embracing the new form of application development:

The Web as a global platform: W3C's vision of "One Web" encompasses all devices and is accessible to all. This presents a single open market based on shared standards rather than one fragmented into proprietary fiefdoms. This is a boon to commerce, in the same way that a common market has encouraged trade and prosperity across Europe, with the removal of barriers, and the economies of scale reducing development costs, reducing maintenance costs, and freeing up technical and commercial resource to focus on more differentiation areas, with respect to their competitors. The Web then becomes a platform fulfilling the vision of Web 2.0

Public roadmap: Unlike a proprietary product, the web technology space, whether it is innovating within the W3C forum itself, or within one of the co-dependent open source projects, is fully transparent. This meant the roadmaps, over a reasonable time frame, are fully transparent. When dealing with device companies (whether mobile, pc, automotive, or home media), their commercial planning horizons are a considerable way out. A public transparent roadmap is clearly a positive thing from this perspective.

Low technical barriers to entry: It is a fact that web programming is easier to get into than more typical native application programming (that typically requires C, C++ or Java skills). Learning basic declarative HTML tag representation, upon which you can slowly build with easily experimental JavaScript programming, gives a much softer entry into the

programming world. Further, HTML and JavaScript as interpreted languages, mean that for a new developer to get started all they need is a working web browser and a text editor. Contrast this with the complex compiler tool chains required to get native development environments up and running.

Although, you could debate that beyond a particular level of sophistication (when you start using complex JavaScript libraries, object-orientated techniques and asynchronous programming) whether these differences still exist, web based technologies mean that instead of being “thrown in the deep end”, you can paddle around in the shallows and immerse yourself slowly, and at you own pace, to the more esoteric programming depths.

Large skills base: The corollary of the above point is that there is a larger skills base for web programmers than native developers.

Large asset base: The ubiquity of the web, the fact that almost every corporation and organisation has a website, and increasingly now, even individuals, means there are massive amounts of content "out there". To support this content, a strong ecosystem of tools and development applications has emerged (both proprietary tools and open source). The net effect, is that any developer looking to create web application content is well supported.

Quick to develop for: and faster time to market: Another implication of the simpler technology, and tools base, is that typically web based applications can be developed quicker than native applications. This has important, valuable time to market implications for application developers and device manufacturers

Easy to deploy: web content is typically hosted on a website and is can be downloaded and executed almost instantly instantly. Although, content must still be thoroughly tested, compared to native application development this deployment cycle, and subsequent maintenance cycle is more efficient and dynamic.

IPR unencumbered: The specifications on which the web is based are designed to be unencumbered by IPR. This has two immediate positive commercial knock on effects. Firstly, it removes the immediate and absolute requirement (or risk) to pay licensing fees to proprietary technology owners. Secondly, and perhaps more importantly, it does not bestow any core strategic advantage onto any single company.

Synergies between Open Source and the Web: Finally and as a partial consequence of the lack of essential IPR on the core technology is that it is easier to create open source assets for. Or, perhaps, it is easier to state the inverse for the Web: it is very difficult to create an viable open source project for technologies on which there is known IPR that may be asserted (simply because there is then an implied liability). It is no accident then that the web ecosystem has been influenced, if not entirely dominated by open source projects such as Apache, Mozilla, and WebKit.

The counter position

Of course, nothing is that black and white. In considering the future of web application, we should always bare three things in mind:

1. Web applications are no panacea. There are whole classes of problems for which they are currently in appropriate. Often those requiring very high performance graphics
2. It's not all roses. As any "real" web developer will tell you, those little differences between browser behaviours, can create huge amounts of application development effort. Basically, as with any standard, it is never perfect
3. Finally, and most importantly, it is not a competition: Native vs Web. They are both tools and as with any tool, you pick the right one for the job (at the right time). Also, increasingly, native application development is integrating web technology and pre-existing web content as a first class, peer component of the application being developed.

The webinos Innovation

If the web application phenomenon is so pervasive, and the arguments for following this technology are so persuasive, why do we need to invest time and effort into a new project?

The fact is there are essential elements missing. If we don't fill these gaps, then the market will fill them with vertical, fragmented, proprietary solutions. Therefore if we want to preserve the vision of interoperability for web applications, we need to work on these gaps in an open and collective fashion.

This is what webinos is: a collective project to make the web work for applications.

At the next level of detail, we can start digging into exactly what critical innovations are required to make web applications across device ranges a success. It is probably easier to do so, by framing each innovation against the principle problem or issue it addresses:

How can a web application find other web applications and services?

Device and service discovery: webinos shall provide new methods for discovering applications, services and devices over a virtual webinos network. This discovery mechanism may make use of notions of social proximity or knowledge of device capabilities.

How can a web application communicate in private networks (non publically addressable internet) and over hardware connections where no IP stack exists?

Connection management: webinos shall provide new mechanisms for communicating between devices. These connection mechanisms shall innovate over pure HTTP and web sockets, providing notification and message passing mechanisms that are optimised to the environment, creating new technology, or integrating existing technology (such as XMPP), where it proves useful. webinos shall provide solutions to the discovery and connection issues not only for applications and devices on the open addressable internet, but shall deal with hard cases of firewall navigation, communication within a closed intranet and connection to local physical connections, that may not have full IP stacks, for example Bluetooth, zigbee and RFID. This innovation will be critical to realising the vision of the "internet of things".

How can a web application access APIs and capabilities outside of the normal security sandbox of the web?

If any web page could open files on my computer and send emails, without protection, the current problem we have with viruses would explode a thousandfold.

webinos APIs: webinos shall extend from the current state of the art APIs to be found in W3C, WAC, and PhoneGap APIs. It shall enhance these APIs so that they work better for environments where we are working on several different devices (PC/mobile/TV/Car). Also applications running on these different devices use each other's services (remote APIs) in a secure manner.

How can we breach the web security sandbox and still grant access to secure APIs in addition to protecting the legitimate business interests of content creators?

Looking at this question we need to break the problem down:

- If we want to protect the end user from "rogue" web applications, we need a better security layer
- If we want these apps to be "interoperable" and not create monopolistic web application stores, we need to standardise this security layer
- If we want developers to make money from their applications, and not be subject to piracy and copyright, we need a rights management layer
- If again we want this ecosystem to have scale, and not create a single web app store, which every piece of content must go through, we must standardise this rights management layer

Policy: webinos shall create innovative distributed policy mechanisms that shall:

- Preserve an end user's preferences no matter what device they are using
- Police policy in a distributed manner, so that an end users "policy" is enforced no matter where in the cloud the API or resource is accessed
- Be based on strong notations of application, device and end user identity

- Shall model critical concepts from a user's social graph so that a user can consistently express his relations and permissions he/she wished to extent to their contacts
- Mediate access to protected APIs where needed
- Protect the assets and code of a content provide in an appropriate manner

How can a web application make a usable secure connection to remote assets, including network APIs?

webinos will fundamentally innovate the way an application connects to external resources, both local and remote. It will create abstractions that are JavaScript friendly, and easy for a developer to use, but implemented to be secure by design: not requiring repeated logons of confirmation prompts.

Philosophy – Principles of Engagement

In the above text you will find a description of what it is that webinos will try to do; to be successful what is equally important is how it goes about it. There are a few fundamental principles that webinos shall adhere to increase its probability of success:

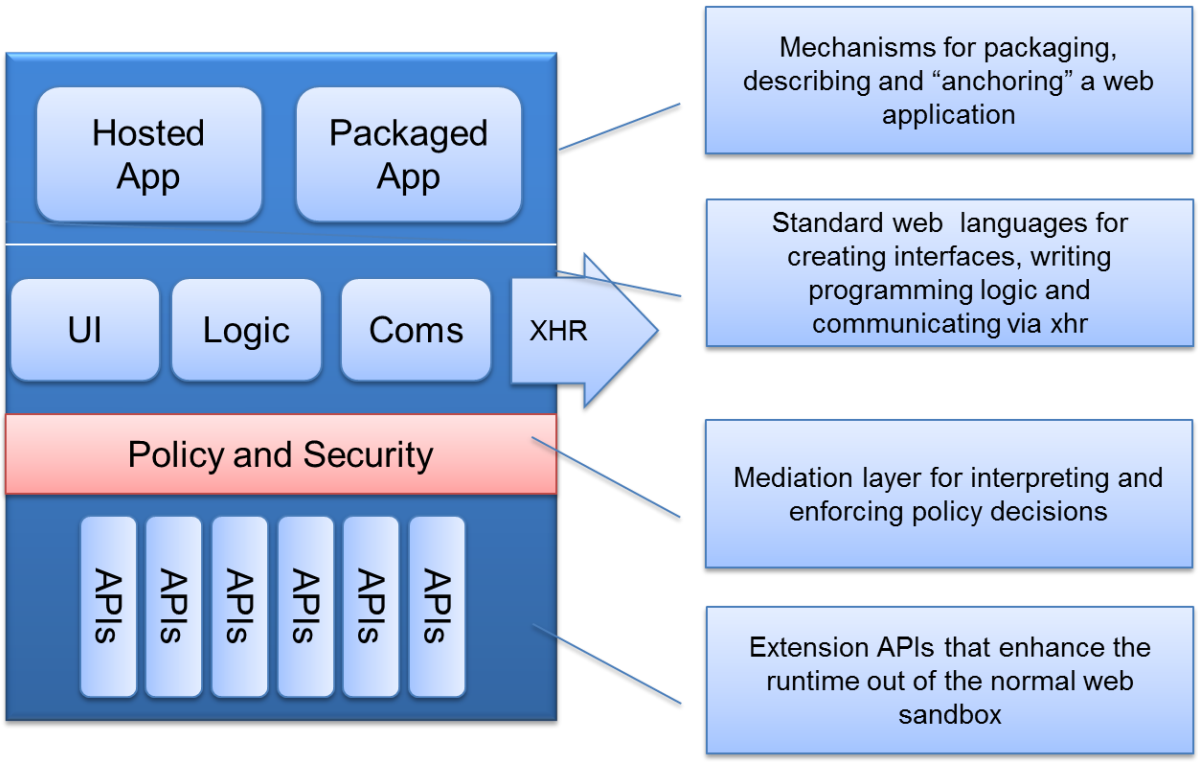
- Focus: webinos shall re-use existing technologies where possible and only focus to close gaps
- Open standards: webinos shall where possible prefer open standards, unencumbered by IPR
- Open source: webinos shall place its assets under a common shared open source framework - to enable the widest possible adoption and usage
- Pragmatic: in all of its standardisation and architectural work, webinos shall implement first and standardise second. Decisions should be informed by practical working code examples at all times
- Transparency: the governance of the open source code asset base shall be open, transparent and not implicitly or explicitly prefer any single company or actor in the value chain
- End user centricity: for all design consideration, most especially the contentious policy and analytics issues, webinos shall make the rights and needs of the developer and end user, the foremost consideration.
- Expertise: webinos is already working in a team composed of cross domain players, coming from the industry as well as academia and thus enabling both, research as well as delivery into concrete products

Architecture

The following architectural details, are a snapshot of current work in progress. The key technologies are still under development and therefore subject to change. This should, however, give a strong indication of direction and technology.

Baseline

The baseline architecture - existing and already used today - is informed by the current state of the art, this encompasses BONDI, WAC, ChromeOS, HP WebOS, Nokia implementations and is the most probable implementation for the W3C Widget and DAP specifications.



The base line architecture consists of four coarse grained conceptual layers:

- A packaging layer for physically distributing the web application, adding descriptive meta data to the application and embedding identity and least privilege security elements to a web application.
- Web interpretation layer: this maps closely to a chrome-less web browsing component, such as can be found within Webkit or Mozilla code bases. It consists of HMTL interpretation, and JavaScript interpreted and an object model on to key dynamic elements of the web page, including the XHR communication mechanism
- There exists a policy layer, to mediate security sensitive action. A basic security layer is to be found in standard browser. This is the policy element that can intercept popups, file downloads or inhibit scripts or plugins running on the web runtime. More advanced policy mechanisms are to be found in widget implementations and extended runtimes. These advanced policy layers will mediate access to remote network components (e.g. WARP) on a least privileged basis. Or may implement global (user preferences) on access to sensitive capabilities such a location access or contacts etc.
- Finally there exists an extensible framework for adding new and exciting APIs that enhance the standard web browsing experience.

These layers are copiously documented with respect to current state of the art implementation in the webinos state of the art analysis documents [1].

This architecture has proved successful, as demonstrated by the number of implementations now employing it; however, it has been designed with the single application on the single device, connected to a standard web server in mind. There are also as the 2.8 webinos deliverable shall demonstrate, a number of security issues, which are not adequately addressed in the current implementations.

And this sums up the technical scope of webinos: to build upon the existing foundations of web applications, and to extend this with the architectural elements necessary

- to allow web apps to run seamlessly across multiple devices and to use resources across devices
- to allow web applications to communicate with other web applications and (non web components) over multiple device
- to link the application experience with the social network
- and to do all of this in a security preserving manner.

In the new world of the cloud where provenance of data and applications can be hard to detect, the fundamentals that bridge the gaps become essential

A not insignificant challenge!

Key architectural components

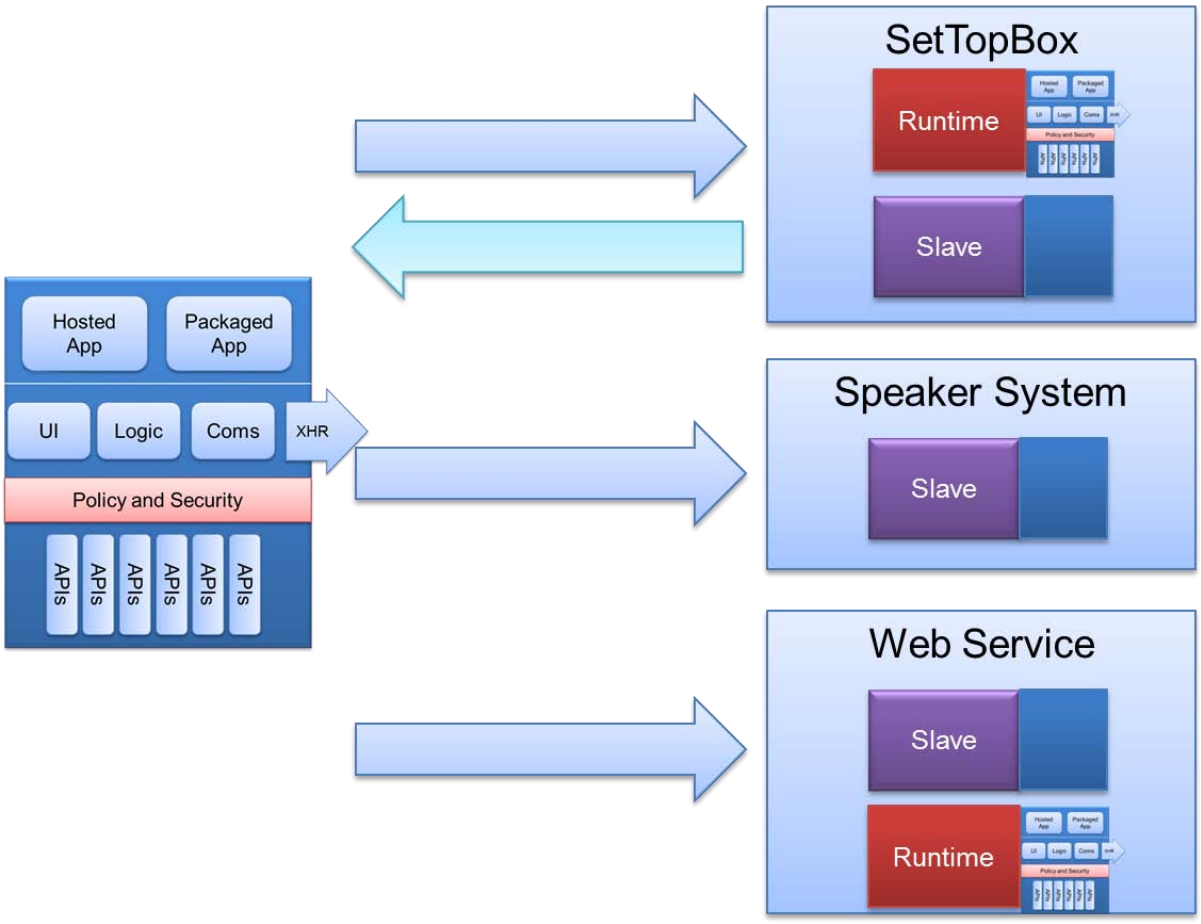
So how do we plan to achieve all this? Below we will introduce some of the key components:

webinos Browsers features

The webinos Platform extends the browser with new capabilities to discover and access services on other devices whether they are local or remote. This allows your tablet to stream audio to your room's speakers, or to work in tandem with your TV to enrich the viewing experience with live discussions with your friends including access to ancillary information, e.g. where to locally get the ingredients for a cooking program. The webinos Platform enables applications to expose services for other devices to use through lightweight discovery and connection mechanisms. The browser thus also acts as a server.

webinos Servers features

Not all webinos devices will include a browser. Some will instead provide specific services, such as the room's speakers above. Other roles include communication hubs that bridge different interconnect technologies; basic sensors and actuators, e.g. to control heating and lighting; media servers, and context managers. Public servers are needed to support wide scale discovery and access to services, including establishing P2P communication paths.



In this new world of web applications, we shall see the architecture move on again. The traditional, simplified one-to-one client server relationship becomes blurred and replaced by new models:

- Applications span multiple devices with a focus on usability and peer to peer communications. The user interface might be split across devices, or it could migrate from one device to another as appropriate to the context. It must be easy to discover devices and services, and to be able to give them human meaningful names.
- Social connections provide the key to realizing multi-user applications where trust relationships determine communication paths irrespective of the underlying network topology and hurdles such as Firewalls and Network Address Translation (NAT). This change of thinking is required to get the best out of the new technologies along with the ubiquitous ease of use needed for widespread adoption.

To illustrate, webinos creates a world where:

- A mobile device runs a web application, e.g. an Electronic Program Guide (EPG) viewer, and the set-top box runs a web application (also an EPG viewer). But the mobile would like to make use of the set-top box data, so it must also act as a server providing information to the mobile. This scenario is not a new one. On Android and iPhone you can find an ecosystem of mobile native applications, set-top box native applications, and background set-top box servers that deliver this service. The webinos vision is to bring this basic use case and deliver it all through a single

technology, and an integrated, easy to build web technology. But all of this delivered under a common security model.

- A second example shows how, again using web technology, we have a media player – a rich HTML5 based application – wants to make use a “dumb device” – a set of speakers. In this scenario the mobile device runs the master, but using web technology (and not necessarily) HTTP, can securely connect to and stream to the speakers, as a webinos server. It is also worth noting here that what makes a webinos browser is its ability to interpret an execute webinos packaged programming script, and the server is remotely executable services. We can also fit native applications into this model, which may not be able to execute webinos script (cannot be the browser) but can expose services, ie act as a server.
- A third and final example, simply shows how a classic web server can be thought of as webinos server. webinos service exposure, is little more than a web service or restful interfaces, packed more effectively. This is the traditional model. However, we can start to see some interesting opportunities when we enhance the classic server role so that it can act as a browser role also. Interestingly a web server implemented using node.js has almost all the technical components required to allow it to act in browser mode.

Discovery, Messaging and Identity

Discovery, messaging and identity are three distinct concepts, and are developed as such within the webinos project, however it is easier to see the benefits of the innovations when we consider them together.

Let us first look to see how these concepts fit in the current state of the art technologies. Using the mobile EPG example again, how does this work in current implementations. Download any “remote” application from app-store and you will see this familiar experience

- Discovery: usually enter the raw IP address of the set-top box on the mobile phone. If you are lucky, a local network name will resolve.
- Identify: generally the server generates a secret pin that has to be entered manually on the phone, and this doubles as security (policy) also.
- Messaging: is either a custom made asymmetric protocol over HTTP, or a symmetric protocol implemented over TCP.

Examining typical HTTP communication scenarios, is also illustrative

- Discovery: simply does not exist in any general sense. The web itself is a set of interconnected notes (URI) and they are discoverable by manual insertion of an address, manually navigating the graph (the web), or using a proprietary engine for discovery. Discovery of clients is even less well supported in an interoperable fashion.

- Identity: should be broken down for the individual elements
  - Users: some form of username and password, but dealt with (generally) at a 1:1 level between client and server (every server has its own credentials mapping)
  - Devices: user agent string is the way a device typically advertises itself to a web server. However, the lack of standardisation in this area, has meant

support is inconsistent and the declared features have lagged the useful discoverable features. Identifying instances of device is not supported
    - Applications: are not supported at all, there is no way for services to systematically know what (web) application is attaching to it, either the type of application or the specific instance. Instance of application are not supported either.
    - Services: services (web services) can be identified by domain. But unless the services is hosted on a TLS/HTTPS session, this is not reliable nor particularly informative.

More generally there is a hard technical use-case/scenario that webinos directly tackles, which makes these problem challenging in the extreme: webinos attempts to

- run over publicly addressable internet
- locally over private connections (that may not be connected to the internet)
- and even peer to peer over hardware connections that might not even support TCP

These are three very different scenarios where the assumptions that can be made of the underlying technology available vary greatly. This model requires some very sophisticated, but usable abstraction for discovery, identify and messaging.

The approach within webinos, is to tackle each of these areas thoroughly, re-architecting the primitives as required in order to fully satisfy the requirements we have identified. A ground up reworking of identity, messaging and discovery in a web friendly manner, is deemed to be the key to delivering multi device, web applications that have been targeted.

The grand vision is to wrap these primitives into a consolidated “overlay network” or personal zone, which is a secure, optimised virtual network that binds the devices, application and services belonging to a user. Within this space, the entities will be robustly identified, easily discoverable and between which entities can be addressed and messages irrespective of the type of physical network they are hosted on.

Overlay Networking Model

The Overlay networking model aims to provide a logical connection between components regardless of the underlying network topology. This will exploit mechanisms for establishing peer to peer (P2P) UDP and TCP connections that work across well behaved NATs. This is needed to enable applications to expose services for others to use independently of whether the devices are on the same local network or are remote, generally involving traversal of multiple NAT boundaries.

Advertising Services and Discovery

webinos applications will expose services (including device capabilities) for others to use. These need to be advertised so that they can be easily discovered. This should be possible both through direct connections, e.g. multicast DNS, and also remotely. We will need to

explore ways to extend browsers to enable web page scripts and standalone web widgets to listen for service requests.

Your devices could (if you chose to) advertise the URL for your social agent (see next section). This would make it easy to find out more information starting from a basic local discovery probe.

P2P and Social Agents

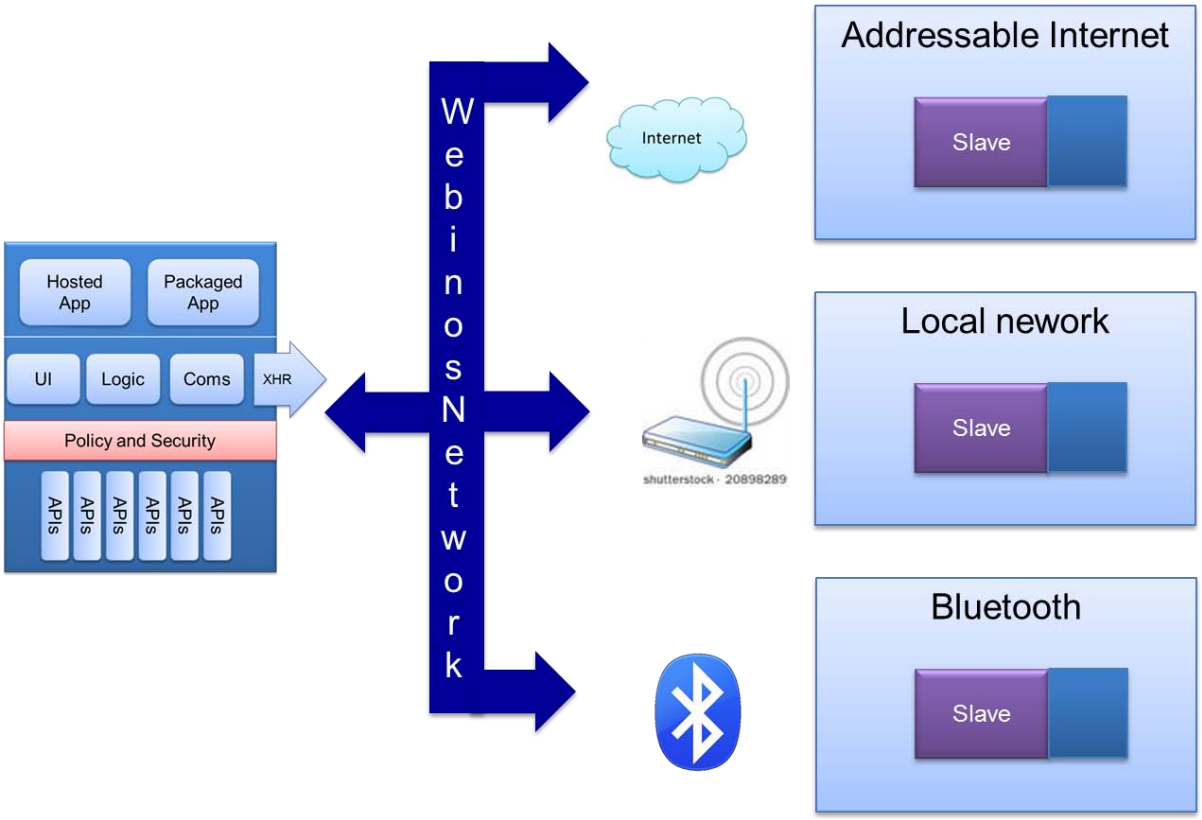
The need for external servers to facilitate P2P connections can be combined with ideas for distributed social networks, where each user would have one or more URLs for webinos servers that can be used for:

- discovering information about that user's services and context
- setting up P2P connections to services provided on behalf of that user
- synchronizing shared state across the user's devices
- representing the user's social network and information on those users
- passing messages to the user's social connections
- implementing trust relationships based upon authentication and access control

Thus, to get access to information and services, you would first have to authenticate yourself as a friend, colleague or member of the public, as according to the trust models established by that user. There are huge opportunities for improving on the deeply flawed approaches to identity and authentication in today's web. The services provided to, or on behalf of, that user represent a huge opportunity for application developers.

Distributed Search and Discovery

webinos may need to consider the role of distributed search and how to realize that in a way that respects privacy and trust relationships. One solution is to export the problem to a proprietary search engine like Google, but another is to implement efficient distributed search using server-modules and building upon algorithms like Distributed Hash Tables. In many cases, the social agent would be hosted by the user's Internet Service Provider using open source modules that implement distributed search collectively with other servers. Each user would thus pay just for his part of the system.

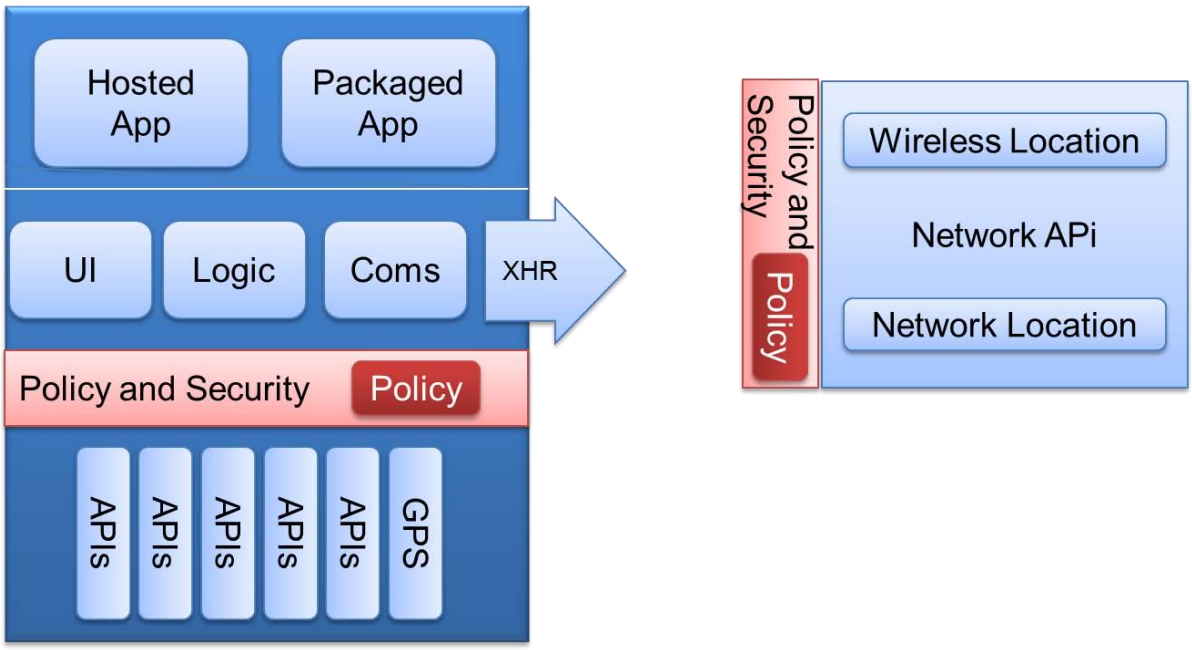


Portable, interoperable and distributed policy definition and enforcement

By providing even greater interoperability between devices, the webinos platform naturally faces new challenges in maintaining security and privacy for users, developers and content owners. One way in which these challenges will be met is the provision of distributed policy enforcement. Policies control the device capabilities a web application is allowed to access, and a cross-device application must be constrained by policies referring to every device. However, policy management is currently implemented differently in each platform and users should not be expected to repeatedly make trust decisions about the same application in a new context. webinos will provide one interoperable policy system which will control applications in the same way across all devices.

Providing cross-platform policy enforcement involves a number of architectural components. Policies must be held and enforced by a Policy Enforcement Point on each device. They must also be synchronised between devices, so the installation of an application on a user's tablet results in the sending of new policies to their mobile phone, car and set-top box. This policy synchronisation will take advantage of the overlay network model mentioned earlier. The policies themselves must be protected on each platform, particularly those relating to privileged device APIs which the manufacturers have obligations to protect. webinos is well-placed to unify policy management and provide a solution for all stakeholders, including users, device manufacturers, developers and network operators.

webinos will be innovating in the content and structure of policies. They must be able to describe people with multiple distinct online identities, for example to separate professional and social networks. It must also be possible for data residing on company-owned devices to remain under the control of the company while also maintaining reasonable levels of privacy for the end user. Usable privacy controls must also be provided so that users can easily and securely become anonymous when required. Furthermore, user should be able to delegate the more complex policy configurations to trusted third parties – perhaps their work IT department, a service provider, or even a more technology-savvy friend. Describing these scenarios as policies can potentially provide greater levels of security and privacy to users of webinos-enabled devices.



Privileged Apps and Services:

The creation of a robust policy framework, as discussed above, is essential to a usable, scalable, consumer focussed security model, that works across devices. And this is the mechanism that shall support "consumer centric privilege ". (In other words, you can use this app, service, data, because I as the end-user/owner of this allow you to.

But, it is our ambition to create a thriving commercial ecosystem. We therefore cannot and must not neglect the needs of the businesses and technology component owners that must also operate in this ecosystem

Specifically, almost every device has some "privileged APIs and services", that are not meant for general consumption. And around which there are legitimate business and safety reasons to restrict access. For example:

- Car: the engine management system

- Mobile: low level SIM access APIs (interfere with network) or deep device management APIs
- Set top box: conditional access module interface

Theoretically, a policy can be written that can protect, restrict access to these APIs through privileged or blessed applications.

Practically, a lot or detailed security work, and commercial ecosystem testing needs to be performed, before anyone would trust these mechanisms sufficiently to place significant business and security issues in its care.

This problem is the focus of the privileged services work groups working in webinos.

Web Technology and Web Foundations

webinos, is of course based upon pre-existing web standards. In the course of it's development it will review and make change recommendations to some of these foundation components. Likely areas for consideration are:

Widget related

- To attempt to reconcile the use of local JavaScript APIs with access to remotely define web based APIS. webinos will look to creating common abstractions, delivered as local JavaScript, but implemented in both remotely and locally, but dealing with the security and performance optimization issues in sensible ways.
- Extending the outreach of a widget to other devices
    - Provide possibility to make 1:1 installations of web applications/widgets on another device (transportable widgets) which allows to use an application further on another device (which might have some special additional features needed by the application)
    - Partially code outsourcing to another device which only contains code suitable for a given task, e.g., take a picture or send an SMS (including code created at runtime). Here we could have an API that allows the creation of widgets on the client site. The newly created widget can then be installed on other devices where it is executed. Providing the application to outsource as common web pages may also an approach in conjunction with HTML5 app cache. However, this may also influences W3C Widget Update over HTTP procedure.
- To make dynamically created or statically provided widgets available to other devices we could provide web server functionality via JavaScript to widgets. Thus widgets could act as common web server and provide the code exchange/widget download functionality by their own.
- Even if we allow very huge payload sizes for the webinos Eventing/Messaging/Notification API it would not allow streaming communication between widgets. For proper Widget2Widget communication, especially for streaming data communication, we could provide some kind of data pipe / socket API to widgets.
- With socket and http/web server APIs we would enable web applications to be first class citizens in the web like native applications.

- Need for automatic execution of code on another device. If an application outsources code which is remotely installed we need to trigger the execution of this application to enable application collaboration.

Extensibility related:

- JS design guides for development of new JS APIs (completely new features, e.g, use device status if possible etc.)
- The open source webinos project website could host a API directory which covers additional APIs or extensions to existing APIs and level of spreading
- Provide best practices for developing extensions for “most common” webinos runtimes
- With the above remote resource access approach there would be no need to define also “remote access” protocols to be interoperable
- Big open question again is versioning and backwards compliance which is relevant for webinos core APIs as well as for webinos extension APIs

Rich suite of APIs:

The precise list of APIs that webinos shall support is still in flux, but likely contenders are as follows

webinos base and generic objects/interfaces

API name	Use existing API / new specification	
webinos core API module	New specification	

Discovery and access to remote services

API name	Use existing API / new specification
findServices API	New specification
bindService API	New specification

HW Resource APIs

API name	Use existing API / new specification
Geolocation API	W3C Geolocation API

Device Orientation API	W3C DeviceOrientation Event
Generic SensorActuator API	New specification. See The sensor module - draft
Media Capture API	TBD W3C Media Capture or W3C HTML Media Capture API ?
Devicestatus API	WAC 2.0 devicestatus module
Device Interaction API	WAC waikiki deviceinteraction module
TV and STB control API	New specification. See The tv control module-draft
Vehicle API	New specification. See The vehicle module-draft
NFC API	New specification

Application Data APIs

API name	**Use existing API / new specification**
Contacts API	W3C Contacts API
Calendar API	W3C Calendar API
Messaging API	WAC 2.0 The messaging module
File Reader API	W3C File API
File Writer API	W3C File API: Writer
File API: Directories and System	W3C File API: Directories and System
Gallery API	W3C Gallery API
Payment API	New specification that is a JavaScrip wrapper API above GSMA OneAPI Payment RESTful Version 1.0

Communication APIs

API name	Use existing API / new specification
Event API	New specification

Application Execution APIs

A number of APIs identified from webinos requirements are stated at Application execution APIs wiki. However, it is currently unclear what will be supported in phase 1.

Security and Privacy APIs

API name	Use existing API / new specification
Platform attestation API	New specification based on TCG TSS TPM_Quote functionality

User and Application Data APIs

API name	Use existing API / new specification
User Profile API	New specification
User Authentication API	OAuth, Facebook Authentication API

webinos Contributors

webinos is a powerful but well balanced consortium, bringing together key players (industry and academia) across the four domains. It currently comprises 22 organisations and secured funding to the tune of €14 million over 3 years. It is made up from

- Mobile Operators: Telecom Italia, Telefonica, Docomo, Deutsche Telekom
- Handset/Vehicle Manufacturers: Samsung, Sony Ericson and BMW
- Universities: University of Oxford, Instituto Superiore Mario Boella, National Technical University

of Athens, Politecnico di Torino, Technische Universität München, Università di Catania
- Research Insititutes: Fraunhofer-Institute FOKUS, IBBT , TNO
- Analyst houses and Consultancies: VisionMobile Futuretext
- SME: AmbieSense, Antenna Volantis Systems, Impleo
- Standards bodies: W3C

However, this is only the start. To succeed webinos must cast its net wider. The long term vision is to create the webinos foundation, to which organisations can join. This shall be the long term resourcing model, to evolve and administer the webinos collective assets. In the shorter term the webinos consortium shall be opening its doors to external parties through two different models:

- webinos affiliate program: whereby organisation can apply, get access and contribute to webinos deliveries, code assets and meetings
- Invited expert program: for individuals with profound expertise and vision who wish to participate and contribute to the work program

Relationship to existing initiatives

webinos does not exist in a vacuum. It explicitly recognises and has a concrete plan to coordinate with the other bodies and technologies that are relevant to its execution. These include but are not limited to

W3C HTML5

The HTML5 emergent standards, are one of the foundations upon which webinos is built. HTML5 itself is a vital part, but insufficient in itself to address all the issues that webinos has identified as vital for multi device web applications to be successful. webinos will actively feedback implementation experience on some of the wider multi device problems to the HTML5 working group to facilitate wider and long term adoption.

W3C DAP

Similarly, the Device APIs and Policy working group, forms one of the principle foundations of webinos – specifically definitions of APIs, but is insufficient in its own right to deliver web applications. webinos will engage to provide direct implementation feedback to this process as it evolves.

W3C Widgets

The 1.0 widget specifications are complete and form one of the principle cornerstones of a web application: how to package and secure a web application. To deliver the innovations webinos envisions, these specifications will have to be enhanced and webinos shall feed into the Widget 2.0 specifications to make this happen.

WAC (BONDI-JIL)

The WAC specifications, which subsume both the BONDI and JIL specifications, shall be used as a basis for delivering the first version of the webinos platform.

And again to clearly differentiate from all of the above

- webinos is an open source implementation with proof-of-concept applications, not only a specification
- webinos is focused on the multi-device and the discovery, communication and security issues that fundamentally move forward the agenda for each of the above organisation.

Where are we – What's next?

webinos is still in its first year. The first batch of use cases, requirements and landscape technology analysis deliveries has been produced. This is just the first step; a necessary one in order to generate the consensus and shared vision to help such a diverse community in a common direction. The next 6 -12 months shall be essential in driving forward the technical platform, at a concrete implementation level.

If this of interest and you would like to follow webinos activities:

- Website: http://www.webinos.org
- Newsletter: http://webinos.org/webinos-newsletter
- Linkedin: webinos group
- twitter: @webinosproject
- Facebook: facebook.com/webinosproject
- email: hello@webinos.org

For those interested in getting involved a new webinos Affiliates program has been created. For details please go to http://www.webinos.org or email join@webinos.org