Introductory Whitepaper

What is webinos

Webinos is a new open source platform which creates the foundations for the next round of innovation in web technology. 30 project partners from across Europe, spanning academic institutions, industry research firms, software firms, handset manufacturers and automotive manufacturers have come together over 3 years to develop this technology.

At the simplest it is three things

- An Application Platform for writing powerful, ‘native like’ web applications to run on many different device types.
- A Connected Everything Protocol, designed from the ground up to support intelligent connected devise that work in the cloud or on local networks.
- A Security Framework: that has been designed with consumer privacy concerns foremost in mind and created to deal with the complexities of applications talking to each other and sharing services
Each of these capabilities has huge value in its own right, and when we take these capabilities in concert, we have a powerful piece of technology which has the potential to revolutionise how we thing of the web.

**Why use webinos**

**Developer**

- Reduce development cost for device development by using Open Source components supporting rich functionality
- Have the same application running securely on all your devices, regardless of the underlying operating system. With a security and privacy model that puts you in control
- Create revolutionary new apps that work across devices and on different device types

**Device Manufacturer**

- Ecosystem benefits: engage with the large developer ecosystem - make your device app capable. increase your device ecosystem by interwoking with complementary devices easily

**End User**

- Have your TV work with your Mobile, with your PC and your Car. Overcome the interoperability problem. Have confidence in an IOT framework which protects your data from prying eyes
Connective devices using the web

Connecting Devices using open, easy to use, evolved web-technologies is the primary objective of the webinos foundation.

webinos has been created for Mobiles, PCs, Tablets, Automotive, TVs, Sensors and more generally any Internet of Things Device.

- By defining a rich suite of JavaScript APIs we expose capabilities of a device to a web developer.
- By defining a new networking paradigm, we allow devices to find each other and discover each other services, no matter how they are connected and who owns them.
- By defining a new security model, we ensure that this rich capability is secure and remains under the control of the end user.

The webinos technology stack has been created with the web developer in mind, and supports JavaScript and HTML applications as first class citizens. The technology is however equally applicable to Native Application Developers.

Code has been released under and Apache 2.0 license for Windows, Linux, OSX and Android platforms already. All specifications have been defined under a Royalty Free IPR framework. The recently created webinos foundation, is founded on Open Governance principles and is accepting membership and board nominations.

Three foundations of webinos

Breaking it down webinos has in detail three distinct but complementary value propositions:

- A web based application framework
- A common inter-device connection technology
- A distribute security architecture

A web based application framework

The debate between native and web based operating systems rages on. But this debate is a false dichotomy; a sideshow that neglects the obvious, which is both have an essential role to play. But by any measure, current or projected the role of web based technologies is increasing over time, not decreasing.

Webinos is another in the family of web based application technologies. It shares many features with ChromeOS, FirefoxOS the former webOS and indeed development technologies such as PhoneGap/Apache Cordova [1]. However there are three important areas where WEBINOS differs:
100% Open and standardised stack

A complete and open web standards stack: within webinos we have as far as possible tried to embrace open protocols on every faceted of the technology stack. A number of the competing technologies in this space use web and HTML5 technology for 80% of their solution, but full interoperability is unfortunately not possible because of proprietary APIs or more often than not proprietary packaging and security models. The end effect of this is that the promise of truly open web based application ecosystem is subverted into singular branded web stores, further extending existing monopolies.

Webinos has taken the contrary view. Open standards, have been embraced at every element of the stack. Where standard have not existed, we have been sure to publish the omitted protocols under unambiguous open terms.

Servers on Devices

The second significant distinction is that webinos is not just about implementing client applications, but implementing server applications. In the webinos model every device, no matter how big or how small, is a potential server. This has two interesting knock on effects. Firstly it allows services, servers and background applications to be run in places you don’t traditionally expect. Secondly it allows device capability (APIs), whether this be access to a file, access to a devices location, or access to local temperature to be remotely accessible to other trusted devices.

A decoupled architecture

Finally, webinos, we believe represents the next innovation in web based architectures. Partly in response to the challenge of delivering the functionality described above and partly as a consequence if natural architectural evolution, a new architecture has been delivered which blurs the traditional roles of web browser and web server. As we state above, it is a desired goal that each device acts as a server in its own right. That single technical requirement has had a profound impact on the architecture. Simply put we have found that by ripping out the application host logic, API extensibility and critical security modules from the browser itself, we can implement a sophisticated web based runtime that is largely independent of the specific browser technology. To prove this point you can find on out developer website examples of how the same application logic can be shown to run seamlessly on Chrome, iPhone (Safari) and Mozilla browsers.

A device connection technology

In addition to acting as a host of web based client (and server) application, webinos provides all the critical technology to allow devices to talk to one another, on open interoperable protocols.

Much is said in the mainstream and technical press about the potential for the Internet of Things, or the Web of Everything, or indeed Connected Devices. At a more concrete level we have multiple innovations, technologies and protocols in subjects as diverse as Connected Cars, Connected TV, eHealth, Home Automation, Smart Cities, Smart Energy. Scratch the surface of each of these and soon you will discover three things.

1) The challenges for all of these diverse subject matters are surprisingly similar.
2) Not surprisingly therefore, there is a lot of "re-inventing the wheel" as each silo tackles this big problems in isolation and largely in ignorance of each other

3) A depressingly large number of the silo solutions are proprietary in nature, encumbered by IPR and as a consequence fail to
   a. Emulate the webs open ethos, with all the positive ecosystem impacts this has
   b. And in many cases fail to interoperate with web base technology and a first class citizen

Again webinos is different. We have looked at all these problems in all different domains and in each case picked, where possible, pre-existing web based technologies to fill the vacuum. Where these technologies do not exist (not that many places) we have patched the holes with elegant lightweight solutions.

What the resulting technology allows us to do is the following:

1) **Describe** in detail what each device is capable of doing in an interoperable way.
2) **Discover** dynamically and in real time what each devices is capable of, both local devices and remote devices.
3) **Route** calls between device over almost any type of networking technology
4) **Call** and finally physically make the call between devices.

This technology is largely web based and certainly allows browsers and servers to integrate with the technology in simple to use ways. But it by no means limited to browsers, and as shall be explained at length in the technical section, Android, iOS, or embedded C++ applications can use this cross device technology just as easily as web browsers can.

**A distributed security model**

The final piece of the puzzle, an essential enabler to deliver both of the above is the security model that makes this all happens. Security is hard. Security on the Internet is very hard. Security that transcends the Internet, every device you own and works across different people would seem well-nigh impossible. Despite the obvious challenges, this is where we within webinos, have invested a large proportion of our efforts, and where the true innovation lies.

The entire model within webinos is based around the concept of a Personal Zone. A Personal Zone, conceptually is simple; it is the logical boundary that sits around you, all your devices, all your services and all you applications. This simple concept is somewhat more challenging to implement as this logical boundary transcends multiple devices, multiple networks, multiple devices types, servers and cloud entity and all of these in connected and disconnected states.

The second challenge is how do you implement this without a single portal. This is easiest to explain by example. Take something simple like calendar sharing. A very simple concept, very valuable and most of us in corporate environments are very used to. Implementing the concept of calendar sharing is quite simple if we have an Exchange Server of Gmail server mediating this access. In this instance the users authentication, the service description and the access controls are all mediated by a single point of control. But would it not be nice to do exactly the same, without handing over all of our data to a single entity. Wouldn't it be nice to be able to share personal data like calendar locally without a server standing in the way.
This is the challenge that webinos addresses. We recognise the power of sharing, but as we extend from calendar sharing, to file sharing, car data sharing, health device sharing, we cannot afford to have all of our data mediated by a single control point.
Webinos Technology Architecture

The ‘webinos technology stack’ when printed is over 1000 pages of PDF documentation. The implementation consists of over 50 separate components, each with its own repository on GitHub. For the interested reader the full detail is available on our developer site https://developer.webinos.org. This introduction does not attempt to replicate this level of detail. In this paper we attempt to summarise each core logical component within webinos and put it in context, so the reader can quickly get and understanding of the how and the why, so that they may better understand the “what” of the platform components.

It is also important to emphasise that webinos has been heavily “componentised”. Each logical component has been written, so that it can be used as a component on its own should the developer require it.

A next generation web application architecture

At one level webinos is comparable to FirefoxOS, ChromeOS or Tizen’s web application framework. All are platforms that allow a developer to create native-like applications, using web based primitives.

The web was not originally designed to do this however; native like applications were not in scope when the original architecture was conceived. The typical browser operates in a very strong sandbox. And this is absolutely necessary: you cannot let “foreign content”, code written on a remote location, to execute privileged functions on a PC or mobile. There exists no trusted model or permission model to allow this to happen.

Necessarily therefore all web based application platforms extend the browser with the following functional components:

1. **APIs**: a suite of rich JavaScript APIs that allow the developer to access native-like capabilities.
2. **A policy framework**, that protects a user from an application abusing these same powerful APIs.
3. **A packaging framework**, which stops applications being compromised and gives each application a strong identity which forms the foundations of trust.

All web based frameworks must provide these capabilities. They all differ subtly in the level of interoperability they provide on each of these functions.

On this subject webinos has been clear: to insist on open standards for web applications.
The architecture of ChromeOS, FirefoxOS and Tizen are remarkably similar.

All components exist, but are bound together into a single runtime.

This is the second way webinos differs from these alternative architectures; webinos supports a decoupled architecture, where the traditional browser rendering components are logically and physically separated from the “application engine”.

This has several benefits:

1) Allows for easy porting of runtime and applications: if you visit the developer site you can see evidence for this in the fact that webinos has been ported for example to both FirefoxOS and ChromeOS with very little effort.

2) Better support for “services” – long lasting applications that have a lifecycle longer than a web session.

3) Better support for IOT services, where another device wants to use an API (see later on cross device protocols)

4) Better security: components are better sandboxed against each other.

This architecture, we believe, is the way we should be looking at next generation web browser architectures.
A rich suite of JavaScript APIs

A developer is restricted by the functions that he has access to. For a platform to be a success it is essential that a rich suite of APIs are supported out the box.

Moreover, webinos has a clear remit to be relevant to several different device categories. PC and mobile are well understood and APIs fairly common place. Cars, TVs and IOT devices are traditionally less well supported.

Finally, the platform needs to be future proofed; that means it is essential that developer can extend the platform with their own APIs.

In all webinos defines 31 API. 24 of these APIs are brand new and created to help a developer extract full value from the platform. For example on the IOT platform a GenericSensor and Generic Actuator API have been defined. For Vehicle a Vehicle API has been created to access in vehicle resources, this can be supplemented with GenericSensor and Generic Actuator to give extended platform capabilities. The TV platform has quite a few new APIs created for it to make it as easy as possible for a developer to create rich media, TV integrated applications including TV Control API, Media Play API and Media Content API.

In addition to these 24 API, 7 APIs are defined by proxy, linking in the main to pre-existing W3C APIs.

The full list of APIs follows:

<table>
<thead>
<tr>
<th>Specification</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Generic Actuator API</td>
<td>The webinos Generic Actuator API provides applications with an API to control actuators.</td>
</tr>
<tr>
<td>App2App Messaging API</td>
<td>Interface for exchanging messages between applications, locally and/or remotely, using named channels.</td>
</tr>
<tr>
<td>AppState Synchronisation API</td>
<td>Interface to enable and manage application synchronisation.</td>
</tr>
<tr>
<td>Authentication API</td>
<td>Provides information to applications about the current authentication status of users</td>
</tr>
<tr>
<td>Contacts API</td>
<td>This API provides access to a user unified address book.</td>
</tr>
</tbody>
</table>

Technology

WebIDL: each JavaScript API is fully specified by a machine readable W3C specification WebIDL.

Feature-URI: each API is identified by a unique URI. This allows APIs to be defined on a distributed basis.
<table>
<thead>
<tr>
<th>Specification</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context API</td>
<td>Context API client side reference.</td>
</tr>
<tr>
<td>Device Interaction API</td>
<td>The webinos The Device Interaction API allow you to control various components in the device.</td>
</tr>
<tr>
<td>Device Status API</td>
<td>The webinos Device Status API lets you get information about various “aspects” of a device.</td>
</tr>
<tr>
<td>The Event Handling API</td>
<td>The Webinos Event Handling API provides means to exchange data in terms of events among addressable entities (e.g., applications, services), either locally or remotely.</td>
</tr>
<tr>
<td>AppLauncher API</td>
<td>Allows activation of webinos applications installed locally on the device.</td>
</tr>
<tr>
<td>MediaContent API</td>
<td>This API provides access to multimedia contents and related information.</td>
</tr>
<tr>
<td>MediaPlay API</td>
<td>The Webinos MediaPlay API can be used to invoke playback of media files in external media player software, instead of using playback in the widget/browser runtime.</td>
</tr>
<tr>
<td>Navigation API</td>
<td>The webinos Navigation API provides mechanism to interact with on-board navigation software.</td>
</tr>
<tr>
<td>NFC API</td>
<td>Near Field Communication (NFC) support.</td>
</tr>
<tr>
<td>The Web Notifications API</td>
<td>The webinos Web Notifications specification provides an API to display notifications to alert users outside the context of a web page.</td>
</tr>
<tr>
<td>Payment API</td>
<td>Interface for Payment functions.</td>
</tr>
<tr>
<td>The Remote UI API</td>
<td>The webinos remoteUI API provides means to create and access a UI on a remote device using DOM manipulation commands.</td>
</tr>
<tr>
<td>Secure Element API</td>
<td>The Webinos Secure Element API provides web applications with an API to access secure elements.</td>
</tr>
<tr>
<td>The Generic Sensor API</td>
<td>The Webinos Generic Sensor API provides Web Applications with an API to access data from sensors in the device or in another device.</td>
</tr>
<tr>
<td>Discovery API</td>
<td>The Webinos Discovery API provides web applications with an API to discover services without any previous knowledge of the service.</td>
</tr>
<tr>
<td>TV Control API</td>
<td>Interface for TV control and management.</td>
</tr>
<tr>
<td>Vehicle API</td>
<td>The webinos vehicle API provides access to specific vehicle data.</td>
</tr>
</tbody>
</table>
### Specification | Summary
--- | ---
**Webinos core interface** | Common interface from which all Webinos APIs can be accessed and it also includes information about the webinos personal zone

**webinos Widget API** | Webinos specific extensions to [W3C Widget Interface](http://www.w3.org/2000/svg)

---

**FEATURE**: webinos creates a wide range of new API that expose the power of native platforms whether they be TV, Car or IOT

---

**Refered APIs**

<table>
<thead>
<tr>
<th>Specification</th>
<th>Summary</th>
<th>Inheritance of webinos Service interface</th>
<th>Feature URI</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>The W3C DeviceOrientation Event specification</strong></td>
<td>This specification defines several new DOM event types that provide information about the physical orientation and motion of a hosting device.</td>
<td><code>interface DeviceOrientat ion : Service { ... };</code></td>
<td><a href="http://webinos.org/api/w3c/deviceorientation">http://webinos.org/api/w3c/deviceorientation</a></td>
</tr>
<tr>
<td><strong>The W3C File API</strong></td>
<td>This specification provides an API for representing file objects in web applications, as well as</td>
<td>N/A</td>
<td><a href="http://webinos.org/api/w3c/file">http://webinos.org/api/w3c/file</a>, <a href="http://webinos.org/api/w3c/file/read">http://webinos.org/api/w3c/file/read</a> (file read only)</td>
</tr>
<tr>
<td>Specification</td>
<td>Summary</td>
<td>Inheritance of webinos Service interface</td>
<td>Feature URI</td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
<td>----------------------------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| **The W3C File API: Writer** | This specification defines an API for writing to files from web applications. This API is designed to be used in conjunction with, and depends on definitions in, other APIs and elements on the web platform such as the W3C File API. | N/A | http://webinos.org/api/w3c/file (all file APIs)  
http://webinos.org/api/w3c/file/write (file write only) |
| **The W3C File API: Directories and System** | This specification defines an API to navigate file system hierarchies, and defines a means by which a user agent may expose sandboxed sections of a user local filesystem to | | http://webinos.org/api/w3c/file (all file APIs) |

interface LocalFileSystem : Service {
... 
};

interface LocalFileSystemSync : Service {
...
};
<table>
<thead>
<tr>
<th>Specification</th>
<th>Summary</th>
<th>Inheritance of webinos Service interface</th>
<th>Feature URI</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>The W3C Geolocation API</strong></td>
<td>This specification defines an API that provides scripted access to geographical location information associated with the hosting device.</td>
<td>interface Geolocation : Service { ... };</td>
<td><a href="http://webinos.org/api/w3c/geolocation">http://webinos.org/api/w3c/geolocation</a></td>
</tr>
<tr>
<td><strong>The W3C Media Capture and Streams API</strong></td>
<td>This specification defines an API that provides access to the audio, image and video capture capabilities of the device.</td>
<td>interface NavigatorUserMedia : Service { ... };</td>
<td><a href="http://webinos.org/api/w3c/mediastream">http://webinos.org/api/w3c/mediastream</a></td>
</tr>
<tr>
<td><strong>The W3C WebRTC API</strong></td>
<td>This specification defines an API that</td>
<td>N/A</td>
<td><a href="http://webinos.org/api/w3c/webrtc">http://webinos.org/api/w3c/webrtc</a></td>
</tr>
<tr>
<td>Specification</td>
<td>Summary</td>
<td>Inheritance of webinos Service interface</td>
<td>Feature URI</td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
<td>------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>allows to establish peer-to-peer connections between browsers, and transmit media streams and data over these connections.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FEATURE: embraces existing standards and implements state of the art features in highly portable ways**

**Implementing APIs**

Specifying 31 APIs is one thing, implementing them is another challenge, especially when we target so many different operating systems.

This is an implementation detail of our open source reference, and not mandated by the specification, but for the webinos platform these are generally done by either

1) Implementing in JavaScript, and/or on top of existing NPM modules that define the required functionality
2) OR by binding to native code by the V8 supported native bindings.

This has proven to be a very effective way of implementing highly portable API

**FEATURE: new APIs can be added by third party developers using their own namespace**
Application Packaging and Basic Policy

The final element to securely delivering web based applications to multiple device types is ensuring the applications (and the user) can be trusted. This requires to complementary features.

1) A technology to package the application. This technology must support two strong requirements
   a. Application integrity: we need assurances that the application cannot be tampered with
   b. Application provenance: we need some mechanism to help identify where the application came from. This is especially important for distribution models, such as webinos, which do not rely on a curated AppStore.

2) A technology to implement policy. There are two aspects to policy for web applications:
   a. Least privilege policy – per application: which is a functional perimeter defined by an application, which forms the basis of a user accepting the installation of an application and can be used to police a functional sandbox around that application executing
   b. User policy – per device: which is a user specified set of rules which determine which application or classes of application have rights to execute which APIs.

FEATURE: webinos support 100% open application packaging, security and policy mechanisms

Personal Zones: bringing your devices together

All the technology elements covered so far related to the problem of applications running in isolation; a single app on a single device. In webinos we call this “running in virgin mode”. It relates to a runtime that has been installed on a device, but this device has no formal relationship with any other device.

The use case we are aiming at however is more complex than this. We want devices to interwork, to allow a mobile for example to control the stereo in a car. To do this we need to enhance the basic web runtime with a lot of new technology.

However, before we start looking at the technology, we need to think about the user experience. How will a user manage all their devices, and perhaps more importantly how will he/she secure these devices from unwanted attachments.

It was by thinking through this problem of user experience and device management that we came up with the user-centric concept of a personal zone. A personal zone, as we will see in a minute has a formal

Technology

W3C Digital Certificates: a W3C standard for signing data, and in this case this data is a packaged application

W3C Widget: a full specification for application packaging

OASIS XACML: a flexible XML based framework for creating policies

Personal Zone – User definition

A virtual network that connects all your devices, no matter the physical connection

A security perimeter or logical firewall, two which you grant access

A pool of resources (data and services) which belong to you
technical definition. But the primary reason for the creation of the “Personal Zone” is to introduce a concept into the end users vocabulary that will allow them to navigate this concept of getting devices working with each other.

The above schematic demonstrates how a user’s devices are logically grouped together and give a flavour or how, when such a logical zone is defined, we can start thinking about sharing between zones.

As we start to think through the problem of how to make this concept real, we face a number of challenges:

- What is it that binds these devices together, so that can act as one?
- How do we communicate between these devices in a robust way?
- How do implement the security perimeter? How do all device coordinate on their security decisions?

What we find as we think through these issues, is there needs to be an authority. If many devices are to act within a logical network, there needs to be a cloud based entity that can act as a coordinating agent and a messaging hub or post box for inter device messages.

This entity we call the Personal Zone Hub, and to all intents and purposes is a cloud based agent that represents an end user.

One each devices there is a mirror of this entity a thing called a Personal Zone Proxy. We will see more of this later.
The architecture in principle is not too dissimilar to an email or IM architectures, where there is an entity that resides in the cloud (the server account) to which multiple clients (email clients) can connect. It is also not too dissimilar to the Google Chrome architecture, where multiple browsers when logged in can synchronise bookmarks, or user names and passwords (identity).

The defining technical characteristics of a personal zone hub are therefore, it must be addressable and addressable by a unique URI. This has the implied requirement that URI are distributed, i.e., there is no single PZH authority. The second key requirement is that each PZH is a certificate authority in its own right and has the capability of issuing new certificates for new devices.

**FEATURE: webinos allows you to share data and services between your devices**

**Enrolment: adding a new device to a zone**

The concept of me as an end user adding a new device to my personal zone now has a very formal definition:

1. Issue a new certificate for a device: based upon the uniquely addressable URI for that device
2. Enrol the device by security embedding this certificate on that device
When this process is complete the new device is now part of the Personal Zone.

Once enrolled, the next step is to establish a session between PZP and PZH. In webinos a session is simply a mutually authenticated TLS connection, which is now of course possible because client certificates have been issued to all relevant devices.

Raw TLS session are preferred over HTTP, or indeed websockets, because they are the simplest form of mutually authenticated secure session possible. They have the minimum possible overhead (in terms of negotiation/session establishment) and are wholly symmetric, unlike HTTP. As we shall see later these properties will be essential for some of the use cases we have in mind.

The basic foundation of our virtual network is now complete.

- We have multiple devices connected to a central hub
- We have secure mutually authenticated connections between each device and the hub
We have a unique routable address for each device so that it can be found by the hub even if sitting behind a firewall.

FEATURE: all your devices can securely address each other, working over firewalls

Peer to Peer: connecting devices without the internet

The protocols defined so far are perfect for connecting devices when they all have access to the public internet. But how likely is this in real world deployments?

- When I’m sat in the car with my mobile phone, it should not be necessary to route my command via an intermittent internet connection
- If I want to stream a video from my set top box to my tablet in my home, there is no reason why these packets need to go via the cloud.

Furthermore, there are number of scenarios we shall look at when we consider Internet of Things devices where physical connections may not even be over IP, eg. Bluetooth serial port connections or RF 868 connections.

For all these reasons and more there are strong reasons to support peer to peer cross device connections. It should, in other words, be possible to connect a PZP to a PZP without connecting to the cloud.

The choice of the distributed PKI architecture is critical to making this happen. Because trusted certificates have been issued to devices we now find making a peer to peer connection is actually not that different to making a PZH-PZP connection. We are able to make mutually attested virtual connections over TLS, irrespective of the physical routing of the connectivity. This is a very powerful concept, that opens the door for many compelling cross device use cases.
Connecting People and their devices: Inter Zone Communication

So far, using a Certificate Authority, and enrolment protocol, and mutually authenticated TLS connections we can see how to create sessions trusted sessions between a users devices, no matter whether the devices are connected over the internet via a hub, or peer to peer using some local connection schema.

We have the foundations of a trusted virtual network for addressing and communicating between devices; we can communicate within the Personal Zone.

What happens therefore when we need to communicate between zones? What happens if John’s mobile phone wants to connect to Peter’s TV?

Many of the primitives remain the same. Meshes of mutually authenticated TLS connections form the basis of a virtual network over which devices talk to each other in a trusted manner. What differs in this instance is how the certificates become trusted.
When Peter’s mobile connects to Peter’s TV, we have trust in that connection because Peter’s PZH issued both of the certificates, therefore they have a common root of trust.

If Peter’s mobile tries to connect to John’s TV no such shared root of trust exists. Therefore we have to bootstrap the required trust in a different way.

Technically, when we grant access to a service across zones in webinos there are two trust hurdles to overcome:

1) As John do I trust an incoming connecting request from Peter? Is it really Peter? Or someone pretending to be Peter?
2) Assuming it is Peter, do I actually want him to access my service?

The first is an issue of identity, the second is an issue of permission. For the moment we shall only discuss issues of identity, we will come back to permissioning later when we look at advance policy.

To overcome the issue of identity in webinos, we have introduced certificate exchange protocols.

The issue of trust is largely dealt with out of band: do I trust the person next to me, do I trust a local Bluetooth connection, do I trust an incoming email is from the person I know, do I trust someone’s facebook account has not been compromised. However, if we assume a number of such trust mechanisms exists webinos defines concrete certificate exchange protocols to allow John to exchange certificates with Peter. These certificates are saved at the PZH and synchronised to all connected PZPs. It is by this mechanism that Peter is able to make a trusted connection with John between any two devices, both over the internet or peer to peer, providing we have bootstrapped a certificate exchange process out of band.

**FEATURE: share your services with friends, in a secure way — both over the cloud and peer to peer**
Discovering Services

Now we have established how to connect devices we need do something interesting over these connections. The first thing to do is to ask a device what it’s capable of: to perform a basic service discovery.

Within webinos, messages are passed over the TLS connection in JSON format. This makes them very easy to work with for the JavaScript developer.

One of the simplest of such messages to be passed over the network is the query: what services do you support.

```json
{
  "status" : "findServices",
  "message" : {
    "id" : ""
  }
}
```

A device which receives such a request responds with a simple description of the devices that are supported. We have already seen how the type services are described: Feature-URI (API). Where each Feature URI will map to a formal JavaScript interface in turn described by WebIDL descriptor.

```json
{
  "id" : "91c78f03e532196234db6ca06dd93d96",
  "api" : "http://webinos.org/api/sensors/energy",
  "displayName" : "electricity - average (kWh)",
  "description" : "average energy [E1l]",
  "serviceAddress" : "tobyealden@pzh.webinos.org/TE-Dev"
},
{
  "id" : "49601108250f62971b71b186e8565b06",
  "api" : "http://webinos.org/api/sensors/energy",
  "displayName" : "electricity - cumulative (kWh)",
  "description" : "cumulative energy [E1l]",
  "serviceAddress" : "tobyealden@pzh.webinos.org/TE-Dev"
},
{
  "id" : "fd28026bc8520e0a8948a3b78ce1e60",
  "api" : "http://webinos.org/api/sensors/energy",
  "displayName" : "electricity - peak (kWh)",
  "description" : "peak energy [E1l]",
  "serviceAddress" : "tobyealden@pzh.webinos.org/TE-Dev"
}
```

These descriptors are complemented with some meta data, including origin of service, friendly name and most importantly serviceID. Essential because of course for every defined API there may be many individual instances of it hosted on an end device.
**FEATURE**: a web friendly easy to use way of sharing JavaScript services between devices

**Invoking Services**

Finally, once discovered, the final element required to make use of this service is the ability to remotely invoke it. This technology is not new to webinos, but simply reused by webinos and it is the JSON-RPC protocol for describing methods, parameter and return types.

With this component in place we are now able to invoke any arbitrary JavaScript API from a remote device is a secure way.

It is worth at this stage taking a step back and understanding how this combination of technology simplifies the problem of one device calling functions on another. In traditional web programming this is of course a common scenario. A client web script will for example call a server to get a list of books for example that it is to display. In a traditional web programming model, this request is usually intercepted within a JavaScript function on the client. Client and server have to pre agree ad protocol for making remote requests and this
JavaScript function then has to manually (with developer written code) encapsulate this request into this ad hoc protocol. This request is then issued via XMLHttpRequest() where it is then intercepted by the server. The server then has to unpack this request, and a call is made into the database. When the database returns the process starts in reverse: a response protocol is defined and the server code must manually format this response. The response is issued back to the client where again it has to be unpacked before presentation on the client.

In summary then for each client server requisition response we need to define two new arbitrary protocols and write four functions (two client and two server) to pack and unpack requests.

In the webinos world with the use of service discovery and JSON-RPC, all of this complexity disappears. ON the client we call GetBooks() and GetBooks() is implemented on the service. The infrastructure supports the marshalling and un-marshalling of the requisite JSON-RPC packets reducing not only the custom code that needs to be implemented but also reducing design work on custom protocols.

FEATURE: use a service on another device as easily and writing a JavaScript function locally
Advanced policy

The issue of basic policy has already been covered. This is the policy that is responsible for determining which application has rights to which API in the context of a single device.

The innovative use of webinos for enabling cross device service requests increases the complexity of the decision space by four dimensions.

A permit deny is no longer simply an issue of Can Application X access service Y.

It is now a question Can Application A on Device B owned by Person C access API D hosted on Device E.

This is a significant challenge and a significant increase in complexity. Fortunately our strategy of re-using XACML as a policy technology has proven particularly prescient in this instance, in that it has proven remarkably capable of modelling these use cases.

Describing the rules is just one facet of the extensions required, we also need to address policy sync and policy editing.

These are two new essential element of the webinos architecture:

- Policy synchronisation: a protocol that allows policy changes made on one device to propagate to all devices within the same personal zone
- Policy editing: is a new UI framework and indeed new UI metaphor to help an end user interactively and intelligently manage their policy.
**FEATURE:** assert total control over who can see your services on your devices

---

**Internet of Things Support**

Each of the technologies discussed above has use in its own right. When taken in concert we have an end to end framework for not only running applications on multiple device types, but also a framework that will allow one device to securely and interoperably discover and user each other services over both the cloud and over local networks.

This technology was designed for and is applicable to a wide range of device types including PCs, mobile TVs, and Cars.

But what about the Internet of Things? This a hugely important and growing area. Wouldn’t it be nice if we could apply some of these concepts to this domain?

Well it turns out we can. Moreover some of the unique architectural elements of webinos make it extremely flexible. Two features in particular make it stand out, against similar IOT protocols such as AllSeen.

1) The webinos protocols work both for local networks and cloud/internet discovery. A common protocol can be used for both.

2) Legacy mode: the API integration model discussed below has proven especially powerful of integrating onto pre-existing IOT devices.

3) Security model: the webinos security models is very flexible is particularly useful for IOT scenarios.
The core webinos stack based on a node.js instance typically is around 2mb in size. Whilst this fits on a mobile phone, it is too large to be deployed onto an IOT device.

The webinos protocols, or a subset of them can be defined so as to be more amenable to IOT deployments (sub 64k) and indeed a protocol subset has been defined within the microPZP project. From an implementation perspective this is still work in progress.

The majority of IOT deployments made so far have been done using what we call the webinos driver model.

This model is elegantly simple. The capabilities of the IOT device are encapsulated by one or more JavaScript Interface. More often than not GenericSensor and GenericActuator are more than adequate for this.

This JavaScript interface, like normal APIs are then either implemented using existing NPM modules (e.g serial port access) or using Native-JavaScript interfaces. This API implementation code takes responsibility for access the base capability of the attached IOT device.

This capability however, once abstracted behind WebIDL described interface and accessible via JSON-RPC is discovered, accessible and protectable (by policy) just like a normal webinos service.

**FEATURE:** support Internet of Things devices interoperability, without re-flashing software
Four Demonstration Gateways

The technology defined in webinos is highly versatile with many potential uses. In order to demonstrate this we have focussed in on four demonstration gateways that show concretely how both the core technology and webinos based applications can be applied in four diverse scenarios, impacting four very real industries.

webinosTV – Connected TVs

The webinosTV solution is an Open Source embodiment of the Connected TV vision. The core platform technology has the following features:

- Advanced web complaint runtime, with hardware accelerated 2D and 3D graphics and support for a wide array of video codecs
- A suite of TV specific hardware extensions, which allow web based technologies to integrate with TV capability: specifically
  - TV API: which abstracts the interaction with broadcast media? There are a number of potential implementations of this, but the main supported one uses SAT>IP so that terrestrial and satellite content can be consumed on any device
  - File API: basic API for low level file access, both locally, on device and remotely: e.g. a NAS drive
  - Media Content API: can be seen a logical extension of the file API but optimised for handling large media libraries, media meta data and doing fast searches.
  - Event API and Application Messaging: provides the capabilities to allow a device such as phone, to send abstract messages to other devices like a TV
  - Media Play API: the media play API is really just HTML media play tags, with one subtle but important difference: using the webinos remoting capability playback can be coordinated and remote controlled from many different devices,
- Appstore model: a TV can be enhanced quickly with an integrated Application Store, using the defined application packaging mechanisms

- Sharing model: finally webinos supports the mechanism for media to be shared between different and different devices, whether these devices sit on the same local network or across the web. Not only can simple media be shared but services, e.g. playback of PVR record services can be shared as easily as a file can.

When we put all these technology components in place we have a very powerful framework for rendering a whole array of connected TV scenarios. We can deliver multiscreen playback and control (ala UPnP) but in the webinos model this can work across cloud as well as just in the local network, it is also more secure, and it makes browser and web interactions very easy to do.

The webinos remoting is the ideal platform for remote control and second screen solutions the frame work for the secure cross device communications is well defined and highly extensible meaning that applications to remote control TV playback, controlling TV rendered 3D games using phone gestures and movement, doing cross device advertising or cross device interactive games can all use the same framework.
Importantly the technology has been defined in such a way, that it is entirely compatible with an indeed adds significant value to existing technologies in the connects TV space such as UPnP and DLNA.

**webinos - Health Gateway**

Health devices, by their very nature have access to extremely private data. Connected health devices in contrast have the potential to offer enormous cost savings to primary care suppliers. This tension between consumer data privacy concerns and the cost saving potential of pervasive all seeing health devices makes it an ideal test bed for some of the webinos principles.

We have taken the example of a midwife providing care to a mother and a new born baby to test the technology with. The relationship between a midwife and new mother is an intimate one that entails a lot of monitoring, a lot of travel and inevitably the sharing of a lot of private data.

The challenge is to produce a system that is easy to use, can enable unobtrusive remote monitoring, will work with many different devices and device types, but at all times respects the mothers right to privacy.

The solution is not intended to replace the Midwife, but to make their life easier, and to provide a facility for the mother to get expert advice on things that may be worrying thing by granting third parties real-time and historic access to some very private data.

Remote health monitoring systems are not necessarily new, nor is webinos unique in providing them. The end to end solution however has certain properties which are quite unique, and positively disruptive to the existing industry.
• Wide device support: unlike many remote health systems, the end monitoring device is not locked into the end to end IT system. The webinos health hub demonstrates interoperability between devices from different manufacturers, supporting different connectivity protocols (ANT+, Bluetooth and RF) and different application protocols. None of these devices was reflashed in the making of this demo! Support was created solely through the creation of a simple web driver. Economically this model has to critical impacts
  o Reduced cost of device porting, through the use of pre-existing open source driver models
  o Reduced supplier lock in, which again both reduces costs, increases choice, and increases potential for innovation
• Real time and historical data: this model supports both real time attachment and the storing and sharing of historical data record.
• Data ownership: in the webinos model data never leaves control of the end user. The data always resides on the PZH host of the users choosing. Specifically in this instance a Raspberry PI. An end users only ever “grants access” to this data. They never lose control
• Data sharing: this model of data ownership, helps facilitate novel model of data sharing. If a patient changes health supplier, wants to grant GP and hospital access, or indeed wanted informal Peer support from a breast feeding clinic, the patient can very simply allow this to happen.

Fundamentally, by introducing interoperability and consumer centric security it is possible to innovate with the supply of services in ways not possible on traditional architectures.

Technically all data is mediate through the GenericSensor() webinos API, and historical data is aggregated with the webinos database API.
webinos – Home Automation Gateway

The Home Automation Gateway although different in industry and application is technically almost identical to the Health Hub.

A large number of different devices can be supported. These devices can be bought off the internet and adapted, generally with just a few hours work, to be entirely interoperable with the webinos control system.

In the world of home automation where the list of standards and protocols boggles the mind, a hub technology that can translate between different protocols is exceptionally useful.

Also like the health scenario the facility control and share access to both real time and historical data is built in.

Controls of lights and switches and media units is of course supported. But at a commercial level it is the interaction with energy systems, heating systems and plumbing that things get interesting. If we compare to existing solutions in this area we can see where it is disruptive.

- **Smart metering**: smart metering technology is being deployed in many places, however the vast majority of these solutions have end meters tied to the service provider. This creates a significant challenge for the consumer that wants to swap supplier, quickly and regularly. A behaviour that is essential if you want a competitive market. The webinos model supports both the interoperability between meter device and metering services. It also supports the security and sharing model to swap supplier. As such it is an important technology to consider for such deployments.

- **Data Mashups and interoperability**: a traditional building may have independent suppliers for Security Systems, Fire systems and Heating control systems. But many of these systems require, or could make good use of the same data. The heating system and the security system both want to know if the window is open. The webinos model allows the sensors to be shared between “services” both reducing costs and increasing the potential to innovate on top of mashed-up data.

- **Smart Sharing**: if we are to consider the aggregation of radiator temperatures, boiler monitors, gas meter reading and window open sensors, we have a rich data stream that several different actors could be interested in. For example
  - Device manufactures could monitor for indication of early failure
  - Energy suppliers could examine in order to make a competitive bid on supply
  - Insurance companies/Service engineers could monitor to should maintenance and help identify early failure
  - Building contractors: could examine to recommend insulation works
  - Eco-consultants could examine to recommend new optimised “control algorithms” or alternative complementary supply, eg solar, geothermal
The model again is positively disruptive in enabling and ecosystem where, with end user consent, data can be shared with a rich array of diverse service suppliers. Thus amortising the cost of sensor and infrastructure investment over many services.

Finally we should also point out that a generic dashboard has been created for the home automation solution that makes it very easy for and end user to construct their own data visualisations and controls systems.

**webinos Automotive – Connected Car**

Finally, webinos automotive implements three distinct but complementary use cases; each of which clearly demonstrates the value of a truly connected car.

**In car head unit**

The first use case is intended to show how web technologies can be used to implement all the core functions typically associated with an in car head unit. This is the subsystem BMW has prototyped. By adding the Vehicle API to the existing suite of webinos APIs we are able to implement all the key functions expected of a head unit in a neatly integrated, easy to developer for, highly adaptable web framework.

This therefore includes
- Core vehicle dashboard showing speed, RPM, emergency indicators etc.
- Navigation system: building on GeoLocation and external navigation services a complete in car navigation system can be built
- Media: by adding in the “home media” components developed for the TV hub we can reproduce and comprehensive highly integrated media experience in the car.

As a technology stack, in and of itself this solution is of interested to car manufacturers and suppliers in that it reduces the cost and increases the flexibility of supply of in car head unit components

**Inter car communications hub**

The second use case builds on the in car communications element. Phones and laptops used by passengers within the car become extended elements of the in car experience. Not only does this complement the in car media experience, but can be extremely useful for a passenger helping with navigation. A destination can be looked up on a phone an pushed to the in car system very simply

This experience is made possible by the local peer to peer networking model facilitated be webinos. It is important as it overcomes the “interoperability” problems facing many in car connected experiences.

**Remote vehicle inspection**

The final scenario and in some senses the most interesting is the remote inspection use case. Again this reflects the health and home automation scenarios. The interesting and disruptive commercial possibilities emerge from a combination of the interoperability and consumer controlled data access that is at the heart of webinos.

Location trackers in cars are common place and have been around for a long time. Location trackers that interfaces the live vehicle diagnostics are less common but growing. Solutions however that integrate all this information (and more) and make it available for and end user to share with whoever they chose to, do not exist on the market. Like smart meters, location trackers tend to be locked to a single service, and the data is never controlled by the end user.
If however we put this data in control of the end user, what services can we enable?

- Driver monitoring: for learner drivers or improving drivers, behaviours can be analysed and recommendations made for improvements
- Eco monitoring: how you drive affects fuel consumption. Consumer volunteered advisor services can be given
- Remote vehicle diagnostics: can manufactures or garage service provides, through access to live and historical services can early diagnose issues, and reduce costs to themselves and the end user. A system where supplier can be changed, by the user is an essential technical component that will ensure a competitive marketplace for such services instead of monopoly lock in
- Emergency breakdown: emergency breakdown services similarly give provide a better end user service with access to remote diagnostics. Again service switching without changing hardware is essential
- Insurance: finally we have the insurance model. Black box based insurance is becoming more popular. But these models are somewhat draconian in term of the conditions of data ownership. A model where the consumer offers their data, for restricted time periods, against which insurance companies can make competitive bids, supports the same essential use case but inverts the current commercial relationship to be far more consumer friendly

A webinos enabled IOT model where the end users controls their data, we believe will be critical to stimulating a rich ecosystem of devices and services, which fluidly interoperate with each other in the same way the internet does.

The costs of sensor and equipment will only be shareable across this wide array of services, if the interoperability and security issues are addressed
Webinos Key Design Principles

The development of webinos has been driven by several key principles that have determined the size and the shape the current project. These same principles have been accepted by the webinos foundation and therefore will continue to drive the evolution of the technology.

**Don’t Invent (for the sake of it)**

Reuse of existing technology is an essential criteria of webinos development. There is a lot of good stuff already out there. We only create new technologies where there is an obvious gap.

**Open Source**

All webinos assets are published as Open Source assets. Open Source is a drive force of all Internet technologies: it makes technical collaborative discussion on new protocols and user experiences very concrete very quickly, thus speeding up the industry collaboration process. It also constitutes a substantial engineering asset in its own right which can reduce the development and maintenance costs of the key technology adopters.

**Open Governance**

An Open Source asset does not necessarily mean anyone can influence it. The nature of Open in the context of Open Source is heavily debated. Indeed as part of the webinos project we undertook some extensive research into the nature of Open Governance. [http://www.visionmobile.com/product/open-governance-index/](http://www.visionmobile.com/product/open-governance-index/)

In its day to day operations the webinos foundation endeavours to follow the principle of open governance, being transparent in its activities and decisions, and founded upon meritocratic principles.

**Open Standards**

A significant cost to technology adopters and barrier to entry to new market entrants is patent licensing costs. The success of the internet and internet technologies is largely attributable to the prescient position held by W3C, IETF and other bodies, which insist on Royalty Free standards.

We have attempted to emulate this within webinos. Any component which has an implied external interface, that is needed for interoperability is supported by a published open standard. All webinos contributors commit to making their contribution available under Royalty Free IPR terms which are entirely compatible with the W3C patent policy.

**Consumer Data Sovereignty**

The rights and the end consumer, and their ability to asset explicit control over access to their devices, their services and their data is a the heart of the webinos design principles. This strategy is and explicitly and deliberate antidote to current trends within consumer clouds services. We also see that technology and policy here will be essential to the real-world widespread take off of the Internet of Things. Consumers are rightly concerned about who will have access to this pervasive and sensitive information.

**Distributed Architectures – No single point of control**

The internet was designed as a decentralised system and based on a distributed architecture. Technologies which look to move forward the state of the art in the internet domain must reflect these design principles.
Great care has been taken to ensure that webinos services and webinos APIs can be implemented on an entirely decentralised architecture. This is manifest in simple statements such as:

- An end user can choose to host their webinos services on any service provider on any internet domain
- A developer can choose to extend webinos with their own APIs, named and specified on their own domain.

**Appstore Agnostic**

As a further example of the decentralised principle is the webinos strategy with respect to AppStores. Since the advent of the iPhone there has been a disturbing trend of binding new Operating Systems and Application Environments to singular AppStores. The notion of a generalised computing engine (an Operating System) has been perverted into being a storefront, for an implied monopoly of content. Android, iPhone, Windows Mobile and RIM all support this model. Disturbingly the new web based Oss ChromeOS and FirefoxOS all have the notion of an embedded appstore.

Webinos has been designed to counter this. There is not one to one mapping between Application Engine and Appstore. Admittedly technically this can be difficult to achieve; life is so much easier if you have a single distribution point Also it can be challenging for an end user, discovery is more complex. However, we believe that the Internet principles are essential here if we want open web application ecosystem, and therefore upload this principle in the design of our application security model.

**Componentisation and Loose Coupling**

The current webinos architecture consist of 39 discrete modules. Over time this will grow. It is a design criteria to produce assets that can be re used in context other than pure webinos. We have a number of success stories here we can already point to in terms of component technology adoption

**Legacy Driver Model**

Finally, and this has particular relevance to the Internet of Things deployment model, webinos defines a large technology stack that needs to be installed on end devices to work. We must accept that installing on a device is not always achievable. Therefore where possible we attempt to created models where existing devices can be made to interwork with webinos, without re flashing or reinstalling core software. You can see evidence of this in the webinos device database
Webinos Ecosystem

Webinos technology is highly innovative. In common with many innovative technologies it is also potentially disruptive. Being open source it is especially so. It has the potential to commoditise the product offering of many incumbent suppliers in key industries. But the other side of that same coin it has the potential to revolutionise the value chains of same industries, lowering the barrier to entry for new suppliers, and increasing the interoperability with all the ecosystem and the innovation benefits this brings.

Webinos technology is also, necessarily, somewhat complex. As it facilitates communication between different device and different device types, it faces not only technical challenges, but the challenge of working across industry sectors.

It is essential therefore that we consider key strategies for technology adoption. Some of the more important ones are summarised below.

Hosted Services

The webinos model requires a server component. Like email or instant messaging someone needs to host a server to make the end to end service real. How do we bootstrap such an ecosystem? Who will be motivated to make the necessary infrastructure investments?

Three strategies have been pursued to help this ecosystem bootstrap itself:

1) Home Appliance Model: the four hub strategy described above is the culmination of a strategy where an end user installs the PZH in their own home. The PZH instead of being a cloud service, becomes a physical box, installed in the house. Albeit one that is web routable. It represents an end user productisation of the PZH concept, and one entirely compatible with the vision we embrace of a user taking control of their own data. These devices are currently productised on a Raspberry Pi, as an Open Source home build project. However, this principle could be embraced on other physical platform (e.g. Routers and Set top boxes) with a more commercial delivery model.

2) Amazon Images: as well as creating software images for end user device installation, our developer site also supports software images (e.g. Amazon) for PZH instances. Targeted at the more sophisticated end users or company, it allows them to easily “host their own” in much the same way as they could host their own email server.

3) PZH Farm: the final strategy assumes a service provide will take a commercial decision to host a Farm of PZH services. Webinos as an open source foundation cannot directly address this, but supports it by providing the essential software assets.

Installers

Installers now exist for each of the main operating systems. These installers have be designed to be end user friendly and can be easily bound to whichever pzh provider the end user decides to use. Installers are provided for Windows, Linux, Android, Mac and we also support Raspberry Pi images.
Early prototypes of build instructions also exist for FirefoxOS, ChromeOS and iPhone.

**Appstores**
An application platform is only interesting if we have applications. Our stated appstore agnostic policy does limit, to some extent how we can address this problem. A prototype AppStore at apps.webinos.org has been provided (kindly donated by Fraunhoffer) and forms a neutral place where any developer can upload content.

More generally, and more importantly for the ecosystem, it is very easy for a device developer to integrate their own application store into a runtime, thus building their own ecosystem.

**Device Database**
A cross device application platform needs lots of devices to work with, to be of perceived value to an end user. The webinos “web driver” model has been crucial here, in term of making this process as simple as possible. Through this model it is possible to get pre-existing hardware as diverse as car system, set top boxes and health devices interoperating with webinos. And no on device installations are required.

A public database has been created to help people share their work on making compatible webinos devices

http://devices.webinos.org

**Foundation**
Finally and most importantly we have the webinos foundation. Webinos is now supported by an independent legal entity; a not for profit foundation which has been created with the explicit remit of furthering the webinos developments and governing its existing assets.

Webinos has been fortunate to date in being supported by a large list of major technology innovators from the academic and commercial spheres

We expect this to grow as we move forward and invite you to get engaged

hello@webinos.org

http://webions.org