Deliverable D3.1
FP7-ICT-2009-5 257103
June 2011

D3.1: webinos phase I architecture and components

Project Number	:	FP7-ICT-2009-5 257103
Project Title	:	Secure WebOS Application Delivery Environment (webinos)
Deliverable Type	:	Public

Deliverable Number	:	D 3.1
Contractual Delivery Date	:	June, 30th, 2011
Actual Date of Delivery	:	June, 30th, 2011
Title of Deliverable	:	webinos phase I architecture and components
Contributing Work Package	:	WP 3
Nature of Deliverable	:	Report
Editor	:	Fraunhofer FOKUS
Authors	:	Fraunhofer FOKUS, Deutsche Telekom, ERCIM, University of Oxford, Telecom Italia, TNO, BMW F+T, AmbieSense, SonyEricsson., Samsung, Antenna Volantis, VisionMobile, NTUA, ISMB, IBBT, Polito, UNICT, TUM, DOCOMO, Impleo, Telefonica

Document History

Version	Date	Author (Partner)	Remarks
0.9	29/06/2011	Fraunhofer FOKUS	Initial version created from Wiki
1.0 α	30/06/2011	Fraunhofer FOKUS	Updated, Word-formatted version

Abstract

This deliverable specifies the architecture and required infrastructure and service components for the first phase of the webinos project.

The primary areas covered in this document are the webinos foundations, extension handling, authentication, discovery, messaging, context, security, privileged applications and analytics. These topics are supplemented with a component overview, a high level network overlay architecture, as well as session and synchronisation handling decriptions.

In the first part of the deliverable, background information is given about the individual topics, presenting the lessons learned from the current state of the art, highlighting which existing standards and practices were suitable for adaptation by webinos and, in those cases where no existing standard covered the requirements of webinos, which standards and practices were used as a basis on which the webinos specifications were built.

This first part is primarily an information section, allowing readers not directly involved with the webinos project to follow the reasoning of decisions made in the work package 3 of the webinos project, without having to study, for example, the underlying use cases, scenarios and requirements from work package 2.

The second part of the deliverable contains the actual architectural specification for the webinos platform. From an implementation point of view, it should be sufficient to read only sections 4 and 5 of this deliverable and have all requires information regarding architectural and component specification available, though, without the background knowledge, this would just provide the "what" without the "why".

Two notes:

1. This document does not describe the webinos platform completelty. It only covers the architecture and its components. It is a companion document to the webinos D3.2 and D3.5 deliverables, which cover the JavaScript device APIs and the Security Framework, respectively. All three documents together comprise the webinos phase I specification.

2. This Word/PDF linear document represents only a snapshot of the specification for the purpose of dissemination, archieving, reviewing, delivery and dissemination as a single document. The actual specification is located on the webinos redmine/Wiki. That version is the one relevant for the work within the project. Due to the close interworking between the specification and the implementation work packages in webinos, experience gained about gaps that need to be filled in the specification will be fed back directly into the online specification. The Word/PDF document has been exported from the online version and represents the status of the specification on June, 30th, 2011.

Keyword list:

webinos, specification, architecture, foundations, authentication, discovery, messaging, context, security, analytics, metrics, network overlay, high level architecture, key architectural components, session creation

Content

1. Introduction

The purpose of the webinos project is to define and deliver an Open Source platform, which will enable Web applications and services to be used and shared consistently and securely over a broad spectrum of connected devices.

To achieve this, it is insufficient to limit the specification to APIs to be provided by individual devices to allow access to device resources, but it is also necessary to define and provide an architecture and infrastructure to allow applications to run not only on a single device, but also across devices and domains.

Increasingly users own more connected devices and users are no longer satisfied to handle devices and applications on an individual basis, but expect applications to keep preferences and status information synchronized across multiple domains, devices and, if appropriate, applications.

This applies to device features as well. Already many modern TV sets allow the use of smart phones as input devices, though this is currently handled on a proprietary manner. A modern Web based platform needs to define and provide functionality to handle such interactions in consistent and, for the application programmer, easily accessible manner.

Other issues that require services that go beyond the capability of individual devices are the handling of user authentication cross device events, metrics and application distribution.

In all these cases, it is not sufficient to provide a simple device API, but it is also required to describe the underlying architecture and service requirements.

The tagline of the webinos project is "Secure Web Operating System Application Delivery Environment", indicating that security is a significant part of the project. In the specification part of the project, the handling of security and privacy aspects and the creation and definition of a security architecture was covered by an individual project task to ensure that the topic is handled adequately.

To cover all areas adequately, the webinos specification consists of three parts:

- D03.1 webinos phase I architecture and components
- D03.2 webinos phase I device, network, and server-side API specifications
- D03.5 webinos phase I security framework

(The numbering is an artefact of the webinos project plan. There are no missing 3.3 and 3.4 deliverables - the numbers are reserved for an update of the 3.1 and 3.2 deliverables in phase II of the webinos project.)

The first deliverable (which is this one, D03.1) covers the architecture and required infrastructure and service components. The intended audience for this deliverable are providers of the webinos platform, since they will need to provide these components. For application programmers the background sections may be of interest to get an overview of these components and their interactions, since a good

understanding of the framework may allow for more efficient use of the system, though an in-depth knowledge of the internal interfaces and structures is not that useful. This is also one of the reasons to have the background section of this document separate from the detailed specification section.

The second deliverable (i.e. D03.2) describes the APIs that will be available to a Web application programmer on a webinos device. The intended audience are application programmers who want to provide webinos enabled and supported applications and are going to use the APIs. In the implementation phase of the platform, the audience, of course, also includes the platform providers, who will provide the APIs.

The third deliverable (i.e. D03.5) describes the security framework for webinos. As security needs to encompass the full application environment, this covers Web security architecture and services as well on-device security and policy handling.

All three deliverables together comprise the initial webinos system specification, which will serve as basis for the development of the open source platform.

Based on the experiences with implementing and using webinos, updated versions of these deliverables will be published in August 2012.

2. Methodology

The methodology for deriving the specification was based on the waterfall model.

In the initial step, a long list of scenarios and user stories involving a Web application platform were defined (see also Deliverable 2.1).

From these scenarios and user stories, a sub-set was selected, representing advanced, innovative and typical usage.

Based on these representative scenarios, which were mainly informal descriptions of applications and their usage, use cases were derived, which describe the interaction between the actors and the software in a more formalized way. From these use cases, requirements pertaining to the underlying platform, and including security requirements, were extracted (see also Deliverable D2.2). This process ignored requirements that were application specific, unless they implied requirements for the underlying platform.

Following this phase, the requirements (derived from presumed representative scenarios) were checked against the original set of scenarios and user stories to ensure that no requirements were missed.

Based on these requirements, together with known requirements from other Web application platform work performed by project partners (such as W3C, BONDI, WAC or OIPF) the webinos specification was developed. Since it is unpredictable what features a future application, utilizing the platform, might require, the set of requirements for a platform can never be complete. However, basing the requirements on a wide range of innovative scenarios, combined with the experience of the project

members and external feedback, should result in a specification that fills the most crucial currently foreseeable needs and remains open for adaption to future needs.

This anticipated adaptation for future needs will be covered in the webinos project in Phase 2, which will review, revise and refine the specification in this document. The development of the platform and applications will serve as proof-of-concept and will provide valuable feedback for Phase 2 enhancement of the specifications.

Guidelines

When drafting the specification, webinos project members took a number of guidelines into account:

Don't re-invent the wheel

Where solutions in an area already existed and were found valuable and acceptable, these were utilized and adapted as far as possible and needed. There was a strong bias in the project against innovating for the sake of innovating. Existing solutions, standards and specifications were referenced and re-used. If existing solutions were almost sufficient, but not meeting webinos requirements in all details, care was taken to do only the smallest number of changes, needed to fulfil the requirements.

Consider licensing issues

If available and appropriate, the specifications are based on open standards to avoid the specification and subsequent implementations to be encumbered by legal and licensing issues, hindering adoption. In areas where only proprietary standards were available or dominating, care was taken to provide specifications in a way to allow implementation independently of proprietary solutions, but, if possible, stay close enough to them to allow easy mapping to such solutions to allow their usage in environments where these are ubiquitous.

Be secure by design

To avoid the common risk of developing an architecture and a specification solely on functional requirements and then tagging on security as an afterthought, webinos has a specific task dedicated to developing a security and policy framework and integrating it into the architecture from the beginning. Unlike other parts of the specification work package, which operate in two distinct phases, the security task runs continuously across the project life time to ensure that security and policy concerns can be addressed quickly and sufficiently.

Be developer friendly

The key for success of a platform is the acceptance by developers. To achieve this, webinos needs to provide features to allow developers faster, more attractive and more efficient development - of applications that can communicate and use resources across devices and apps. Whilst current platforms provide access to local hardware and other resources, they do not provide an infrastructure and

significant support for multi-device usage. Communication, cooperation and sharing between apps and devices currently need to be provided by the developer.

To make life easier for developers, webinos considers not only the API on the device itself (covered in deliverable 3.2), but also a supporting infrastructure for, among others, user ID management, discovery (of devices, applications and services), event handling across devices, metrics and context handling. webinos seeks to provide the developers and service providers with a common non-proprietary infrastructure enabling or simplifying the use of resources between applications (also across different developers).

Work Areas

To create the first specification of the webinos platform, based on the requirements and following the guidelines, fifteen work groups were created initially - tasked to analyze existing solutions, issues and remaining gaps in respect to the requirements.

As a result of this initial work, some overlap between the areas was detected and the groups were restructured to avoid duplication of work and unnecessary communication overhead. (As an example: Whilst the scope and problem space of the groups *Application/Service Discovery* and *Device Discovery* differed, the similarities of these groups were larger and consequently, the two groups were combined.)

Work then continued in ten work areas for which specifications were defined, which led to the Overlay Network Architecture and the seven specification areas contained in this deliverable. (For better structuring of the deliverable, two of the smaller areas, *Web application packaging/handling* and *Extension Handling* were combined in the *Foundations* section; *Privileged Applications* became part of the *Policy* chapter, reducing the ten work areas to eight sections of the specification.)

The purpose of splitting the effort into different, mostly disjunct, work areas was to be able to work in small teams, often based in only one or two companies, to allow fast and efficient work and communication within the work areas. To avoid the risk of company bias and to ensure the fitting in of the individual areas into the overall concept of webinos, alignment and "bringing the pieces together" was assured through regular conference calls, cross-team workings and through final peer review across the teams.

3. Background

Foundations

The foundations section is about specifying the technological background of webinos applications, including application packaging and life-cycle. This also includes functional and non-functional requirements of webinos Web Run-Time (WRT) environments, e.g., how to pass applications to webinos WRTs and how to share applications across WRTs.

What's in scope

Foundations define how webinos look like from a technical point of view. In general, it defines the packaging and configuration of webinos applications as well as the life-cycle, including requirements on the WRT which are related to application handling and interoperability across WRTs. It also includes how webinos applications can be packaged for distributed application use cases, including the exposure of application functions. Thus, applications will be able to share their functionality across distributed components of an application as well as across other full applications.

What's out of scope

Allowing Web based applications to access device specific features introduces security risks. The foundations specification does not define the webinos security framework.T. This is done in separate sections and deliverables, but it includes relations where needed. Also content protection like DRM or licensing is out of scope.

Webinos applications will be created using Web technologies, e.g., JavaScript, CSS and HTML. To achieve a good level of interoperability between WRTs, a common set of supported Web technologies is crucial; but elaborating on Web technologies to be supported by webinos WRTs and defining which features of which Web technology must be supported is not discussed in the foundations section. Instead, WAC has done much work here and the outcome is referenced and mandated for webinos WRTs.

Review of State of the Art

In general applications have a central entry point for both, installing and executing, which makes them easy to transport, install and use. Web technology based applications are commonly hosted on Web servers, where each document is linked to other documents which are needed for proper rendering. This approach provides a good mechanism to access an application simply by using an URL. But there is a lack of descriptive data for all this content (including e.g. JavaScript).meta-data like the author, an application description or links to contacting the authorauthor, which can be valuable for the user. This information can be added to native applications like MS Windows executables. In addition, in the Web there is no means to describe which documents are belonging to an application and are needed for a

proper execution. In this section some recent approaches for packaging Web content as a whole application are described.

Google Web application packaging (.crx)

Chrome Google introduced support for installable hosted and packaged Web applications (stored in .crx format). The developer has to write a JSON based manifest file (manifest.json) that contains some metadata about the application. This manifest.json file must then be placed in a .crx file, which basically is a renamed zip file.

Example of Usage taken from Google Chrome Developer Page [CRX]

```
{
  "name": "Google Mail",
  "description": "Read your gmail",
  "version": "1",
  "app": {
    "urls": [
      "*://mail.google.com/mail/",
      "*://www.google.com/mail/"
    ],
    "launch": {
      "web_url": "http://mail.google.com/mail/"
    }
  },
  "icons": {
    "128": "icon_128.png"
  },
  "permissions": [
    "unlimitedStorage",
    "notifications"
  ]
}
```

The JSON example describes the Google Mail Web application as installable hosted application, the application URLs pointing to remote locations, using the *web_url* key. In addition to the provided metadata, HTML5 permissions can be requested during installation. If the user clicks a link to a .crx file within the Chrome Web browser, the application is installed to the Google Chrome application Gallery.

If the developer doesn't want to maintain a server that serves a hosted application or if he wants to provide the best off-line case experience, he can create a packaged application. To make an application a fully packaged application the contents are placed in the .crx file and the manifest file must include these details:

```
  "app": {
    "launch": {
      "local_path": "main.html"
    }
  },
```

In addition to access to common Web features, installable applications can have access to Google Chrome's extension APIs, e.g., manipulating context menus or creating background pages. To stimulate

the usage of Google's application packaging, Google's Web application store supports .crx files which can be uploaded to the store using a developer frontend. Afterwards they are search and browse-able in the store where they can be installed from to the local application Gallery.

Mozilla Web Applications

In early 2011 Mozilla announced the Open Web Apps project [OpenWebApps] which aims to allow everyone to develop their own Web application store. This also includes the definition of application packages and the possibility of installing them. Mozilla also uses a JSON based manifest file that includes human readable and machine readable metadata about the application. Applications are able to "self-install" using an API call provided by Mozilla Browsers (navigator.apps.install()). Manifest files can be served as files where the file extension .webapp or via HTTP where the content type application/x-web-app-manifest-json should be used. Off-line usage is supported through the use of HTML5 AppCache, while an API to check the online status is provided.

Example of Usage taken from Mozilla Developer Page [OpenWebApps]

```
{
  "version": "1.0",
  "name": "MozillaBall",
  "description": "Exciting Open Web development action!",
  "icons": {
    "16": "/img/icon-16.png",
    "48": "/img/icon-48.png",
    "128": "/img/icon-128.png"
  },
  "widget": {
    "path": "/widget.html",
    "width": 100,
    "height": 200
  },
  "developer": {
    "name": "Mozilla Labs",
    "url": "http://mozillalabs.com"
  },
  "installs_allowed_from": [
    "https://appstore.mozillalabs.com"
  ],
  "locales": {
          "es": {
     "description": "¡Acción abierta emocionante del desarrollo del Web!",
     "developer": {
       "url": "http://es.mozillalabs.com/"
     }
    },
    "it": {
     "description": "Azione aperta emozionante di sviluppo di fotoricettore!",
     "developer": {
       "url": "http://it.mozillalabs.com/"
     }
    }
  },
  "default_locale": "en"
```

```
37      }
```

HTML5 AppCache

W3C's HTML5 introduces an application cache, which allows Web content available on the local device for off-line usage. Thus, online based applications can be used without internet connection. To add AppCache support to a Website, a specific manifest file must be provided on a server that must be referenced from each HTML page of the whole application. The manifest defines which parts can be online and which must be available offline. In addition, a fallback can be provided for the files only accessible when online.

Example of Usage HTML5 AppCache [AppCache]

```
1 CACHE MANIFEST
2 NETWORK:
3 /online.cgi
4 CACHE:
5 /offline.css
6 /offline.js
7 /offline.jpg
8 FALLBACK:
9 /fallback.html
```

W3C Widgets

The W3C Web Applications Working Group started to work on Widget specifications [W3CWidgetFamily], small packaged and installable Web applications, back in 2006. Currently the main specifications relatated to packaging and configuration, APIs, and signing are in a last call phase which means that the specifications are mainly completed and W3C recommendations are upcoming. A W3C Widget is basically ZIP file which contains Web documents like html, css, or JavaScript in addition to media files like pictures. Everything a Widget needs to be functional must be located in this application package which ensures off-line capability. For describing the content of the package a manifest file is contained which contains meta data like author, application description, desired screen modes and signatures.

The following small example describes a Widget with an attached title, an application icon, which can be shown to the user by the Widget runtime, and a start file, which is used by the Widget runtime to execute the application.

Example of Usage W3C Widget configuration file

```
1 <widget xmlns="http://www.w3.org/ns/widgets">
2   <name>Hello World Widget</name>
3   <content src="index.html"/>
4   <icon src="icon.png"/>
5 </widget>
```

Additional specifications of the W3C Widget family describing API access to the applications meta data, how to update widgets over HTTP, and how to sign and validate the origin of Widgets.

Opera Widgets

The Opera Widget [OperaWidgets] specification has slight differences compared to the W3C one. The configuration file contains information about the author, the application, potential icons and security related requirements while the packaging is also a ZIP file, changing the extension from .zip to .wgt. However, Opera claims that they will support W3C Widgets if the specifications are final. In addition to accessing the configuration documents metadata, the Opera APIs provide basic application live cycle management, e.g., allow applications to react on events like "gone to background" or "gone to foreground". The following example shows the same semantics as the W3C description.

Example of Usage W3C Widget configuration file

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <widget>
3   <widgetname>Hello World Widget</widgetname>
4   <icon>icon.png</icon>
5 </widget>
```

WAC Widgets

Widgets as specified in the Wholesale Application Community are compliant to W3C Widgets with additions related to security and privacy. For example, WAC defines a policy system to protect access to device features and user data which must be declared in the Widget configuration document and evaluated during installation of the Widget.

Recommendations from state of the art

W3C Widget specifications describe different parts around packaging and handling Web applications. This includes packaging, signing and the definition of APIs which are all also relevant for webinos. Also, these specifications are in a closely to final stage and will be W3C recommendations soon and the industry is adapting it (e.g., WAC and Opera). Thus webinos should base its application definition on the W3C Widget specifications and extend them in order to meet additional webinos requirements like distributed application design and exposing functions as service.

Extensions

Extensions in webinos provide access to unique device features as stated in requirement CAP-DWB-FHG-002 and described in the use case WOS-UC-TA3-004 "Embedding Proprietary Extensions".

In order to enable third party developers to build and use extensions, a sub system to handle extensions has to be established.

In the browser space there are several solutions available, which we can leverage from. However, there is a fine distinction between browser plug-ins (e.g., Adobe Flash) and extensions/add-ons (e.g., Firebug). Whereas plug-ins add support for alternative content types to the rendering engine (which can be embedded into a Web application), extensions modify or add to existing functionality of the browser. From a generic stand point, three distinctive parts have to be specified for the extension handling:

1. Application APIs for accessing extensions inside a webinos application,
2. Pre-defined interfaces for integrating the extension into the webinos runtime (e.g., initialize function of the extension, mapping of extension methods/attributes to JavaScript methods/attributes)
3. Data schemes for providing metadata about the extension (e.g., name, supported platforms)

Furthermore, we can distinguish extensions in webinos by their "user group". There are on the one hand platform specific extensions, which are available to all applications executed on the device and on the other hand there are extensions which can be coupled with a specific application.

The platform specific model is used in the general browser plug-in concept. Once the plug-in is installed on the device, the plug-in will be usable by all Web applications, which embed an object mapped to the specific MIME-Type of the plug-in.

The concept of application specific extensions has been applied in HP webOS Plug-in Development Kit (PDK) [HP-PDK]. A similar approach can be found in Chrome extensions, where an extension can embed a NPAPI plug-in [Chrome-NPAPI].

State-of-the-Art extensions and plug-ins in the browser environment

In the state-of-art analysis we are going to evaluate different solutions for extensions in webinos such as browser-plug-in (NPAPI), browser extensions (Chrome extensions) and JavaScript engine add-ons and provide a recommendation, which solution shall be incorporated into the webinos runtime.

Plug-in standards

The Netscape Plug-in API (NPAPI) has been adopted by all major browser platforms, ranging from Webkit browsers (Safari, Chrome) to Firefox and Opera. MS Internet Explorer does not support NPAPI in favour of ActiveX.

Plug-in are executed directly on the underlying operating system. NPAPI plug-in are browser independent but rely heavily on the operating systems, especially for 2D and 3D graphical output or audio output. For each operating system the plug-in needs to be customized and compiled. However, there are a few frameworks such as [FireBreath] or [Luce] available for simplifying the cross-platform development of NPAPI plug-ins.

In order to provide a richer interaction between a Web application and a NPAPI plug-in, the NPAPI addition "npruntime" was introduced. npruntime has been adopted by all major platforms as well. [npruntime]

Google proposed an extension to NPAPI called PEPPER (or PPAPI) to reduce the dependencies between the plug-in and the operating system. Currently PPAPI is only supported by Chrome [PPAPI]. Mozilla stated that they are not interested in working on PPAPI at the moment [moz-ppapi].

The unlimited and direct access to the operating system for plug-ins raises many security considerations, but is nevertheless an important factor to build unique Web applications and enabling the access to unique device features. To overcome the security concerns about NPAPI, Google introduced the Google Native Client project (NaCl) to execute native code in a sandboxed environment and prohibits the access to all hardware resources (e.g. file-system).

Due to the lack of support for PEPPER in different browser runtime and the limited usability of NaCl we are going to focus our analysis on plug-ins on NPAPI.

Using a NPAPI plug-in

From a Web application developer's perspective the usage of a plug-in is fairly simple. The following lines of code describe how an app developer checks if a plug-in for given MIME-Type is already installed on the device and how the Web application can interact with the plug-in afterwards.

```
1 if (navigator.mimeTypes["application/webinos-extension-x1"] &&
2     navigator.mimeTypes["application/webinos-extension-x1"].enabledPlugin != null){
3   document.write('<embed type="application/webinos-extension-x1">');
4   var embed = document.embeds[0];
5   embed.nativeMethod();
6   alert(embed.nativeProperty);
7   embed.nativeProperty.anotherNativeMethod();
8 }
```

Building a NPAPI plug-in

The NPAPI standard mandates the developer to embed methods inside the plug-in for interaction with the browser, as described in the following document [npapi-plugin-side-api]. These methods include the initialization (NP_Initialize), terminiation (NPP_Destroy) of the plug-in as well as receiving information about the supported MIME-Types (NP_GetMIMEDescription) and version of the plug-in (NPP_GetValue). NPP_GetValue also provides mechanism to handle requests from the Web application.

As described in [npapi-browser-side-api] and [npruntime] the browser itself has to embed several methods in order to support NPAPI plug-ins. The API exposed by the browser to the plug-in incorporates methods to invoke JavaScript functions of a Web application (NPN_Invoke), to allocate memory of the browser mem space (NPN_MemAlloc) or to receive information about the browser engine (NPN_GetValue).

Extensions

There are no cross-browser extension standards available. Each browser engine provides a different set of functionality for its extensions.

Firefox provides for their add-ons anefficient interface called js-cytpes to invoke native libraries without the need to integrate an extensions into Mozilla's XPCOM architecture. The js-ctypes is detailed in the following section. The interaction possibility between a Web application and the extension are fairly limited and is possible using events.

Extensions in Chrome are a zipped bundle of files (HTML, CSS, JavaScript, images). Extensions are essentially Web pages with access to all the APIs that the browser provides to Web pages. They can interact with Web pages or servers using content scripts or cross-origin XMLHttpRequests. Additionally extensions can also interact programmatically with browser features such as bookmarks and tabs.

However, there are no direct mechanisms available for extensions to call JavaScript functions of a Web page or vice versa. JS functions can be invoked using DOM manipulation.

Although there is no API provided to interact with the underlying operating system, NPAPI plug-ins can be part of zipped bundle.

A prototype chrome extensions for webinos built with the webinos discovery plug-ins underlines the weakness in communicating between the web-app and the extension.

Direct JavaScript additions

The JavaScript engine plays a crucial role in the webinos runtime. For that we are going to analyze two projects, which propose methods to access the native functions outside of the JavaScript engine. These two projects are add-ons in Node.js [node.js] and as already mentioned js-ctypes for Firefox extensions [js-ctypes].

js-cytpes

js-cytpes is an interface for add-ons in Firefox running inside the chrome. The add-on cannot interact with scripts of a Web application.
js-ctypes is a slim interface to call native libraries stored on the hosting device. It enables the access to these libraries, but does not provide any methods to store or install platform specific binaries. The following code snippet illustrates how js-cytpes can be used by a developer to open the native message box on a Windows system.

```
 1 /* importing the js-ctype library */
 2 Components.utils.import("resource://gre/modules/ctypes.json");
 3 /* TODO */
 4 var lib = ctypes.open("C:\\WINDOWS\\system32\\user32.dll");
 5 /* Declare the signature of the function we are going to call */
 6  var  msgBox  =  lib.declare("MessageBoxW",  ctypes.winapi_abi,  ctypes.int32_t,
ctypes.int32_t,ctypes.jschar.ptr, ctypes.jschar.ptr,ctypes.int32_t);
 7 var MB_OK = 3;
 8 /* triggering the previous declared function*/
 9 var ret = msgBox(0, "Hello world", "title", MB_OK);
10 lib.close();
```

[using-js-ctypes]

node.js addons

Node.js is a server-side JavaScript environment that uses an asynchronous event-driven model. It is based on Google's JavaScript engine V8. Add-ons for node.js are dynamically linked shared objects and provide glue to C and C++ libraries [node.js].

From an application developer perspective the usage of an add-on in Node.js is straight forward as illustrated in the following code snippet.

```
1 var extension = require(./extension);
2 extension.doSomething();
```

The development of an add-on in node.js involves knowledge of numerous libraries:

1. V8 JavaScript library for creating objects, calling functions etc
2. libev, a C event loop library, if there is need to wait for a file descriptor to become readable, wait for a timer, or wait for a signal to received one will need to interface with libev. That is, if you perform any I/O, libev will need to be used. Node uses the EV_DEFAULT event loop.
3. libeio, a C thread pool library for executing blocking POSIX system calls asynchronously.

All Node add-ons must export a function called init with the following signature:

```
1 extern 'C' void init (Handle<Object> target)
```

Mapping requirements to technical solutions and Recommendation for the webinos runtime

Table X compares the different solutions with the relevant requirement developed in work package 2. The table provides an overview how the different solutions fulfil the relevant requirements.

	NPAPI	**js-ctypes**	**Node.js Add-ons**
(CAP-DWB-FHG-002)	Designed to add support for	Enables the	Enables the developer

The webinos runtime **SHOULD** allow access to non-webinos APIs to device features	additional MIME-types for the rendering engine. Plug-in is executed on the OS level. It supports graphical output inside a web-application.	developer to call native libraries within JavaScript. No graphical output inside the web-application supported.	to execute native code on the OS level. (Add-on is statically linked). No graphic output inside the web-application possible.
(PS-DWP-ISMB-202) The webinos runtime **MUST** ensure that an application does not access device features, extensions and content other than those associated to it.	Not supported. Mechanisms for (dis)allowing to load plug-in needs to be integrated	Not supported. Mechanisms need to be integrated	Not supported. Mechanisms need to be integrated
(CAP-DEV-FHG-100) Access to resource on remote devices **SHALL** be available	Not supported. Addition would be required. Hard to enable since NPAPI is tightly coupled to the Web application DOM events. What about the graphical output? What about the graphical output, when remotely accessed?	Not supported	Partially supported. Server module of Node.js could be used to make extensions remotely available. Middleware for exposing the data needs to be developed.
It **SHALL** be possible to define meta-packages containing a collection of applications and/or extensions.	Partially fulfilled. For application specific extensions, the plug-in is part of the application package and could be described in the packages/application manifest	Not integrated in the system yet	Not supported
Extensions **SHALL** be packaged in a way that is as similar as possible to applications.	NPAPI is one binary file. Meta data about a NPAPI plug-in such as name, version, description is stored in the binary itself.	No package system defined	Each node.js add-on is described by a manifest file in JSON syntax. Add-ons are not packaged, but are stored in a separate folder
Extensions **SHALL** be treated in a way that is similar and consistent	Partially fulfilled: Plug-in is embedded object in HTML and		Extension API is used in the same way as the

with standard device features.	provides a scriptable interface.		regular APIs
An Extension that contains platform-specific code **MUST** be associated with the supported platform(s).	Must be specified in the metadata description of the application	JavaScript code is OS specific. Platform association needs to be integrated	Must be specified in the metadata description of the application

For local usage a solution based on NPAPI is the most compelling one. It's widely supported in browser runtimes and supports graphical output on the device. The graphical output could be relevant for games (one of the reasons, why HP/Palm introduced the webOS PDK). A remote access to a NPAPI plug-in could be achieved, but would be limited to its scripting interface.

Security aspects

Security aspects are detailed in Deliverable 3.5

Authentication

User authenticity is the property granting that the user who wishes to access the system is whom he declared to be.
Verifying the user identity is often the first step for granting other security properties, like authorization (what the user can do) and access control (what resources the user can access).

The webinos framework aims to grant authentication property in a user-friendly fashion, hiding to the user and to the application developer most of the more complicated aspects of the authentication mechanisms.

What's in scope

Authentication topic involves:

- Authentication to the personal zone (user authentication with the personal zone hub).
- Authentication outside the personal zone (user authentication with the service provider). Preliminary analysis of problems and possible solutions, more analysis is deferred to phase 2
- Personal zone identity data management (where the user credentials are stored, how are used, how are synchronized with personal zone proxies).

What's out of scope

Anonymous authentication methods (e.g. group signature, direct anonymous attestation, Idemix) and identity based encryption methods are deferred to phase 2.

Review of State of the Art

OpenID

OpenID is a user centric, decentralized authentication protocol using Web technologies allowing single sign on. An OpenID provider can do the authentication of a user for some service and the service does not have to store identity or credential information.

OpenID uses standard HTTP(S) requests and responses. Protocol extensions exist for example for attribute exchange. The identifier used is either a HTTP(S) URI or an XRI (Extensible Resource Identifier)

Oauth

It allows a resource owner to grant a client access without giving away its credentials for the resource. OAuth uses Web technology (HTTP(S)) to give fine grained access.
The client requests authorization from the resource owner. The authorization request can be made directly to the resource owner, or preferably indirectly via an intermediary such as an authorization

server.
The client receives an authorization grant and requests an access token by authenticating with the authorization server using its client credentials and presenting the authorization grant. If the client credentials and the authorization grant are valid, the authentication server issues an access token.
The client requests the protected resource from the resource server and authenticates by presenting the access token.
If the access token is valid, the resource server provides access to the resource.

WebID - Web Identification and Discovery

WebID is an early draft by W3C which intends to define how to perform user authentication on the Web using X.509 certificates, TLS and URIs. The user agent (UA) is associated to the user by a URI. Both endpoints use TLS to exchange their X.509 certificates for authentication. At the moment, it is not clear if WebID will evolve to a standard, but if so, it would be quite interesting for webinos as WebID relies on widely used technologies.

Liberty Alliance / Kantara

Liberty alliance is a consortium for developing a distributed identity management system. It includes an Identity Federation Framework (ID-FF), an Identity Web Service Framework (ID-WSF) and Identity Services Interface Specifications (ID-SIS). ID-FF enables identity federation and management and it is designed to work with heterogeneous platforms and with all types of network devices; ID-WSF provides a framework for creating, discovering, and consuming interoperable identity services; ID-SIS are a collection of specifications for interoperable services to be build on top of ID-WSF.
The work of the Liberty Alliance is transitioning to the Kantara Initiative.

The Alliance adopts and extends industry standards, rather than attempting to develop similar specifications.
ID-FF Liberty architecture needs an Identity Provider (IdP) and uses HTTP protocol to exchange messages between IdP and Service Provider to authenticate the User Agent.
ID-WSF is a foundational layer that utilizes the ID-FF and provides services. The Discovery Service determines where the needed resources are located (e.g. user attributes). The Interaction Service allows an IdP to interact with the owner of the resource that it is exposing. The Data Services supports the storage and update of specific data attributes regarding a user.
ID-SIS provides specifications for interoperable services (e.g. Geo-location Service, Personal Profile Service Specification, Employee Profile Service Specification, Contact Book Service Specification).

Shibboleth / SAML

It's an open source implementation of SAML 2.0 specifications. It provides an authentication and authorization infrastructure to allow federated Web single sign on and attribute exchange. A user authenticates with his organizational credentials. The organization (or identity provider) passes the minimal identity information necessary to the service manager to enable an authorization decision.

SAML 2.0 is an XML-based open standard for exchanging authentication and authorization data between an identity provider and a service provider. Its specifications recommend SSL 3.0 or TLS 1.0 for transport-level security; XML Signature and XML Encryption for message-level security. SAML 2.0 permits direct use of XML Encryption in various places, including an <EncryptedID> element that can replace the usual <NameID> element.
SAML 2.0 allows for arbitrary mappings between any two formats by using the <NameIDPolicy> element to describe the properties of the identifier to be returned.

Kerberos

Kerberos is a mutual client/server authentication system designed to establish sessions and support the secure transfer of data. Kerberos can be used as a single sign on mechanism.

It requires a trusted third party and uses tickets and ticket granting tickets to allow it to scale to multiple services without repeated user authentication. Kerberos does not require the use of asymmetric cryptography and uses time stamps for validity periods.

Identity mechanism of XMPP

XMPP is an XML based protocol for near-real-time messaging, presence and request-response services for confidential and integral message exchange TLS.

The XMPP identifier (e.g. node@domain/resource) has as mandatory field only the domain identifier and is used to address an endpoint. To authenticate an endpoint SASL is used enabling a server to offer multiple authentication methods from which a client can choose.

Identity Metasystem

"The Identity Metasystem is an interoperable architecture for digital identity that enables people to have and employ a collection of digital identities based on multiple underlying technologies, implementations, and providers."

Three different parties participate in the Metasystem:

- **Identity Providers**, which issue digital identities.
- **Relying Parties**, which require identities.
- **Subjects**, which are the individuals and other entities about whom claims are made.

Five key areas compose the Identity Metasystem:

- Identity representation using the data elements carried in Information Cards (called claims). Claims are carried in security tokens in the same way adopted per Web service security (called WS_Security, an extension to SOAP to apply security to Web services).

- A negotiation process among identity providers, relying parties, and subjects. Negotiation occurs using WS-SecurityPolicy (an extension of WS-Security) statements exchanged using WS-MetadataExchange (a Web Services protocol specification). Identity Metasystem is flexible enough to carry various format of token and different kinds of claims needed for a digital identity interaction
- An encapsulating protocol to obtain claims and requirements. The WS-Trust (an extension of WS-Security) and WS-Federation (an Identity Federation specification) protocols are used to carry requests for security tokens and responses containing those tokens.
- A means to bridge technology and organizational boundaries using claims transformation. Security Token Services (STSs) as defined in WS-Trust (an extension of WS-Security) are used to transform claim contents and formats.
- A consistent user experience across multiple contexts, technologies, and operators. This is achieved via Identity Selector client software such as Windows CardSpace representing digital identities owned by users as visual Information Cards.

Firefox account manager

The Account Manager allows users to create new accounts with optional randomly generated passwords, and log into and out of them with a click.

The Account Manager specification proposes two changes to Web sites:

1. The browser needs to know how to register, sign in, and sign out. A static JSON document describes what methods the site supports and how they should be executed.
2. The browser needs a way to check which user (if any) is currently signed in. To do this the site has to set an HTTP header or to supply a URL the browser will ping.

Recommendations from state of the art

SAML 2.0 standard could be useful to exchange authentication data outside the personal zone, to log into external services. It can also be used to login to the personal zone (to be more precise to login to the personal zone hub) and to synchronize authentication data among the personal zone hub and the personal zone proxies.
An account manager similar to the Firefox one could be hosted on the personal zone hub (with a copy into the personal zone proxies) to implement a more user-friendly authentication mechanism.

Discovery

Discovery is a procedure for retrieving addressing information of a device or services either through local or remote networking access mechanism.

The discovery mechanism varies on different discovery protocol adopted. Some technologies are not involved in discovery mechanism directly; rather they gather discovery information from existing discovery protocol, e.g. Serverless XMPP uses ZeroConf as its underlying discovery protocol and XMPP Core uses DNS resolve to gather server information.

The webinos discovery framework aims to define a set of interfaces that hide the complexity of internetworking technologies for both third party developers and Web developers.

What's in scope

Webinos discovery mechanism investigates the following issues:

- Local Discovery that are based on a variety of local discovery protocols, e.g. UPnP, ZeroConf, BT, WiFi, and USB.
- Remote Discovery that enables remote access to devices or services. Technologies investigated include XMPP and its extensions, Web Introducer and Web Finger. Distributed Hash Table (DHT) has been investigated, but deeper analysis is postponed to Phase II (see below)

What's out of scope

DHT is an optional P2P discovery technology for webinos. It provides a better flexibility to discover. However, in comparison, XMPP is capable of providing functions for storing information about friends, authenticating system, finding services. It also defines Protocol format required for webinos architecture communication. In Phase I, we focus on XMPP technology. Further investigations on DHT is not in scope of this phase of work.

Focus will be on Bluetooth and USB devices for local discovery in Phase I, this will give an understanding on local discovery aspect. Firewire and Zigbee will be considered in Phase II, since they are not widespread local communication technologies. In respect to the target core of webinos, their analysis is deferred.

Review of State of the Art - Local Discovery

Zeroconf

Zeroconf is set of technologies to address following issues:

- Address configuration (assigned address using DHCP or host-configured link local address)

- Resolving host name to IP Address using Multicast DNS
- Description of services supported on device and way to communicate with the device, using DNS-SD Service Discovery

The multicast DNS/DNS-SD in Zeroconf makes use of following commands to browse, find address and named instances.

- Register (Services)
- Browse (Named instances)
- Resolve (Address and ports)

It fetches following information at end of DNS service discovery:

- Pointer record - PTR : mapping address to name which is of form <servicename>.<transport_protocol>.domain>(e.g. operator._http._udp.local, where operator is a unique user-visible name, no other node can have same name, _http._udp is protocol, and local is the domain)
- Service locator - SRV record (hostname + port) (e.g. operator.local port 6313). Specifies service location for fetching information about the protocol. This is also used in XMPP/SIP messages.
- Text Record - TXT (e.g. pdf:application/postscript).

Host offering publishes instances, service type, protocol information, domain name and config parameters.

See implementation section for more details on ZeroConf examples and message format.

Depending on the messages, they are sent as unicast or multicast query. New services announce about their presence. Addresses are resolved before sending packets, if it fails to find devices, it updates other devices too about the service unavailability.

In wide area network (i.e., if domain is specified such as dns-sd.org) it will fetch the services available in this domain. This could solve the problem of service discovery outside local domain but imposes a domain registration for each user, which in general is not practical.

UPnP

UPnP is a ISO standard for home network that supports automatic configuration, i.e. the network should be self configurable. It is a protocol based approach and, differently from older PnP technologies, no device drivers are involved. UPnP specification is based on the following protocols: HTTP, TCP/UDP, SOAP, and SSDP.

Communication is between controllers/control points and controlled services. One or more services are combined to form a device. Controlled devices handle requests from a control point.

UPnP allows a device to advertise its services to control points and allows a control point to search for devices. Results of discovery are device type, device unique identifier, and URL for obtaining device description. Search and advertise are multicast messages, while request/response between control point and device are unicast messages. If IP address or services are changed it is the responsibility of the device to advertise the changed IP address. Each device, embedded device and service offered by each device should be advertised via multicast. Because of the nature of the UDP, advertisement messages are sent multiple times and are based on the value specified in Cache Control. Cache controls the expiry time of advertisement.

In UPnP, discovery is done using SSDP (Simpler Service Discovery Protocol), it is based on HTTP protocol 1.1. It is a simple protocol which comprises of start line and list of message headers.

```
SSDP Start line:
```

- `NOTIFY * HTTP/1.1 \r\n`
- `M-SEARCH * HTTP/1.1 \r\n`
- `HTTP/1.1 200 OK \r\n`

Bluetooth

The process for Bluetooth service discovery involves two steps - inquiry of all nearby devices, and connection to each of those devices in order to search for the requested services.

HCI inquiry can be used to detect nearby devices. It provides a command interface to the baseband controller and link manager, and access to hardware status and control registers. Essentially this interface provides a uniform method of accessing the Bluetooth baseband capabilities.The HCI exists across three sections, the Host - Transport Layer - Host Controller. Each of the sections has a different role to play in the HCI system.

In order to search services, the bluetooth stack provides Service Discovery Protocol (SDP). SDP enables network devices, applications, and services to seek out and find other complementary network devices, applications, and services needed to properly complete specified tasks. The attributes of a service include the type or class of service offered and the mechanism or protocol information needed to utilize the service.

SDP involves communication between a SDP server and a SDP client. The server maintains a list of service records that describe the characteristics of services associated with the server. Each service record contains information about a single service. A client may retrieve information from a service record maintained by the SDP server by issuing a SDP request. If the client, or an application associated with the client, decides to use a service, it must open a separate connection to the service provider in order to utilize the service. SDP provides a mechanism for discovering services and their attributes, but it does not provide a mechanism for utilizing those services.

Normally, a SDP client searches for services based on some desired characteristics of the services. However, there are times when it is desirable to discover which types of services are described by an SDP server's service records without any a priori information about the services. This process of looking for any offered services is called browsing.

Review of State of the Art - Remote Discovery

XMPP (Extensible Messaging and Presence Protocol)

XMPP architecture and messaging protocol is quite simple and is primarily a client-server technology. The established connection between client and server allows searching for friends, resources, items and services. **Resources** could be user devices; **items** are the devices which are not IP capable (e.g USB). More description about resources, items and services is included in the protocol definition section.

XMPP core is defined in RFC 6120, it describes the connection between XMPP client and server over public IP. It involves stream message exchange, feature and mechanism negotiation, security (TLS and SASL), presence, and message exchange. Discovery mechanism will rely on XMPP extension, XEP-0030 to search services, XEP-0174 for Serverless messaging (Discovery is based on ZeroConf, once device are resolved they can use XMPP). At different layers of webinos system different XMPP extensions can be used and if particular exchange of message is required to support it is easy to implement extension using XMPP. More details about nodes and items is covered in protocol definition section.

XMPP in case of public IP uses DNS SRV record to fetch information about the domain. At transport layer it uses TLS (Transport Layer Security) and allows usage of different SASL authentication mechanism to authenticate users. Different XMPP servers impose different mechanisms and different compression technologies to communicate between client and server.

Distributed Hash Tables (DHT)

DHT allows sharing of information between peers. Users wishing to exchange information use a hash file and a filename. The filename becomes the key and is searchable by entities connected to same bootstrap and to same port. One entity acts as server that holds information about keys and responds back to clients about the node holding key. All nodes connected exchange information about nodes available i.e. which nodes are online and are checked for their presence from time to time.

There are different DHT Protocols such as Chord, Tapestry, Kademlia, which differ in how nodes are found. DHT is relevant to webinos and could be used for remote discovery. The main issue is to find relevant keys; we should use a layer on top to do searching of all remote nodes with appropriate search query. For example, searching for George specific device, information is stored in form of a key. Information can be only obtained if user searching gives correct names based on which key was generated. If the key is generated in form of George+Bluetooth, searching for George+bluetooth will not get result. A layer on top is required to handle these permutations to search for the proper key.

If no adjacent clients are there, it is not clear how it communicates with remote nodes. Other issue is restrictive usage to allow only certain devices that matches user device should be accessible. For example: if devices are available in DHT based network stopping George device to be discoverable by Alice device is a problem. George device to be searchable by only George's device, is not clear and how to do it needs to be defined in order to use DHT in secure way.

Major problem in DHT are change in node information becomes difficult to address a node, node acting as bootstrap needs to be available all the time, and controlling namespace for the key generated. In

webinos context, DHT could be used to find peers which provides information of services available and will be further investigated in Phase II.

Web Finger

The WebFinger is protocol inspired by the old Name Finger protocol defined in [RFC 742]. The Name Finger protocol enables the possibility to get information about a given user. WebFinger is an evolution that instead of using a direct TCP connection, it uses HTTP [RFC2616], XRD and Web Host Metadata [IETF, draft-hammer-hostmeta-16] to provide a descriptor of a single user. The protocol consists of two parts:

1. One URI schemes to identify user accounts, e.g. acct:joe@example.com
2. A simple protocol for resolving a user account into an extensible descriptor formated as a XRD resource.

The protocol is not an official standard but the work has been driven by several parties involved Identity Commons, which is a community of groups working on developing the identity and social layer of the Web.

Web Introducer

It addresses finding services that user has registered on Web. For example, it allows user to connect with his choice of a photo sharing website; all the information is present with registrar and it provides information for communication of services.

It is quite relevant for webinos to allow addressability of different resources that are available over the Web for a particular user and could be part of personal hub.

Recommendations from state of the art

Local Discovery

The diversity of internetworking technologies for local discovery introduces a variety of discovery protocols and implementations. Our literature review and hand-on demo work recommend the use of SSDP defined in UPnP and ZeroConf for local IP network discovery due to their efficiency and popularity. For other devices that don't support UPnP and ZeroConf, specific discovery mechanisms shall apply, e.g. standard Bluetooth HCI inquiry and SDP for Bluetooth device discovery.

Remote Discovery

XMPP is a relevant technology as it supports finding friends, their resources, items and services. It supports publish and subscribe mechanism, event mechanism, and also serverless messaging for local area network.

The webinos platform could utilize XMPP core and its extensions. XMPP core specification allows device connectivity and establishing secure communication. XMPP is mostly considered as chatting protocol but for webinos platform, XMPP specification/extension considered are service discovery, getting node information, and resource information.

Use webfinger to discovery Personal Zone Hubs (PZH). The PZH will be identified by a URI and this will introduce another personal address. However by using webfinger it will be possible to leverage on existing identities like an e-mail address, Facebook identity or Google Id which can be obtained from many different sources like the local contact book, social graph or from business cards. The end user does not need to be aware of the PZH URI.

Messaging

The webinos architecture features a powerful and extensible messaging framework that allows to easily exchange arbitrary data, in terms of *events*, among addressable *entities* (e.g., applications, services), also completely hiding away any complexity related to the different underlying interconnect technologies.

It is based on a flexible, yet rigorously defined, event description that is both independent of the actual payload data format and serialization format for data transmission. This basically means that custom event-based protocols can be easily defined and implemented and that it is possible to choose the "encoding" that better suits the involved interconnect technology.

Despite the swiss-army knife-like nature of this system, at a basic level the handling of events reduces to just a few simple concepts from the developer perspective: generating and sending events, or forwarding them, and registering listeners for incoming events. More advanced features are also offered, including, but not limited to, the possibility to send/forward events to multiple destinations at once, to associate event listeners to a particular event source, destination and/or type, to specify a time frame for event delivery, to ask for delivery notifications and to control/monitor the storing and forwarding of events that cannot be immediately delivered.

Furthermore, two more specific protocols are defined on top of this low level generic framework, one regarding event delivery notifications and another describing RPC functionality needed to implement webinos services.

What's in scope

- Event description: what a generic event looks like from the developer's point of view, which metadata is compulsorily or optionally associated to each event;
- Event processing: how generic event metadata influences the sending, caching, storing, forwarding and listening to events;
- Application-, device- and network-level event routing: how the event handling mechanism interfaces to other parts of the webinos architecture to allow event exchange;
- Event delivery notifications and RPC protocols: what they are, how they work, what their relationship with the generic event handling mechanism is.

What's out of scope

- Interconnect technology-dependent details of data transmission over the network;
- Discovery and binding of addressable entities;
- Definition of special-purpose event-based protocols;

- User-visible notifications and user interaction.

Review of State of the Art

The XMPP core protocol and some of its extensions (also known as XEPs) have been analyzed as today's state of the art technology for generalized routing of data.

Formerly known as Jabber, XMPP is an open, decentralized and extensible protocol for near-real-time XML data exchange; it is backed and formalized by IETF (RFCs 3920-3923, 4854, 4979, 5122) and further developed by the XMPP Standards Foundation, with several mature implementations already available.

Our XMPP for Event handling state of the art analysis documents clearly outlines how it would be possible to satisfy most webinos' functional requirements concerning remote notifications and messaging by simply adopting XMPP and requiring a specific set of XEPs to be supported by the implementations; furthermore, such a choice would also allow to reuse at least part of the already existing XMPP server-side infrastructure without modifications.

On the other hand, the scope of such analysis is strictly limited to the exchange of structured data (i.e., events) and does not take into account issues that are of fundamental importance in other functional areas.

Recommendations from state of the art

Given XMPP's maturity and suitability for applications in many different contexts, the webinos' event handling system will borrow a consistent set of concepts, features and technical solutions from such technology.

Such a strategy should also be of help in defining a bidirectional mapping between the two technologies for interoperability purposes, yet without creating unilateral or mutual interdependencies.

Context

The Context area addresses all issues relating to management of contextual information (detection, acquisition, representation, distribution, etc) as well as all the potential consequent capabilities (such as content Adaptations and Reasoning) that could enabled by being aware and process this information within webinos.

Through webinos users will be able to access and use applications that work across devices allowing them to have an uninterrupted usage experience. Such a capability will eventually propagate activities, events and even connections that users maintain to be expressed also through the set of owned devices too. For example sharing a piece of multimedia (photo, photo-album, a playlist) with another person within the scope of using one application across several devices.

The innovation of the webinos approach in context framework is that it structures the context data that occur from these activities/events that are performed through connected devices in a way that could "make sense" and make this information available in a privacy preserving way to support the creation or context aware application that can provide a better user experience.

What's in scope

The webinos context framework comprises the following points:

- The Context Architecture, outlining the basic component of their interconnections. The architecture outlines how context data are acquired through context related events that occur in the system and are made available through the system APIs as well as how these activities are integrated with the overall webinos Privacy architecture (through a policy enforcement point).
- An analysis of how contextual structures are formed within the context framework and an outline of some fundamental context structures such as the User context, Device Context, Application Context context objects.
- An analysis of technologies to implement the Storage and Extensibility Framework for the Context Model - where different part of the context model are stored, for example user context in the cloud, device context in the device and how these are linked among them.
- APIs for Context Access through two basic mechanisms, Querying for Context Data and Subscribing to Context Related Events.

Whats out of scope

Currently the following areas are out of the scope of the webinos context framework:

- General data management and storage within webinos.
- User ID management, privacy and security specificaiton within webinos.

Review of State of the Art

Context awareness and adaptation constitutes quite an extended area with regards to underlying state of the art. The following dimensions have been examined within the context awareness activities:

- EU research projects that deliver relevant specifications/prototypes.
- Existing and emerging standards that can enable context awareness and adaptations functionality with a focus in representing social activity.
- Underlying technologies, academic research and prototypes coming either within or outside of the consortium.

Recommendations from state of the art

The underlying state of the art - particularly from dedicated context projects - reveal some design patterns when it comes to designing context oriented solutions, specifically:

- Context is tightly related with the occurrence of events that signify the presence (or existence) of a situation.
- Context data refer both to present moment but also to past (or even future) moments. This means that is is necessary to provide a storage facillity that holds not only current but also history context data.
- Acquisition, Access or Reasoning of Context Data should take into consideration the User preferences, empowering the user to control or define these activities.

All the above points have been taken into consideration in designing and later implementing the webinos Context Framework.

Security and Privacy

One of the primary aims of webinos, and of future internet projects in general, is to provide a secure, privacy-preserving internet experience. There are many well-documented problems with security on Web applications and the Web in general, including weak authentication [BICH11] and numerous forms of content injection attacks [OWASP10]. Furthermore, mobile devices contain enormous amounts of private and confidential information, the protection of which is paramount for both business and home users. The webinos project has many of the same problems and, by creating a joined-up cross-device application infrastructure, it could be argued that there was an increased potential for harm: attackers - such as the webinos personas Ethan [D027-Ethan] and Frankie [D027-Frankie] - could potentially steal valuable data or the end user's identity on every device they own. Furthermore, webinos has multiple stakeholders with different security requirements. Some of these produce contradictions, for example a developer - such as the Jimmy persona [D027-Jimmy] - may want to find out demographic data and take advantage of analytics to profile users, whereas users such as Helen [D027-Helen] wish to preserve their private information. These factors, as well as the diverse number of devices that may be supported by webinos, means that a sound security architecture is of vital importance to the webinos system.

However, the security architecture is also an opportunity to make a significant contribution to the current state-of-the-art in mobile application security and privacy. By introducing a standardised and robust security framework, webinos can potentially increase security and privacy on the four device domains simultaneously. Part of this is due to the fundamental webinos vision of creating a standardised application environment: by providing a unified user interface for making access control decisions, webinos will significantly increase usability and therefore encourage users to make better (and more privacy-friendly) security decisions. This is one place in which existing application architectures are fragmented, as the major mobile operating systems such as iOS and Android have different security models, making the experience less familiar and potentially discouraging users from expressing their privacy preferences. The current systems have also been criticised for having an "all or nothing" approach, with Android and iOS requiring that applications are installed with access to all features the developer asked for, or not installed at all. This has proven unpopular with users, with alternative Android systems appearing which offer the revocation of privileges [DEME11). It has also been noted that many applications request more privileges than they need. This means that users are not as cautious of applications which request many privileges as they should be. Webinos is well positioned to provide better solutions than the current state of the art.

The webinos security and privacy architecture is fully outlined in deliverable [D035) but the essential functional components of the policy enforcement mechanism are explained in this document. Implementing a policy enforcement mechanism requires several novel features, including:

1. Support for flexible access control policies referring to all APIs and data sources on the webinos platform

2. Policies which can refer to cross-device interaction, both inside a webinos "personal zone" and between users with no prior trust relationship

3. Synchronisation of access control policies between devices within the webinos personal zone

4. Connecting application requests to privacy policies, so that users can make well-informed decisions about their personal privacy.

The security policy system builds on work from WAC [WAC] and BONDI [BONDI] and is based on the XACML language and architecture [XACML].

What's in scope

This document describes the policy architecture for webinos, and covers the following topics:

- Access controls for applications. This includes permissions for:
    - Device APIs, such as features, location services and cameras
    - Other devices, both inside and outside of the personal zone
    - Remote content and services
    - Personal profile data
    - Other applications on the same device and on other devices
- Policy synchronisation between a users' devices
- Privacy data-usage policies and obligations for applications
- Application trust chains and certificates

Details on authentication and user identity management are also included in this deliverable in later sections.

Whats out of scope

The following issues are not included in this document, but are covered either in another deliverable D03.5 or in the second phase of webinos specification:

- Remote management of devices and remote policy enforcement (phase 2).
- Implementation details for the protection of applications and the webinos runtime during use (recommendations in D03.5, further details in phase 2).
- Platform integrity reporting and attestation (specified in D03.2 and discussed in D03.5)
- Integration with social networking for more usable policies (phase 2)
- User interface specifications for policy editing and resolution (phase 2 and D03.5)
- Digital rights management and content protection (phase 2).

- Detailed specifications of privileged applications (D03.5)

Review of State of the Art

There is a great deal of existing work in access control in general and mobile platforms specifically. A comprehensive summary of related work to security is covered in deliverables D02.7, D03.5 and D03.6.

Recommendations from state of the art

The security architecture, much like the rest of the webinos specification, has been designed to reuse as much existing technology as possible. This is particularly important in security, as creating new designs and writing new code will introduce new design flaws and vulnerabilities. Many existing solutions have already undergone extensive testing and will have been patched to fix many outstanding issues. Therefore, we have built primarily on the existing WAC specifications and the general XACML architecture. From the analysis, we can see that these already solve many problems in webinos - such as mediating access to device features - but must be modified to support new requirements.

However, we can also improve on WAC designs by implementing features such as privacy policies which remained underspecified. We propose to take advantage of the work produced by the PrimeLife project to create usable policies which protect user privacy.

Privileged Apps

The scope of this section is to provide Access Control or Privileged Apps and Services specifications. The objective of this section is to recommend a security solution for implementation using the Privileged Apps and Services concept in webinos project. The use of the concept of Privileged Apps and Services is an important factor in webinos. A webinos application will be signed with a certificate that is in the privileged certificate store on the device. Target an application based on its Digital Certificate and there shall be policies assigned to these applications.

Privileged applications are those apps that request additional capabilities, e.g. access to a location or to restricted data such as your contacts, vehicle engine details. Although these kinds of applications also require access to special APIs like an automotive or home media app and the access to these APIs has to be granted by administrator or a privileged user, these applications shall focus on the management of the webinos runtime.

A privilege management creates, stores, and manages the attributes and policies needed to establish criteria that can be used to decide whether a user's request for access to some resource should be granted. Access control uses the data made available by authentication, privilege management, and other information provided by the access request provider, such as the form of access requested to make an access control decision.

The design principles for the privileged architecture in this section are:

1.) Guarding against threats that access critical data.
2.) Establishing levels of security for data and other resources by using Policies.
3.) Implement dashboard, installer, launcher, and policy manager.
4.) Allow direct control over API access by the API provider. E.g., a car manufacturer can write engine monitoring APIs and allow them to access only via the car manufacturer signed applications. Let there be a signed certificate, Identity and Integrity checks for widget based Apps. Protection and Security from hacks at the runtime and accessing sensitive API's. To support these Cryptographic methods, Encryption code be used. The Apps are signed and confirmed by the Device Manufacturer when using Sensitive API's and Critical data, the Monitoring system checks and manages that there is no access to critical data like Engine Diagnose API's and HW data.
5.) The Privileged Apps and Services shall provide information related to Date, Event ID, Event Description, Username, Parent PID, Policy, Application Group, Reason, Custom Token, Filename/Codebase, Type, Instances, Description, and Certificate.

Two ways of using Privileged Apps and Services in webinos for security purpose:
1.) Enforce access control policies at the Runtime Environment.
2.) An Application which uses system commands and classes which manages the OS services, access rights, registries, roles based on the users and so on.

State-of-the-Art

In the state-of-art analysis we are going to evaluate different solutions for Privileged Apps and Services(Access Control) in webinos such as W3C, WAC , Android and BONDI
Privileged Application in JavaScript and provide a recommendation, which solution shall be incorporated into the webinos runtime.

The working of XACML with Privilege Apps and Services (Access Control)

The deployment of the XACML access control system **SHALL** work:

• A User seeks access to some resource and submits a query to the entity (Policy Enforcement Point (PEP)) protecting the resource.
• The PEP forms a request (using the XACML request language) based on the attributes of the subject, action, resource, and other relevant information.
• The PEP then sends this request to a Policy Decision Point (PDP) that examines the request, retrieves policies (written in the XACML policy language) that are applicable to this request, and determines whether access should be granted according to the XACML rules for evaluating policies.
• The answer (expressed in the XACML response language) is returned to the PEP, which can then allow or deny access to the requester.

Policy Language and Enforcement

- The implementation of a policy system requires to choose algorithms for reconciling conflicting policies. It should independently administer multiple policies controlling access to the same resources.
- An efficient way of locating all the policies that are potentially applicable to a given decision.

Authentication and Authorization

- Grant security properties, like authorization and access control like what resources the user can access.
- The logging of the Authentication to the personal zone (user authentication with the personal zone hub). The Notification and keeping track of the Personal zone identity and personal zone proxies.
- Running retrieve data in privilege app space.
- Updating user credential information such as password, certificates.
- Enable access to recorded decisions when the user isn't available in real time.

Authorization and Privileg

- Common authorization model for all the trust domains.
- Common language for expressing security policies.
- Support of authorizations at all levels of granularity.
- Storing Authorization in a safe and protected place if they are not digitally signed
- Identify applications which have been granted particular privileges.
- List of all their webinos applications for the users.
- Restrictions of the access control policies on applications from potentially malicious applications.
- Ensuring that only trusted components are downloaded
- Delegate decisions to a trusted third party when appropriate.

Discovery

- The access control should check whether the address, devices or services are valid or not.
- If service driver is required to be installed for the device, privilege application should support the driver.

- Device visibility control, device in multicast mode could be passive or active listener.
- Access control to access different file system area and obtain user credentials information.
- Specify a access format.

Context

- Defining Policies to access his (photo, photo-album, a playlist) and other stuffs across other webinos devices.
- Grant and retrieve the data and the Policies based on Context.
- Storing of device context in file system.
- Review and manage which applications users have granted permissions to, and in what context.
- Policies based on Subjects and Resources

Tasks in the scope of Privilege Apps and Services

The PZP can handle many devices and multiple users. So there should be certain level of permissions enforced to on a particular user for viewing, editing files, modifying system files. Similarly, there may be certain Web apps which would try to access the restricted registry files, drivers or at the kernel level.

So the owner of the device can permit privileges to delete files, view private information, or install unwanted programs.

Most Privileged

When the user or process is able to obtain a higher level of access than an administrator or system developer intended, possibly by performing kernel-level operations

• An attacker may then be able to exploit this assumption so that unauthorized code is run with the application's privileges.

• Some services are configured to run under the Local System user account. A vulnerability such as buffer overflow may be used to execute
arbitrary code with privilege elevated to local system.

• Any user which accesses binary in the file system or Registry can therefore elevate privileges.

• Core dump be performed in case it crashes and then have itself killed by another process.

• Cross Zone Scripts should be identified so that the running of the malicious code on the client side can be prevented.

Least Privileged

Least Privileged: An application allows gaining access to resources that normally would have been protected from an application or user. The application would perform actions but different security context than intended by the application developer or Administrator.

RBAC - Role Based Access Control

RBAC is an approach to restricting system access to authorized users. The permissions to perform certain operations are assigned to specific roles. The webinos shall provide a RBAC model where the Privileged Users are assigned particular roles, and through these role assignments acquire the permissions to perform particular system functions. Since users are not assigned permissions directly, but only acquire them through their role (or roles), management of individual user rights becomes a matter of simply assigning appropriate roles to the user; this simplifies common operations, such as adding a user devices, or changing a user's role.

Three primary rules are defined for RBAC:
1. Role assignment: A subject can execute a transaction only if the subject has selected or been assigned a role.
2. Role authorization: A subject's active role must be authorized for the subject. This rule ensures that users can take on only roles for which they are authorized.
3. Transaction authorization: A subject can execute a transaction only if the transaction is authorized for the subject's active role. With rules 1 and 2, this rule ensures that users can execute only transactions for which they are authorized.

Areas to Consider

- If a subject has roles R1 , R2, ... Rn enabled, can subject X access a given resource using a given action?
- Is subject X allowed to have role Ri enabled?
- If a subject has roles R1 , R2, ... Rn enabled, does that mean the subject will have permissions associated with a given role R'? That is, is role R' either equal to or junior to any of roles R1 , R2, …Rn?

Access Control Matrix

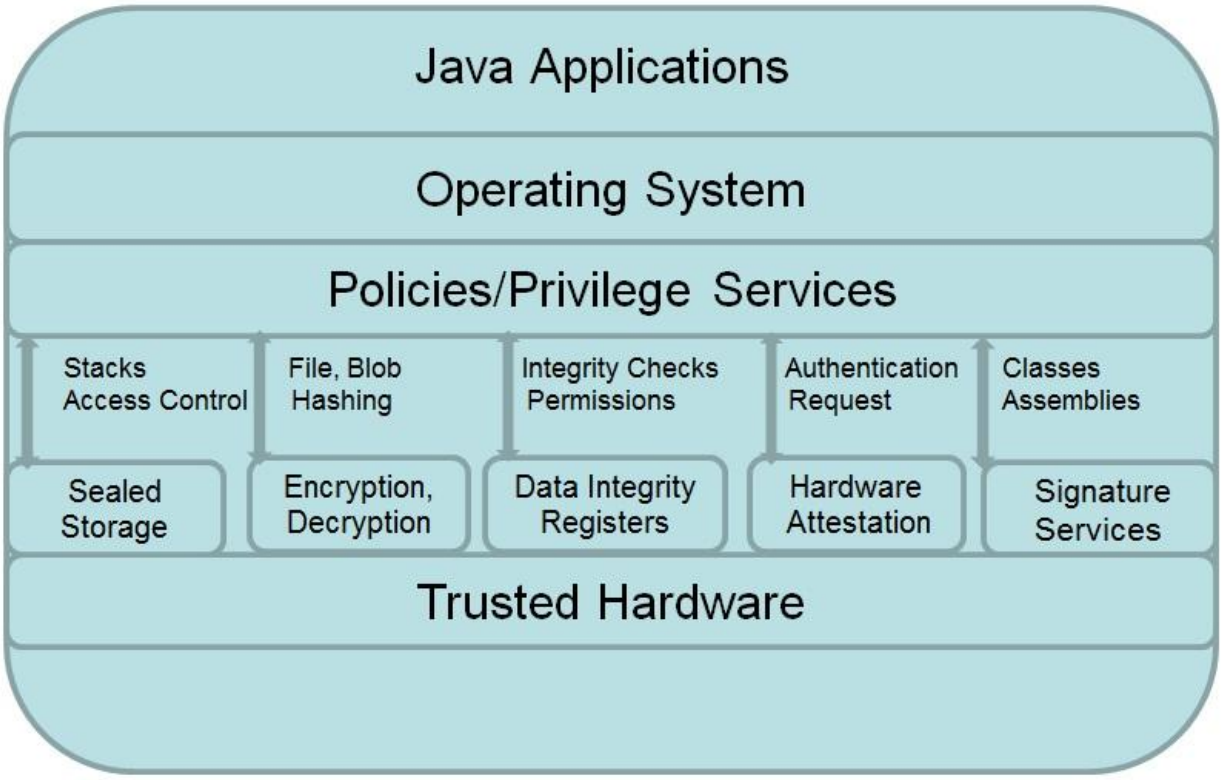
It would be important for webinos to include the Access Control Matrix it is a useful model for understanding the behavior and properties of access control systems. This matrix defines the trust relationships between the control domains and sub-domains. The implementation of the access control matrix can be based on a combination of Access Control Lists, permission files, and an enforcement engine, such as Java’s Security Manager and Access Controller. Policies are based on the relationships defined in the Access Control Matrix.

Types of accesses that are necessary to define the relationships between the objects and subjects in webinos:

1. File Access, which includes the following permissions: Read, Write, and Execute.
2. Message Access, which is necessary because of the need to control the exchange of messages between trusted and non-trusted domains and subjects. Message Access includes the following permissions: Send and Receive.
3. Process Access, which controls the start and termination of processes such as Core Software Download, including the following permissions: Initiate, and Terminate.
4. Key Accesses. This access type includes Create and Use

Examples showing the Policy Enforcement Point in Mobile Platform and in the High Level Vehicle Bus Infrastructure

Policy Enforcement Point in Mobile Platform:



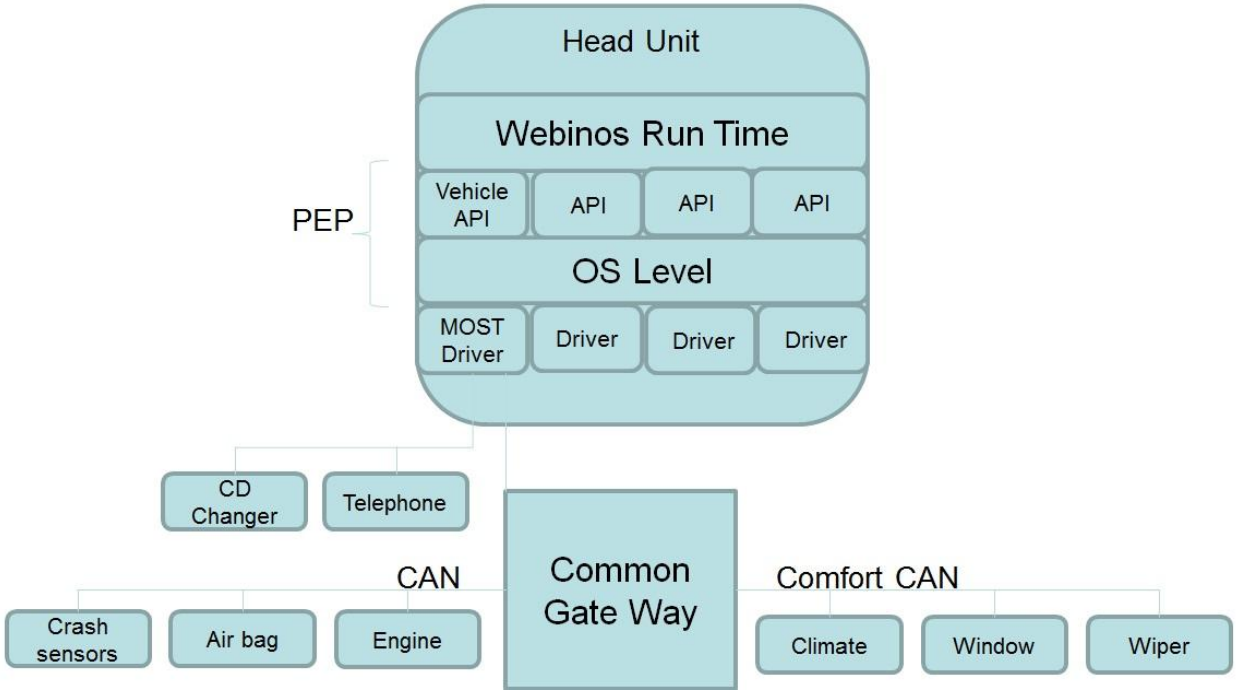
The security features and the policies supported by the Mobile device are enabled or enforced in the Policies and Privilege services layer, which extends Java's 'SecurityManager' and 'AccessController' classes. At runtime, the Policies and Privilege services layer decides what policies are to be enforced and what privileges can be applied when a connection request is made. Based on the domain, device, device type and whether the domain or device is trusted or un-trusted, the Privilege layer can enforce application level authentication, encryption of the session and any other specific access policies. For the

Information regarding the services that the decision is to be are available in the manifested files and are described in hash of Policy files that are stored securely.

The Privilege services layer provides security services:

- The Privilege Monitor stores event logs in its registers for auditing purposes.
- The Privilege Apps and services will provide cryptographic services, to include asymmetric key generation, digital signatures, hashing, and encryption.
- The key used during access control can be securely stored using the Certificate store's Secure Key Storage capabilities.
- The runtime Privilege Apps auditing functions will make full use of the registers, Secure Data Storage, and Session Storage capabilities.

Example for Policy Enforcement Point in High Level Vehicle Bus Infrastructure for IVI:



This example illustrates the Policy Enforcement Point in a High Level Vehicle Bus Infrastructure. The in-car headunit is basically an in-car PC connected to the infotainment bus. All infotainment relevant control units such as the cd changer, telephone, gps module are connected to this bus and can communicate with each other on this bus by sending messages. At BMW MOST driver is being used as the infotainment bus (for more information on MOST Bus see this link: http://en.wikipedia.org/wiki/MOST_Bus. This infotainment bus is connected to the Common Gateway (CGW). To this central gateway all other vehicle buses (e.g., High speed CAN or comfort CAN) are connected as well. At the CGW some messages from the 'Comfort-CAN' and 'High Speed CAN' (e.g., speed, wiper status, climate) are converted to MOST messages and routed into the MOST bus.

The MOST bus transports control data as well as data from audio, video, navigation and other services. MOST technology provides a logical framework model for control of the variety and complexity of data. The MOST Application Framework organizes the functions of the overall system. MOST is able to control and dynamically manage functions that are distributed in the vehicle.

There are two places to enforce the access to the vehicle data. We can place the enforcement point inside the webinos runtime: When an app calls a specific vehicle function, the runtime checks back with a PDP, if the access is allowed or not. If allowed, the request is pushed to the OS service to create a MOST message and put it onto the bus. At the OS service (before we built the MOST message for this request) we could also check back with the PDP, if the access is allowed or not.

Technical use cases

This section includes the Technical use cases and requirements identified from the WP2.1 and WP2.2 in the area of Privileged Apps and Services.

User Stories, Use Cases Identified

Related User Stories

WOS-US-7.1: Designing Policy-aware webinos Applications
WOS-US-7.4: Privacy Controls and Analytics for Corporations and Small Businesses

Related Use Cases

- WOS-UC-TA8-002: Interpreting policies and making access control decisions
- WOS-UC-TA8-003: Enforcing multiple policies on multiple devices
- WOS-UC-TA8-007: Policy authoring tools
- WOS-UC-TA4-013: Dynamically Sharing Content with other Users in a Controlled Manner
- WOS-UC-TA6-00X: Checking access to APIs – Refers to Content Adaption
- WOS-UC-TA1-008: webinos Federation
- WOS-UC-TA4-014: Continuous sharing of a medical file through webinos enabled devices
- WOS-UC-TA7-008: Create contexts from a pre-defined template

This section of the specification aims to satisfy the following requirements

- PS-USR-Oxford-50
- PS-USR-Oxford-51
- PS-USR-Oxford-116
- PS-DEV-ambiesense-08
- PS-USR-TSI-4
- PS-DWP-ISMB-202
- PS-USR-Oxford-35
- PS-USR-Oxford-38
- PS-USR-Oxford-115
- PS-USR-Oxford-72
- PS-USR-Oxford-36
- PS-USR-Oxford-34

- PS-USR-Oxford-5
- PS-USR-Oxford-17
- PS-DEV-Oxford-28
- PS-USR-TUM-*(124)

Privileges and Access Control Use Case and Requirements identified

Policy management, authoring and usage features

ReqID	Requirement	Notes	Use Case Refs	Review	Architecture	Priority
PS-USR-Oxford-50	Users **SHALL** be provided with the ability to identify applications which have been granted particular privileges		WOS-UC-TA9-006		UI, Policy layer	
PS-USR-Oxford-51	Users **SHALL** be able to view a list of all their webinos applications and show the authority that certified the application		WOS-UC-TA9-006		UI, Policy layer	

Runtime Protection:

ReqID	Requirement	Notes	Use Case Refs	Review	Architecture	Priority
PS-USR-Oxford-116	The webinos runtime environment **SHALL** protect applications and itself from potentially malicious applications and **SHALL** protect the device from being made unusable or damaged by applications. The webinos Runtime Environment is a naturally privileged process that should be strongly protected from applications. Furthermore, it must prevent applications from misusing device capabilities when they run.				WRE, APIs	Phase 1
PS-DEV-ambiesense-08	The webinos runtime environment **SHALL** support customised encryption of any data stream (independent of its data type or format) The main threat is anyone seeking the information/ data transferred in the data stream		None		WRE	Phase 2
PS-USR-TSI-4	webinos shall ensure that only trusted components are downloaded, and that applications are guaranteed some level of execution (to prevent				Policy later, App manifest	

	from denial of service) Device integrity – prevent malware compromising reliability + conf. + availability. QoS. This implies knowledge of the components that are trusted?					
PS-DWP-ISMB-202	The webinos runtime **MUST** ensure that an application does not access device features, extensions and content other than those associated to it.	Moved from LC	WOS-UC-TA6-00X: Checking access to APIs		WRE, APIs	

Policy management, authoring and usage features

ReqID	Requirement	Notes	Use Case Refs	Review	Architecture	Priority
PS-USR-Oxford-35	webinos access control policies shall be able to specify fine-grained controls involving the source and content of an access control request	This implies that application instances are identifiable	* WOS-UC-TA9-002: Interpreting policies and making access control decisions		WRE, Policy layer	
PS-USR-Oxford-38	webinos **SHALL** allow policies which specify confirmation at runtime by a user when an access request decision is required	See WAC.	* WOS-UC-TA9-002: Interpreting policies and making access control decisions		Policy layer	

Application policies and protection

ReqID	Requirement	Notes	Use Case Refs	Review	Architecture	Priority
PS-USR-Oxford-115	webinos **SHALL** encourage good design techniques and principles so users are not forced to accept unreasonable privacy policies and access control policies. webinos Applications **SHALL** be designed with user policy negotiation and preferences in mind.				APIs, Apps, Dev tools	

PS-USR-Oxford-72	The webinos System **SHALL** support applications which apply access control policies to data produced or owner by the application developer. These policies **MAY** support revocation of access control permissions		WOS-UC-TA4-013		WRE	Phase 2
PS-USR-Oxford-36	webinos APIs shall provide error results when an access control request is denied Developers **SHALL** be aware of how to program for graceful handling of access control requests.		* WOS-UC-TA9-002: Interpreting policies and making access control decisions		WRE, Policy layer	Phase 1
PS-USR-Oxford-34	webinos shall provide complete mediation of access requests by applications and enforce all policies		* WOS-UC-TA9-002: Interpreting policies and making access control decisions		WRE	Phase 1

Device discovery, communication and authentication

ReqID	Requirement	Notes	Use Case Refs	Review	Architecture	Priority
PS-USR-Oxford-5	The level of authority associated with a client webinos device **SHALL** be established before an association is established with a webinos cloud. How is authorisation and access control defined by webinos?		* WOS-UC-TA1-008: webinos Federation		Policy layer, Comms	
PS-USR-Oxford-17	The webinos Runtime Environment **SHALL** be capable of setting dynamic access control policies for device data when initiating an association to another webinos Device. What format do these access rules take?		* WOS-UC-TA4-014: Continuous sharing of a medical file through webinos enabled devices		WRE, UI, Policy	

Sharing and protecting personal and contextual data

ReqID	Requirement	Notes	Use Case Refs	Review	Architecture	Priority

PS-DEV-Oxford-28	The webinos Runtime **SHALL** provide access control for context structures with user-defined policies		* WOS-UC-TA7-008: Create contexts from a pre-defined template		WRE, API, Policy layer	Phase 1

Privilege apps for Device Manufactures

ReqID	Requirement	Notes	Use Case Refs	Review	Architecture	Priority
PS-USR-TUM-*(124)	webinos **SHALL** provide privileged apps and services to support the trust based factor for the device manufacturers, for the applications that access wide range of critical information from vehicle data, mobile, setupbox will have to be approved by the manufacturer of the device	Let there be a signed certificate, Identity and Integrity checks for widget based Apps. Protection and Security from hacks at the runtime and accessing sensitive API's. To support these Cryptographic methods, Encryption code **SHALL** be used. The Apps **SHALL** be signed and confirmed by the Device Manufacturer when using Sensitive API's and Critical data, there **SHALL** be a Monitoring system which checks and manages that there is no access to critical data like Engine Diagnose API's and HW data.	* WOS-UC-TA8-002: Interpreting policies and making access control decisions		Comms, WRE	

Note:

- The geolocation could possibly be also provided by the vehicle API, but we have already the Geolocation API.
- Applications using the vehicle API have to be approved by the manufacturer of the vehicle/Device. If the application is not approved, then the application cannot access the vehicle API.

Analytics

Definition of analytics

Analytics is the epitomy of Business Intelligence (BI) science born by the overwhelming wealth of information in the cloud and the shockingly detailed digital trail left throughout the user journey. It is a mix of advanced statistics and data mining techniques which combine, homogenize and translate this wealth of digital information from disparate sources, into actionable business intelligence.

Applications of Analytics solutions include:

- hardware/network/application troubleshooting,
- hardware/network/application performance optimization,
- usability testing,
- Security analysis/forensics
- marketing:
  - usage tracking
  - recommendations,
  - targeting,
  - campaign performance tracking & management,
  - sales tracking

Analytics solutions are also categorized based on the available measuring/probing points: device (on chip), network (transport, IP & DPI), Web (http), application (in app), app store (sales). Since the webinos platform will reside between the transport layer of the network and the actual application (as part of the Web run-time) the closest category is "application (in app) analytics".

Key issues & challenges of analytics solutions

- visibility: an analytics solution is inherently limited by the data that can be collected e.g. an application has no "visibility" about what the user is doing with other applications.
- device issues: caching (device caching consumes memory), processing cycles and device battery,
- data homogenization across devices and services: different devices and platforms produce different kind of data that need to be "normalized" in order to be comparable.
- privacy issues of data gathered
- data ownership issues of analytics (who owns the insights about the users data?)

It is also important to mention that the value (and main challenge) of an analytics solution is in the actual analysis (define meaningful and actionable 'recipies') & reporting. This however depends on the

actual BI application. Therefore, at platform level, the aim is only to provide the means to easily collect and log information.

To this end, webinos is a game changer from existing platforms in that it enables:

- a uniform interface for extraction of information across many different device domains; car, pc, home media, mobile
- a uniform interface for extraction of information across service domains.
- webinos has a strong security framework which provides a solid mechanism for ensuring that privacy control is in the hands of the user.
- webinos authentication mechanism enables tracking of usage across devices / services without the need for separate logins

The combination of these enablers give webinos a big advantage of end-to-end visibility across devices, services, networks and usage contexts.

Analytics solutions high level architecture/work-flow

- define metering rules (what/when/how to capture)
- deploy metering rules
- capture/metering based on metering rules and privacy/security settings
- transfer and store data to (log)
- homogenization (if necessary across devices)
- transfer to analysis engine/repository
- production of actual analytics processing - report

Note: The word *Metering* refers to the low-level process of collecting/measuring & recording data points and event triggers, i.e. without any kind of further processing,
normalization or analysis. *Metrics* are therefore the directly measurable events and data points. *Analytics* refer to the product(s) of the statistical analysis of those metrics.

The architecture can be divided in three parts based on different sets of implementation requirements:

- platform: definition and deployment of metering rules, data capture and local (within personal network) storage
- implementation: data homogenization and transfer to analysis engine (and potentially to external repository)

- application dependent: data mining and statistical analysis for the generation of business intelligence and reporting

To this end, WP3 activities are only concerned about **platform** related issues. **Implementation** and **Application dependent** issues may be investigated as part of a WP5 proof-of-concept application implementation.

References to requirements

The need to provide support for analytics at platform level has been identified in the following webinos requirements (see Deliverable D02.1 Use cases & requirements).

ID	Description
WOS-US-7.4	Privacy Controls and Analytics for Corporations and Small Businesses
WOS-UC-TA8-013	Collecting Analytics from webinos Applications
WOS-UC-TA8-014	End User cross Platform Privacy Analytics in Healthcare, Smart Grids and Home Environments

What's in scope

- Identify an initial list of "meterable" data points and events based on existing list of APIs
- define metering rules (what/when/how to capture)
- define necessary privacy/security policies
- data capture/metering based on metering rules and privacy/security policies
- deploy mechanism for metering rules (how is the metering client/logging mechanism getting updated with new rules over-the-air)
- transfer and store data to (log)

What's out of scope

Deferred to second phase:

- provision for complex metering rules which may involve more than one-state, data point (pattern/sequence signatures)

May be implemented as part of proof-of-concept applications (WP5):

- homogenization of data across device and services (if necessary)
- data transfer to analysis engine/external repository

- implementation of an actual analytics processing & reporting engine

Review of State of the Art

Existing analytics platform provider are split broadly in two categories: Web and application-based. For mobile telecoms three more categories exist, namely: mobile messaging, SIM and network.
The categories are defined based on the metering software touch-point, i.e. its visibility in terms of available data and event triggers.

Here follows an overview of some prominent commercial *application* analytics solution providers:

Distimo http://www.distimo.com/

Main proposition *e.g. application analytics SaaS, custom reports, app store analytics - tag line*

App Store Analytics (no device client)
Distimo Report custom reports aimed to companies providing insight into trends happening within application stores.
Distimo Monitor free analytics tool for developers to monitor their own and competitive applications across all app stores

Core product *e.g. application analytics SaaS, custom reports, app store analytics*

(App Store Analytics - no device client)
Distimo Report: custom reports aimed to companies providing insight into trends happening within application stores.
Distimo Monitor: free analytics tool for developers to monitor their own and competitive applications across all app stores

Platform targets *e.g. iOS, Android, or app stores supported*

Distimo Monitor: Apple, Android already; Blackberry, Nokia Ovi in 2010
Distimo custom reports are currently available for the Apple App Store for iPad, Apple App Store for iPhone, BlackBerry App World, Google Android Market, Nokia Ovi Store, Palm App Catalog and Windows Marketplace for Mobile.

Data ownership and privacy
n/a

Type of analytics *e.g. app store, in-app, billing, Web*
app store

Bango http://bango.com/

Main proposition *e.g. application analytics SaaS, custom reports, app store analytics - tag line*

Mobile Billing: in-app and mobile websites (credit card, paypal, in-bill)
Mobile Analytics: mobile Web and campaign analytics, in-app analytics

Core product *e.g. application analytics SaaS, custom reports, app store analytics*

Mobile Billing: in-app and mobile websites (via credit card, paypal, on-bill)
Mobile Analytics: mobile Web and campaign analytics, in-app analytics (unique users, avg session length, avg sessions per day traced by specific Web calls).

Platform targets *e.g. iOS, Android, or app stores supported*

Libraries for probing on the application side and Web-API are provided.

Data ownership and privacy

Data ownership is offered for a premium. Data are accessed by all subscription packages with timeframe limitations.

Type of analytics *e.g. app store, in-app, billing, web*

web; in-app; billing (via own solution)

AT Internet http://atinternet.com/

Main proposition *e.g. application analytics SaaS, custom reports, app store analytics - tag line*

Web/ecommerce/mobile/social networking analytics;
tagline: How to help with the challenges of customer acquisition, transformation and retaintion.

Core product *e.g. application analytics SaaS, custom reports, app store analytics*

Service analytics: through reports provide information like connection type (wify/network) what's the speed , network provider;

Mobile and Campaign: originally was only focused on mobile websites, last couple of years developed solutions to provide app level (usage) e.g. number of times used, number of crashes, navigation through app, popular pages, offline data and how is being used.

Social media: BuzzWatcher measures activity on social media channels (including social networks, video platforms, RSS feeds, blogs etc,) in real time.

mobile nx ""module"" is the web-side component which integrates with the ""digital workspace"" dashboard (server performance, Web analytics, social media, mobile analytics). Libraries for probing on the application side are provided.

Products are defined by what data collection method you use (i.e. probing points) - mobile is: tags for mobile sites, behavior, apps, purchase and offline usage info

Platform targets *e.g. iOS, Android, or app stores supported*

in-app analytics: Symbian, iOS, Blackberry, bada, Android.

Data ownership and privacy

n/a

Type of analytics *e.g. app store, in-app, billing, web*

web; in-app; e-commerce (via own solution)

Flurry http://www.flurry.com/

Main proposition *e.g. application analytics SaaS, custom reports, app store analytics - tag line*

application analytics
tagline: aiming to make the life of developers and publishers easier.

Core product *e.g. application analytics SaaS, custom reports, app store analytics*

Analytics: core free in-app analytics solution with libraries, Web dashboard and reporting interfaces, and API access to the data.
Analyzer Mobile: targeting service for applications;
Appcircle: affiliate network for recommendation engine.

Developers have free access to event data and reports via API and can download via Web interface as "".csv"".

Platform targets *e.g. iOS, Android, or app stores supported*

iOS, Android, Blackberry and Java

Data ownership and privacy

Flurry keeps data for in-house analytics and benchmarking in "aggregate" form. Anonymity is ensured in Ts&Cs.

Type of analytics *e.g. app store, in-app, billing, web*

in-app

Localytics http://www.localytics.com/

Main proposition *e.g. application analytics SaaS, custom reports, app store analytics - tag line*

Mobile application analytics

Core product *e.g. application analytics SaaS, custom reports, app store analytics*

2-tier Application analytics solution:
1: library/sdk for integration in app during building (Open Source Software)
2: Web portal/service which collects the data and processes in real-time (reportedly 2-3 mins update).

offered in Community (free) and Enterprise (paid) editions.

Localytics Enterprise service builds on the Community version adding the following premium features:
- Bookmark / save charts
- Export session data for integrating mobile analytics data with other enterprise reporting packages.

enteprise customers can access the raw data. free users can only download generated reports
- Data export API for integrating mobile analytics data with other enterprise reporting packages.
free users only via website
- Location services
- Optional first-party analytics

Platform targets *e.g. iOS, Android, or app stores supported*

iPhone, iPad, Android and BlackBerry

Data ownership and privacy

Based on level of agreement:
Enterprise customers: don't use or publish data without approval.
Free users: not-uniquelly identified customers and people without approval.

Type of analytics *e.g. app store, in-app, billing, web*

in-app

Mobixy http://mobixy.com

Main proposition *e.g. application analytics SaaS, custom reports, app store analytics - tag line*

Mobile data analysis and management solutions.
tagline: mobixy provides mobile data analysis and management solutiosn for marketers and product managers. (beyond the if and answer the question of what/when/how/why)

Core product *e.g. application analytics SaaS, custom reports, app store analytics*

Data analytics: in application events and interactions between application and Web services (e.g. yahoo or google apis), location and context, on/off : load times, response times for screen loading etc. Currently integration only via API. Wrapper libraries will be available probably end of month.
Data management: Mobixy monitors device and network connectivity (speed, device type etc) so they can profile the session and optimize the stream/be selective in the order of information transactions with the Web.The aim is to enable the developer to prioritize the data that are being retrieved from the ""cloud"" or any REST Web service. Automatic optimization can then establish which data ""blocks"" can be left out/postponed dynamically based on device and connection performance.

e.g. So the developer can optimize the data or detect why some parts are not being used (e.g. maybe they are very slow to load)

Platform targets *e.g. iOS, Android, or app stores supported*

in-app analytics: currently available via API and is being encapsulated in a library. Library available for iPhone. Roadmap: Java Android, Blackberry and later Windows.

Data ownership and privacy

mobixy doesn't reuse any of the logged information.
Developer/publisher can opt-in marketing program and share aggregate information with marketing partners of mobixy.

Type of analytics *e.g. app store, in-app, billing, web*

in-app; Web and data services (API/REST) application interactions

Recommendations from state of the art

Currently, Localytics (http://localytics.com) is the only commercial end-to-end application analytics platform provider (both client libraries & server components for in-house hosting). Two other companies which offered similar solutions are: Appclix (closed beta since late-2010 no trace of activity since) and Motally (purchased by Nokia 4Q2010 and discontinued products).

Localytics is also the only solution which offers a public version of the source code under a BSD-derived open source license (http://wiki.localytics.com/doku.php?id=the_localytics_modified_bsd_license).

Due to the lack of openly available code bases, technical specifications or standardization efforts to date, webinos could potentially offer a reference open implementation for such systems. To head start the specification and development of the metering/analytics functionality webinos can leverage the open source version of the Localytics code base.

4. High level overlay architecture

Architecture

This section will describe the webinos architecture, which is centered on the notion of a Personal Zone as a means to organize your personal devices and services. Each device, whether it be a mobile, tablet, desktop, TV or in-car head unit, includes a Web browser that is extended to enable the device to be a part of the Personal Zone. The Personal Zone Hub runs on a Web server with a public URL, and provides the means for other people to access your devices and services subject to your preferences. All devices in the Zone have access to a shared model of the context, this allows them to operate when offline, or when temporarily unable to access the Internet.

The webinos architecture seeks to make it easier for Web application developers to create applications that span devices and firewalls. This is achieved through:

- Logical communication paths based on trust relationships, and decoupled from underlying interconnect technologies
- Simple access to local and remote services
- Simple discovery of devices/services
- Trust based on social relationships between people
- Adaptation based upon access to the context

The simplicity of the high level APIs for Web application developers is realized through 3rd party components that layer on top of lower level APIs and mask the complexity involved. It is anticipated that this will lead to a market for such components as demand is stimulated by the continuing evolution of devices and interconnect technologies. This in turn will feed the market for services provided by Web developers. This report mainly focuses on the high level APIs exposed to Web developers, and further reports are expected to elaborate on the lower level APIs and protocols as a basis for interoperability across implementations of the webinos platform.

Applications and Services

Applications may be downloaded and installed on devices, or they may be hosted by servers, with components that are dynamically downloaded when needed. Applications can make use of services, and in turn can provide services. Services may include a user interface exposed as part of an application, e.g. within an HTML iframe element. The ability to combine and tailor services is used to support "mashups". Applications are essentially services that can be installed or bookmarked.

Personal Zones

We individually own an increasing number of devices, for instance, a smart phone, tablet and desktop computer, TV, and other consumer devices. The Personal Zone provides a basis for managing your devices, together with the services you run on them. This includes personal services you use in the Cloud. The Personal Zone supports:

- Single sign-on, where you authenticate yourself to a device, and the device authenticates to the zone. This avoids the need for establishing direct peering relationships between each pair of devices. It also allows for stronger authentication with the services you use. No more typing user ids and passwords into Web page forms! Note that the architecture allows for situations where you are offline, e.g. when you are away from home and are currently unable to access the Internet.
- Shared model of the context. This covers users, device capabilities and properties, and the environment. It enables applications to dynamically adapt to changes, and to increase usability by exploiting the context.
- Synchronization across the devices in the zone. This includes support for distributed authentication, as well as personal preferences, and replication of service-specific data, e.g. social contacts, and appointments. Synchronization is essential for supporting offline usage.
- Discovery and access to services. This includes local discovery, e.g. of services exposed by your devices, whether connected through WiFi, Bluetooth, or USB, as well as remote discovery for services exposed in the Cloud. The high level discovery API allows Web developers to search for all local services, or to filter by service type and context, or even to locate a named service instance. Remote discovery is based upon the URL for a Personal Zone, or an email address or phone number, or even someone's name or pseudonym.
- Licenses for the services you have purchased and run as part of your Personal Zone. This includes locally installed applications and hosted applications, dynamically loaded from Web servers. The aim is to provide an open market for Web developers that is not controlled by a single vendor.
- Trust relationships based upon social graphs. You have full access to all of the devices in your Personal Zone, as well as to shared devices, e.g. a network enabled TV that is accessed through the home's WiFi network, and shared by all family members. You can determine which of your devices are visible to your friends, and what services they can make use of. This is based upon preferences associated with your social graph. The preferences are updated as you make decisions in the course of using services, or through a Zone preference editor.

Binding, privacy and security

The webinos platform provides each device with an API for accessing services exposed directly by the Personal Zone. An example is the method used to discover services matching the given service type and

context constraints. The method is asynchronous, and results in call backs as service instances are discovered. Developers can then provide a user interface for selecting between alternatives, where the list is dynamically updated as services become available or cease to be available. The approach allows Web developers to offer users the means to obtain further information about each of the choices, as well as to record preferences for use in future situations.

The process of binding to a service (having first discovered it) involves:

- mutual authentication, where the Zone authenticates the service, and the service authenticates the Zone
- secure communication through the use of transport or application layer encryption, and checks against man in the middle attacks, spoofed IP addresses and spoofed DNS records
- agreement on data handling obligations as set out in the service's privacy policy
- reviewing and granting the request by the service for elevated privileges

The architecture allows for an extensible set of authentication technologies, including those needed for existing (non-webinos) services, such as Facebook. Users are able to set up multiple pseudonymous identities and to choose which of them should apply in the current situation. Webinos-based services provide authentication requirements and account management information expressed in JSON. To cater for privacy, webinos provides support for machine interpretable privacy policies based upon a subset of P3P also expressed in JSON, together with a link to full human readable policies. Users can further make use of third party assessments of services, e.g. black lists of harmful services, and crowd-sourced assessments. The webinos platform provides a secure basis for executing applications in which error prone features are disabled by default, where such features are a common source of attacks.

Applications (or embedded services) can request elevated privileges. This is typically handled when the application first runs, and the user's decision recorded for subsequent uses. A Zone API enables applications to request a list of privileges, and should be accompanied by information on what the application needs these for. The underlying model is that of notice and consent. The associated user interface is provided by the webinos platform, and not by the applications. A further user interface is provided to enable users to review and revoke decisions. The device itself may impose security policies, e.g. white listing which services may have particular privileges.

Extensibility

The webinos platform APIs are designed for extensibility. It is common to pass an object as an argument to a method where the object supports one or more interfaces. These interfaces are interpreted by third party components, and such third parties are also responsible for documenting the extensions. Web developers can call a standard QueryInterface method to cast an object to a named interface, when necessary to avoid name clashes.

Events or call-backs

Having been discovered and bound, a service is exposed as an object in the Web page's script execution environment. This object acts as local proxy for the service, which may be provided by a remote device. A design decision is whether to support DOM eventing along with the capture/bubble module. The alternative is to allow Web page developers to register a simple call back function, or to pass an object supporting a given interface, i.e. with a named method that is used as a call back. The DOM eventing model fits well when markup elements are used as proxies for services, with the content of the element acting as constraints on the service type and context.

Webinos in the browser

A "webinos" object is exposed as part of the global namespace for Web page scripts, and provides the core set of webinos APIs as methods and properties. The implementation may further involve scripts and other resources running as part of browser extensions (Chrome extension or Firefox addon). These may in turn make use of browser (NPAPI) plugins or local servers where native code is needed for discovery or for service adapters, etc. An example is the discovery of devices connected via USB, where a native code driver is dynamically loaded based upon the vendor and product ids. Service adapters may involve a combination of a low level native code driver together with a script library to interface the service to Web page scripts.

Synchronization and secure storage

Every webinos device will need some secure storage to support authentication, personal preferences, policies and other data requiring synchronization. Synchronization involves detecting and merging differences, and asking the user to resolve conflicts, taking into account periods of offline usage. The process involves a comparison of clocks as a basis for correcting for skews prior to comparing the time of each change. The approach is inspired by work on distributed revision control and 3 way merge algorithms for tree structured data. Synchronization takes place when a device connects to the Personal Zone, and when changes occur. This is also coupled with local discovery, to enable a shared model of the context. For IP-based networks, multicast announcements and query responses can be observed to update a local cache. Information which needs to be kept private can be protected and accessed through HTTP together with transport layer security (TLS) and authentication. Different parts of the context have different security requirements, and it may be appropriate to encrypt them with different keys.

The Personal Zone is exposed as a local API in each webinos enabled device. This needs to function even when the device is operating in isolation, or with a subset of devices in the absence of access to the Internet. This relies on being able to synchronize the devices in a peer to peer model. Synchronization depends on being able to merge changes, and to detect and resolve conflicting changes. If the context data model is independent, then one approach is to simply take the latest change to a particular part of the context. If the context data model has inter-dependencies, the updated model needs to satisfy the integrity constraints. A transactional treatment of changes can help with this, as well as with providing

support for rolling back changes. Synchronization and secure access to the context form a crucial part of the webinos platform. Browsers already support mechanisms for recording preferences and application specific storage, e.g. cookies. Webinos could build upon this with additional database files held as part of the browser profile, and accessible from trusted code in browser extensions.

Personal Zone Hub

To enable external access to your zone, webinos defines a Personal Zone Hub (PZH) as a service that is accessible via the public Internet. This could for instance, be provided as a value-added service to users by Internet Service Providers or it could be integrated in the DSL router at home. The Personal Zone Hub is identified by a URL and supports a RESTful API based upon JSON RPC. The hub is part of your Personal Zone and supports access by you from other devices, e.g. when you walk into an Internet Cafe, enabling you to access your Zone's devices and services for the duration of a browsing session. It also enables access by others, subject to the policies that you have defined.

Personal Zone Hubs collectively form a federated social Web with support for social messaging based upon your relationship to other people. For instance, you could keep a diary and allow your friends to add comments. Your Zone Hub can subscribe to near instant notifications when a topic (feed URL) you are interested in is updated. You can install third party social applications to suit your interests.

The Personal Zone Hub further provides support for discovering other hubs based upon someone's full name or pseudonym. This is implemented as a federated discovery process across hubs, starting from your own hub. The results are ranked according to a measure of social relevance, drawing upon information provided in your profile, or gleaned from other sources. The process is trusted with access to personal data for ranking purposes, but is designed to avoid disclosing such data, except as permitted by the owner's policies. Distributed hash tables provide a solution for locating candidate matches, but further work is needed to determine the best approach for implementing a scale-able solution for privacy friendly ranking of results.

Personal Zone Hubs can also be discovered starting from someone's email address or phone number. The email addresses domain name can be used to locate a query service (typically provided by the domain owner). Note that users may choose to limit discovery, e.g. to people within a given group, or to prevent discovery altogether, in which case it is up to the user to communicate the URL for their Personal Zone Hub to others as needed.

NAT traversal and efficient use of communication networks

The Personal Zone Hub supports the establishment of UDP or TCP connections across well behaved Network Address Translation boundaries. This will not normally affect Web developers, as the webinos platform hides the establishment of such connections. The Personal Zone Hub can also help with the efficient use of communication networks, e.g. by tunneling events through a shared connection rather than setting up new peer to peer connections, which is expensive on current mobile networks. Common NAT devices have TCP session timeouts of 30 minutes to several hours versus just a few minutes for

UDP. Longer lived connections can be realized at a virtual level, with SMS wake up messages to re-establish lapsed connections, providing a means for maximizing battery life on mobile devices.

Key architectural components

This section defines the roles and responsibilities of the key architectural components in webinos and at a high level defines the logical flow and process during normal webinos interactions.

Webinos builds upon the state of the art for Web applications. Taking HTML5 and W3C DAP technologies as a foundation, it extends these concepts to allow for the following:

- Applications which make optimal use of the resources on the featured devices of TV, Automotive, Tablet, PC and Mobile
- Applications which interoperate over diverse device types
- Applications which can make user of services on other devices owned by the same person
- Applications which can make user of services on devices owned by other devices
- Discovery mechanisms to find services, devices and people, on multiple network types - even when they are not connected to the internet
- Efficient communication mechanisms, that can pass messages over different physical bearers, can navigate firewalls, and make sensible use of scarce network resrouces
- Strongly authenticated, communication mechanisms that work bi directionally - we know we really are talking to the remote service, device we thought we were - tackling head on the spoofing and phishing weaknesses of the Web
- And finally, implementing distributed, user centric policy:
    - allowing the user to define what applications work on what devices,
    - to define what information is exposed to other services
    - and ensuring these capabilities are interopable and transferable - ensuring a user stays in control of *their* devices and *their* applications

Webinos Web Runtime (WRT)

A webinos Web runtime, is a special type of browser. It should be capable rendering the latest JavaScript, HTML4/5 and CSS specifications. It is responsible for rendering the UI elements of the webinos application

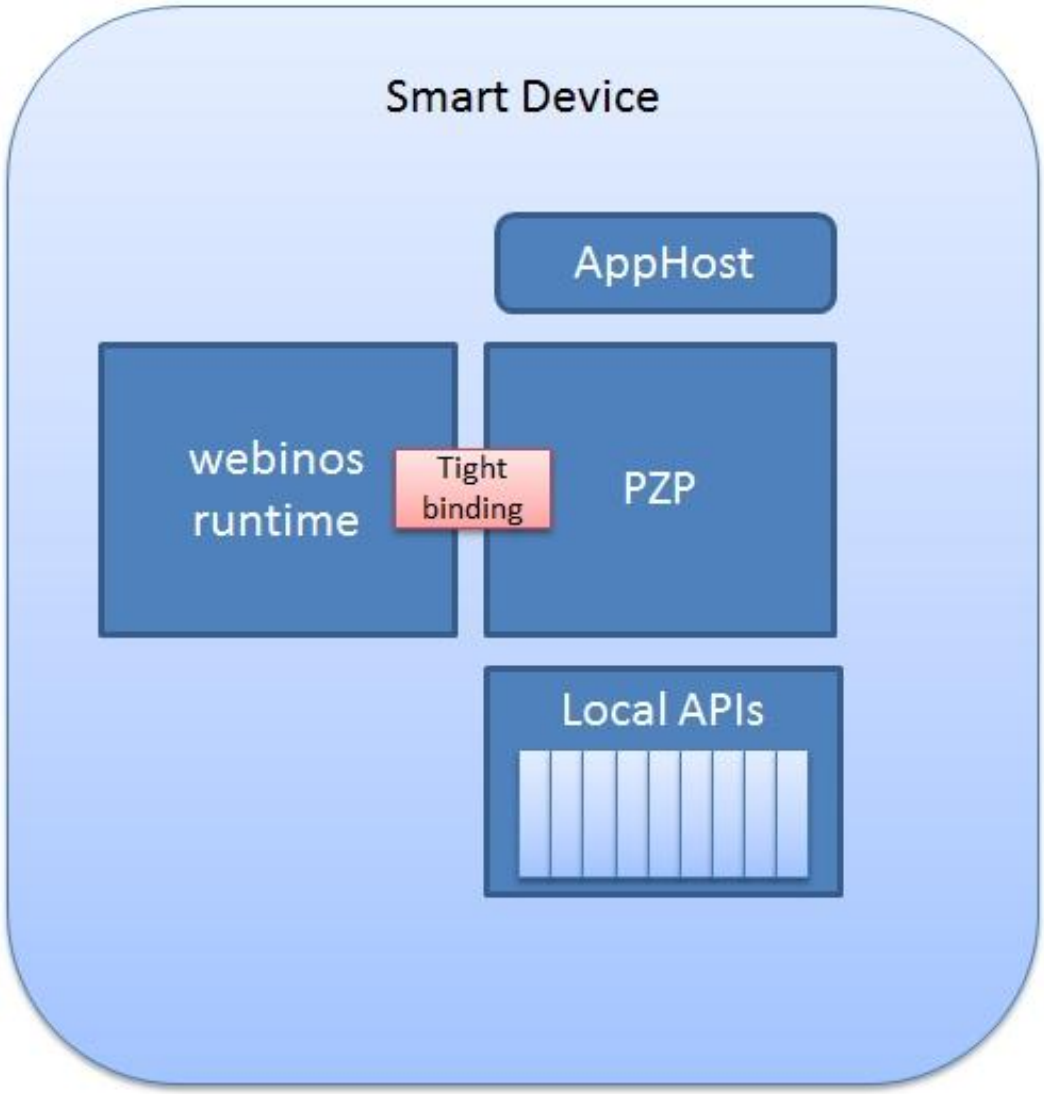
A webinos WRT must be able to access the webinos root object from JavaScript. Via this root object the third party developer will be able to access the webinos functionality.

A webinos WRT differs from a normal browser or Web runtime in that all extended JavaScript functions as well as some normal browser behaviours (such as XHR) must be mediated by the webinos policy enforcement layer.

A webinos WRT will present environmental properties and critical events to the Personal Zone Proxy (PZP) so that it may process the security policy and contextual events, correctly.

A webinos WRT should be deemed *tightly bound* to the Personal Zone Proxy (PZP).

There is special case of WRT that binds to the PZH not the PZP, hence server vs device centric. This variant is called a Server Based Runtime (SRT), rather than a WRT.



Specification areas

The web runtime component must implement the following aspects of the specification

- Foundations:
    - Rendering and code: the WRT must be compliant with all the HTML5, CSS, JavaScript versions defined in the foundations document
    - Packaging: the WRT may be responsible for unpacking the application manifests (W3C widget specification). However, in the implementation phase we may also evaluate the advantages of performing the processing of this at the Personal Zone Proxy (PZP) instead
- Security:
    - PIP: the WRT must act as a policy information point for the webinos policy. In other words the WRT must provide "security context" and call backs into the PEP (Policy enforcement Point) which resides within the Personal Zone Proxy (PZP).
- APIs: the WRT MUST provide the webinos object at document level upon which all the webinos objects and methods may hang. In the implementation phase we shall evaluate the pros and cons of implementing the APIs within the WRT natively vs

webinos Personal Zone Hub (PZH)

The Personal Zone has already been introduced in the Overlay Networking Section.

The Personal Zone is a conceptual construct, that is implemented on a distributed basis from a single Personal Zone Hub (PZH) and multiple Personal Zone Proxy (PZP)s

The critical functions that a Personal Zone hub provides are:

- An fixed entity to which all *requests* and *messages* can be sent to and routed on - a personal postbox as it were
- A fixed entity on the web through which *requests* and *messages* can be issued, for security and optimisation reasons.
- An authoritative master copy of a number or critical data elements that are to synced between Personal Zone Proxy (PZP)s and Personal Zone Hub (PZH), specifically
    - Certificates for Personal Zone Hub (PZH), Personal Zone Hub (PZH) mutual authentication
    - Hashes for user authentication
    - Certificates to authenticate PZXs of *trusted people* against each other
    - Application identifiers (and/or certificates) of applications granted access into the zone

  - Service identifies (and/or certificates) for trusted services to which the personal zone may attach
  - (Subject to investigation) device identifiers, to assist with platform integrity tests
  - (Subject to investigation) credentials for "non webinos" services to give a pseudo single sign on experience
  - All policy rules, for distributed policy enforcement
  - All relevant context data
- The functions therefore that a Personal Zone Hub (PZH) can support are
  - User authentication service
  - Personal Zone Proxy (PZP) secure session creation for transport of messages and synchronisation
  - Service session creation for secure transport of messages between applications and services
  - Secure social networking: using the exchanged certificates between *trusted people*
  - Potentially: single sign on service to other web services, using the Personal Zone Hub (PZH) as a secure proxy
- A webinos service host: a Personal Zone Hub (PZH) can host directly Services/APIs that other applications can make use of.
- Context sync: the Personal Zone Hub (PZH) should act as the master repository for all context data
- A webinos executable host: a Personal Zone Hub (PZH) will be able to run a server resident webinos applications (these will be JavaScript program files wrapped in a webinos application package)

Specification areas

The Personal Zone Hub (PZH) component must implement the following aspects of the specification

- Foundations:
  - Rendering and code: the Personal Zone Hub (PZH) will run server resident webinos applications using node.js or similar
  - Packaging: the Personal Zone Hub (PZH) should be capable of unpacking and performing security checks on packed widgets
- Security:
  - The Personal Zone Hub (PZH) must store the policy files

  - The Personal Zone Hub (PZH) should act as a server based Policy enforcement point- therefore must mediate all relevant traffic

- Messaging: the Personal Zone Hub (PZH) must be able to route messages to the relevant Personal Zone Hub (PZH) or Personal Zone Proxy (PZP), or in cases where the message is routed to a locally hosted service, pass it for execution
- Synchronisation: the Personal Zone Hub (PZH) must implement the synchronisation algorithm and process synchronisation protocol messages.
- Authentication:
  - the Personal Zone Hub (PZH) must allow for a user to authenticate, or raise their authentication level.
  - the Personal Zone Hub (PZH) must authenticate Personal Zone Proxy (PZP)s and *other users* to set up trusted sessions

webinos Personal Zone Proxy (PZP)

The webinos Personal zone satellite proxy, acts in place of the Personal Zone hub, when there is no internet access to the central server.

In order to act in its place, certain information needs to be synchronised between the satellites and the central hub.

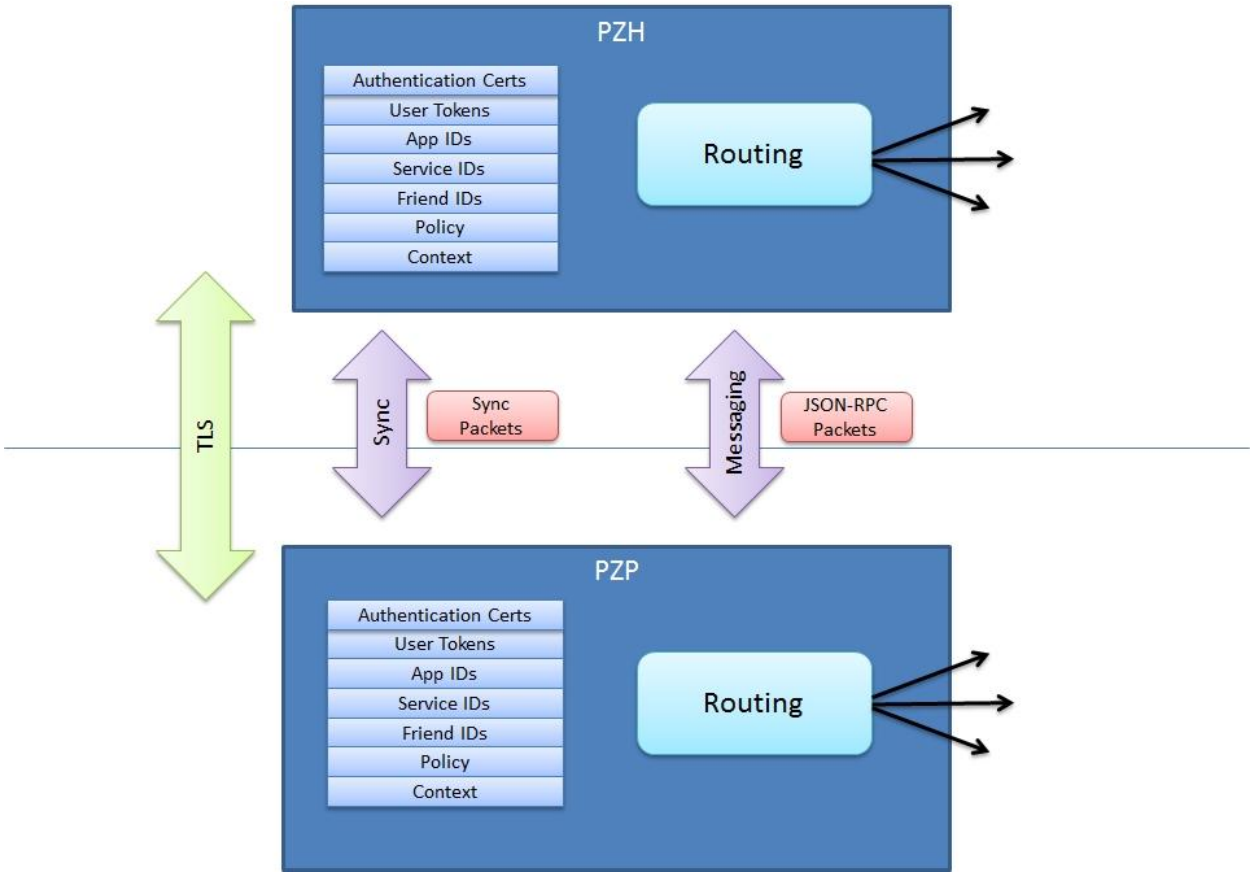
This information has already been listed above.

The Personal Zone Proxy (PZP) fulfils most, if not all of the above functions described above, when there is not Personal Zone Hub (PZH) access

In addition to the Personal Zone Hub (PZH) proxy function, the Personal Zone Proxy (PZP) is responsible for all discovery using local hardware based bearers (Bluetooth, ZigBee , NFC etc)

Unlike the PZH, the PZH does not issue certificates and identities.

For optimisation reasons PZPs are capable of talking directly PZP-PZP, without routing messages through the PZH



Specification areas

The Personal Zone Proxy (PZP) implements all of the above functions, with the following differences

- Messaging:
    - A Personal Zone Proxy (PZP) routes all "internet" messages to the parent Personal Zone Hub (PZH) for distribution
    - A Personal Zone Proxy (PZP) routes all "local" messages to the relevant local device, using
- Discovery: the Personal Zone Proxy (PZP) must implement a full array of local device discovery protocols.
- Security:
    - the Personal Zone Proxy (PZP) is THE primary policy enforcement point for all application processing
    - the Personal Zone Proxy (PZP) will attest to the integrity of other key components on the device
- Packaging: a Personal Zone Proxy (PZP) may optionally - subject to investigation - perform webinos application package processing and integrity checking on behalf of the WRT

webinos Application

A webinos application runs "on device" (where that device could also be Internet addressable i.e. a server).

A webinos application is packaged, as per packaging specifications, and executes within the WRT.

A webinos application has its access to security sensitive capabilities, mediated by the active policy.

A webinos application can expose some or all of its capability as a webinos service

An application developer is granted access to webinos capabilities via the webinos root JavaScript object.

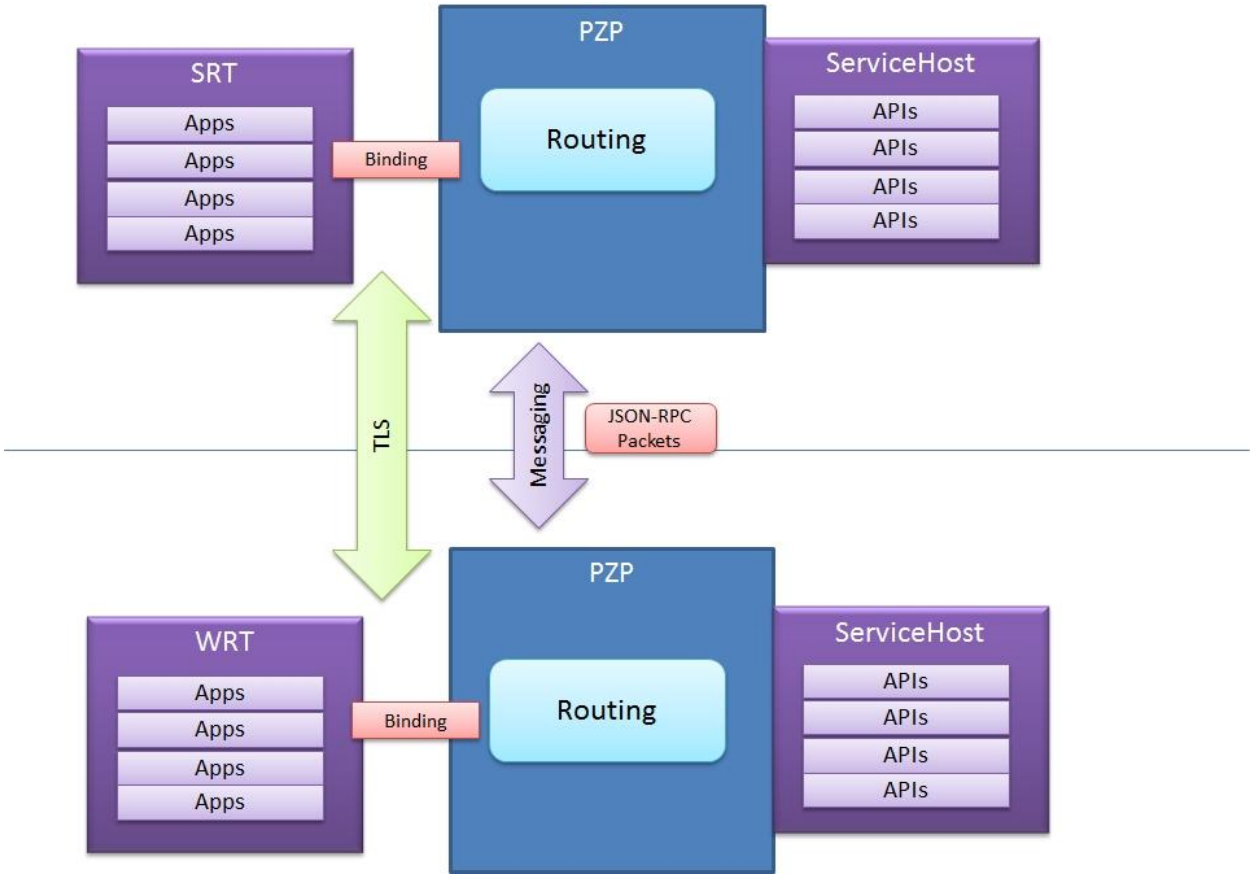
Specification areas

An application developer needs to be aware of the following parts of the specification

- Foundations: an application needs to packaged and programmed according the foundation specification
- APIs: a developer has access to the rich set of capabilities defined within the API specification
    - infrastructure capability: much of the intelligence of webinos is provided transparently to users. However certain key functions, such as discovery, and service binding, are provided

webinos Service

A webinos service is a collection of functions and events, that are accessible by an webinos application



These functions and events are always presented to the application developer as a set of JavaScript functions, no matter where the implementation resides.

There exist the following sub-types of webinos services

1. native device webinos APIs: such as specified in deliverable WP3.2. These may be implemented on the same device on which the application resides, and the implementation may be provided through a JavaScript binding to native code, via plugin technologies such as NPAPI, or indeed hard-coded enhancements to a JavaScript engine. Access to the API must still be mediated by the PEP (policy enforcement point) within the Personal Zone Proxy (PZP)

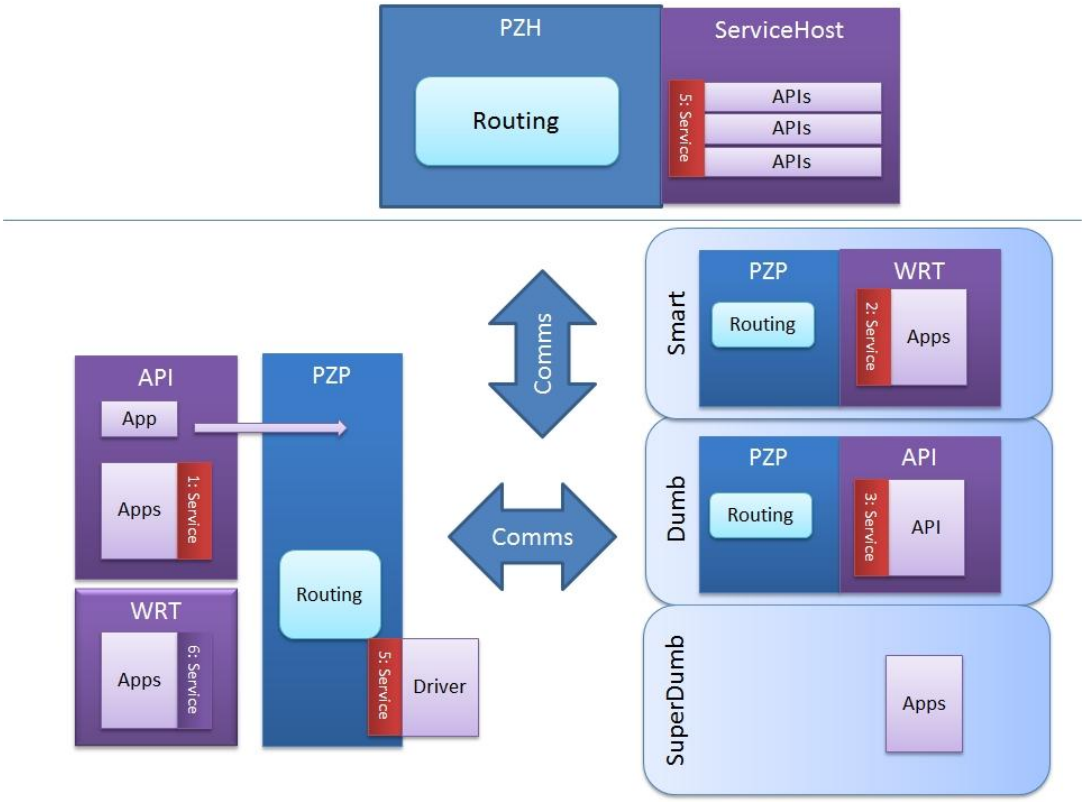
2. remotable smart-device hosted webinos APIs: APIs that can be accessed remotely (using JSON RPC). A remotable webinos API is hosted by a Personal Zone Proxy (PZP) and again access is mediated by the PEP on the Personal Zone Proxy (PZP) of caller AND the Personal Zone Proxy (PZP) of provider

3. remotable dumb-device hosted webinos APIs: As above, where the device is not a smartphone, PC or tablet, but a small sensor-like device which has its own Personal Zone Proxy (PZP) and therefore can directly authenticate to the PZ and communicate via JSON RPC

4. remotable super-dumb-device: as above but device is even more lightweight - and cannot talk webinos directly. Instead a host device presents as webinos driver (a mini Personal Zone Proxy (PZP)) that can communicate natively to the super-dumb-device and transcode the bi-directional comms into webinos protocols.

5. remotable server hosted APIs: these are web services, accessible JavaScript (using JSON RPC). These are hosted by a Personal Zone Hub (PZH) and security mediated by the PEP within the Personal Zone Hub (PZH)

6. application hosted APIs: a full application, which is hosted by the WRT may present external services (JavaScript APIs) that other applications can then make use of



Specification areas

A webinos service must take note of the following parts of the webinos specifications

- Discovery: a service must be discoverable and be able to describe itself to the application in accordance with the discovery specification

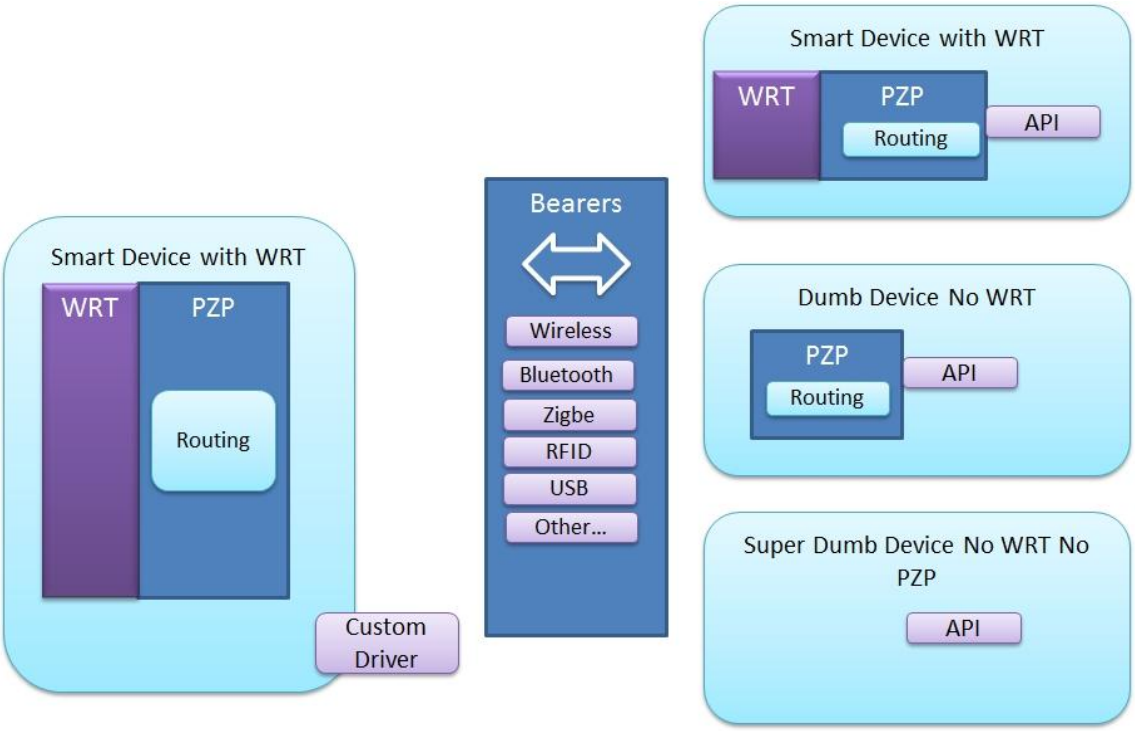
- Messaging : a service must be able to receive and respond to incoming RPC messages

Local Connections

One of the critical innovations of webinos, is the virtual overlay network that allows different applications and services to talk to each other over many different interconnect technologies.

Not only are the interconnect technologies for local messaging, there are three different scenarios in which this communication can take place

These are highlighted in the diagram below.



In turn:

1. Connecting to a full smart device, that hosts both a PZP (therefore can host native APIs presented as services) and a WRT (so can host webinos applications exposing webinos services)
2. Connecting to a dumb device, it hosts a PZP but not a WRT. This means that it can expose only native APIs, not webinos applications
3. Connecting to a super-dumb device, it hosts neither a PZP nor a WRT, but can expose webinos services - if the client PZP hosts a customised driver

Specification areas

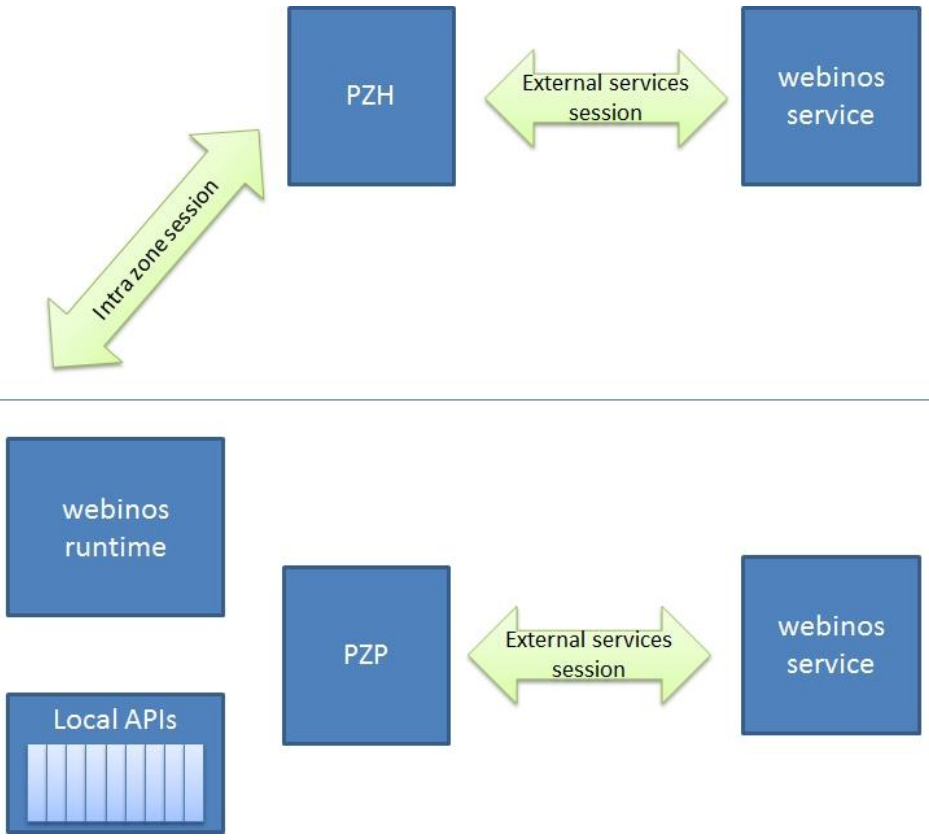
The personification that outline the detail of these connection scenarios are to be found in

- Overlay networking
- Discovery
- Messaging

Sessions

A functioning webinos network will consist of, multiple devices, multiple servers and multiple applications. It will require the interaction of PZPs and PZHs belonging to different users, over multiple different networks.

Within that complex interaction, there will be many notions of session at different levels. It is important therefore to be clear with our terminology.



Intra Personal Zone Relations

A single user will have many PZPs, on different devices, but only a single PZH, hosted on the Web.

PZPs need to be installed securely on devices PZP installation bootstrap, however once this is done a long term relationship now exists between that PZP and the PZH. We will call this an "Intra Personal Zone Pairing". This pairing shall be manifest by the PZP and PZH having exchanged certificates. Section Conceptual Architecture explains the details.

If a pairing exists between a PZP and a PZH they should try to enter an active Intra Personal Zone Session with one another.

PZP-PZH sessions (Intra Personal Zone Sessions) always take place of the publicly addressable internet. This is because one of the defining characteristics of a PZH is that it is must be permanently addressable on the internet. A PZP-PZH session takes place over a TLS connection, assuring that the information

exchanged in the session is secure. This secure channel, once established is the route via which all communications between PZH and PZP, in either direction, takes place.

Note: as an optimisation out of band "wakeup" notifications may be required to issue PZH->PZP messages in a timely fashion. These out of band notifications should only contain the bare minimum information to request the PZP reactiviates the connection. No sensitive information should be passed in the wakeup notifications, as they are not protected by the TLS channel.

When and PZH and PZP session is active it means that messages can be routed in either direction, within a reasonable time-frame. (In practice less than a pre-stipulated time-out value)

A session is established when a PZP and PZH have successfully authenticated.

When a session is successfully established between a PZP and a PZH, a bidirectional channel of information is established across which all the following may happen

1. A PZP may authenticate/re authenticate against the PZH
2. Outgoing webinos messages to external services (multiplexed through the PZP-PZH channel)
3. Synchronisation traffic, of the following
    1. webinos user identity information
    2. webinos user data information
    3. webinos PZP and PZH certificates
    4. webinos policy
    5. webinos friends identity information and certificates of their PZHs
    6. webinos services tokens
    7. webinos device identity tokens
    8. webinos application identity tokens
    9. webinos application data for synchronisation

External services relations

Individual webinos applications create "sessions" with webinos services.

All such sessions must be mediated by the PZP of the originating application. In other words only application bound to a users PZP can make user of webinos services. The originating users PZP will take care of authentication and the user centric policy enforcement.

The application can connect to two types of service:

1. anonymous service: this is a webinos service that is mediated by an un-authenticated personal zone

2. PZ hosted service: that is a webinos service owned by someone. Note in this scenario the permission to access to the services, is mediates by two policy enforcement points, the requester of he service and the hoster of the service.

Much like intra zone sessions, there are distinct notions of "Pairing" and "Binding".

If an application has been "paired" with a service, tokens may be exchanged at the PZX level. This token exchange can be used by the webinos infrastructure to short-cut authentication and permissions. The active policies on any participating app-service flow, must still have this permission granted.

The process of binding an application with a service (irrespective of whether it has been already paired), is akin to the notion of an intra zone session as described above. Whist the session is active, in other words whilst the binding is fixed then:

- Messages can be routed bidirectionally, between app and service within a reasonable time frame (within a pre stipulated time-out value)

External service sessions can happen over the public internet or via the local network

Public internet sessions to Web hosted services will be mediated by the PZH

Local webinos sessions to local device services (including also APIs local to the WRT) are mediated by the PZP.

External service sessions can therefore be carried over a number of physical networks and higher level protocols.

5. Specification

The formal specification is broken into subject areas, each of which has its own section

- **Core architecture and specification:** defines the key architectural components and processes that dictate how the webinos technology works together
    - **Foundations:** by reference to the critical foundation technologies, such as core widget packaging specifications, defines and extends the data formats and protocols required to write and package applications, that interact with webinos.
    - **Authentication:** defines the protocols required to authenticate the critical webinos components against each other
    - **Discovery:** covers the mechanism by which different services can be found over multiple different networks.
    - **Messaging:** webinos defines its own mechanism for efficiently pasting different message types over the webinos overlay network
    - **Context:** defines privileged mechanisms by which the contextual information and events from multiple devices can be aggregated and supplied and later disseminated from the Personal Zone Hub
    - **Security:** outlines the critical security elements, which are explored in greater detail in the Security Architecture definition
- **APIs**: formally defines the JavaScript APIs which a local application may discover and use on device, and which may be invoked remotely from an application on another device
- **Security:** formally specifies the APIs, data formats and protocols required to implement a secure distributed Web application execution environment

Foundations

The Foundations specification is about defining the structure of webinos applications and how they are able to interact with webinos. This also includes packaging of applications, application APIs and embedding webinos extensions in applications.

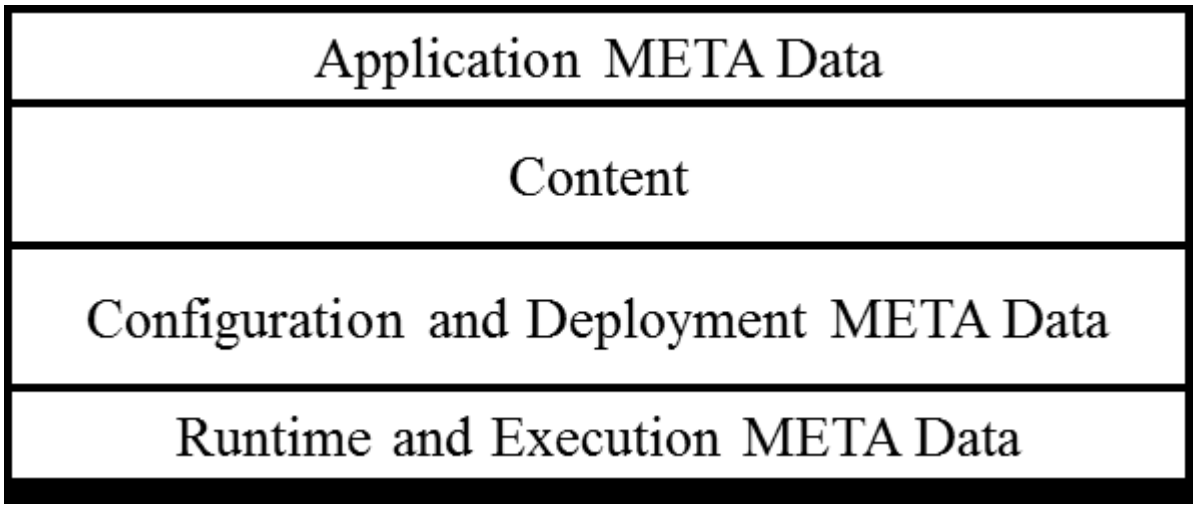
Formal Specification of webinos Application

A webinos application is defined as follows (citing D2.2):

```
An application written using webinos technologies that will run on a device, across a
range of devices reflecting the domains mobile, stationary devices, automotive or home
```

media and/or server. The application will be able to securely and consistently access device specific features, communicate over the cloud and adjust to the device and context specific situation.

Webinos technologies include several existing and upcoming Web technologies, as well as webinos-specific add-ons which enable applications to be developed for multiple platforms. This includes access to device features, transparent communication across devices, and secure application execution. The relevant Web technologies, if already available, are referred in the dedicated sections of this specification.



A webinos application package consists of four types of metadata put together in an application manifest file.

1. Application metadata provides human-readable semantic information about the application itself, e.g., version, description, author etc., which can be presented to the user and is accessible by the application using APIs.

2. The content provides all the application and layout logic and also media content such as images and video needed by the application.

3. The deployment and configuration metadata provides information for the Web runtime about how to deploy and install the application, e.g., whether the application can run in the background or not and which view mode the application prefers to use. This metadata also contains information about distributed application deployment as well as any additional functionality the application will expose.

4. The runtime and execution metadata part of the application package provides information about conditions which must be fulfilled to execute the application, e.g., fulfilled policies or conditions under which the application may be automatically executed (non user driven).

Packaging

The core structure and content of webinos applications is defined through the W3C Widget Packaging and Configuration [Widgets] specification. Webinos application packaging, as a superset of W3C Widget packaging, adds some further features to this specification in order to meet additional requirements. The following requirements show that webinos must conform to various W3C specifications.

WRT-01: The webinos WRT **MUST** be capable of processing widget packages as defined in [Widgets].

WRT-02: The webinos WRT **MUST** support the [WidgetDigSig] specification in order to verify the author and/or distributor of the application.

WRT-03: The webinos WRT **MUST** support application network access control as defined in the Widget Access Request Policy [WARP] for applications that want to communicate with remote resources.

WRT-04: The webinos WRT **MUST** implement the URI scheme as defined in [WidgetURI] to address resources within W3C Widget and webinos application packages.

In addition to the XML elements defined in the Widget configuration document [Widgets] webinos applications can make use of webinos-specific extensions. The following separate webinos-specific XML namespace must be used to reference webinos-specific xml elements.

Webinos Extension XML namespace: http://www.webinos.org/webinosapplication

The following elements are webinos-specific extensions to the metadata part of the [Widgets] specification and **MUST** be supported by webinos WRTs.

The distributor element

A distributor element represents people or an organization that distributed the instance of a webinos application. Commonly application stores are distributors.

Context in which this element is used: As a child of the widget element.
Content model: Any.
Occurrences: Zero or one.
Attributes: href, email.

The email attribute represents an email address associated with the distributor. The email attribute is optional.

The href attribute represents an URI that associates with the distributor. The href attribute is optional.

Example of Usage:

```
1 <widget xmlns="http://www.w3.org/ns/widgets"
2    xmlns:webinos="http://www.webinos.org/webinosapplication">
3                              <webinos:distributor    email="info@examplarystore.com"
href="http://www.exemplarystore.com">
4            Examplary Store
5        </webinos:distributor>
```

```
6        <content src="widget.html"/>
7 </widget>
```

The versionName attribute of the widget element

A version name element represents the version of the application in a human-readable manner. The versionName element is optional and is not used for application life cycle management, e.g., application update.

Context in which this element is used: As a parameter of the widget element.
Content model: Any.
Occurrences: Zero or one.

Example of Usage:

```
1 <widget xmlns="http://www.w3.org/ns/widgets"
2    xmlns:webinos="http://www.webinos.org/webinosapplication"
3    webinos:versionName="Silver">
4        <content src="widget.html"/>
5 </widget>
```

The validfor attribute of the widget element

The *validfor* attributed defines a time period when the application is valid and can be used. The time frame is specified in elapsed milliseconds after the first application execution. If the specified time is elapsed the user should not be able to execute the application any more. This value gives only the semantic information without any security or licensing mechanism behind. Additional security, digital rights management (DRM) or licensing methods are implementation specific and left unspecified.

Context in which this element is used: As a parameter of the widget element.
Content model: Number.
Occurrences: Zero or one.

Example of Usage: Application valid for one week

```
1 <widget xmlns="http://www.w3.org/ns/widgets"
2    xmlns:webinos="http://www.webinos.org/webinosapplication"
3    webinos:validfor="604800000">
4        <content src="widget.html"/>
5 </widget>
```

The validuntil attribute of the widget element

The *validuntil* attributed defines a date and time until the application is valid and can be used. The time frame is specified as in milliseconds whereas the date and time is encoded as milliseconds since midnight of January 1, 1970, according to universal time. If the specified date and time is reached the user should not be able to execute the application any more. This value gives only the semantic information without any security or licensing mechanism. Additional security, digital rights management or licensing methods are implementation specific and left unspecified.

Context in which this element is used: As a parameter of the widget element.
Content model: Number.
Occurrences: Zero or one.

Example of Usage: Application valid until 12.31.2011 12am.

```
1 <widget xmlns="http://www.w3.org/ns/widgets"
2    xmlns:webinos="http://www.webinos.org/webinosapplication"
3    webinos:validuntil="1325332800000">
4        <content src="widget.html"/>
5 </widget>
```

The copy-restricted element

Adding the *copy-restricted* element to the configuration document indicates that copy, export or installation of the application on another device using webinos application sharing features is forbidden. It is possible to allow copies on devices belonging to the same personal zone as the device where the application was installed at first, using the *restricted-to* attribute with the value 'personal-zone'. If the element is absent no restrictions are applied and exporting the application to the file system and installing it on another device is possible if the WRT provides means for exporting applications. The information provided by the *copy-restricted* element gives only the semantic information without any security or licensing mechanism. Additional security, digital rights management or licensing methods are implementation specific and left unspecified.

Context in which this element is used: As a child of the widget element.
Content model: None.
Occurrences: Zero or one.

Context in which the restricted-to attribute is used: As an attribute of the copy-restricted element.
Content model: DOMString.
Occurrences: Zero or one.

Example of Usage: Allowing copies on devices of the same zone

```
1 <widget xmlns="http://www.w3.org/ns/widgets"
2    xmlns:webinos="http://www.webinos.org/webinosapplication">
3        <content src="widget.html"/>
4        <webinos:copy-restricted webinos:restricted-to="personal-zone"/>
5 </widget>
```

Application Interface

In [WidgetAPI] the W3C defines an API to access application specific data defined in the configuration document as well as a persistence model. This API is used and extended to provide support for the webinos-specific XML elements defined in this specification.

```
1 interface Widget {
2
3     //webinos-specific attributes
```

```
 4     readonly attribute DOMString     distributor;
 5     readonly attribute DOMString     distributorEmail;
 6     readonly attribute DOMString     distributorHref;
 7
 8     readonly attribute DOMString     versionName;
 9     readonly attribute unsigned long long validfor;
10     readonly attribute unsigned long long validuntil;
11
12     //Widget standard attributes
13     readonly attribute DOMString     author;
14     readonly attribute DOMString     authorEmail;
15     readonly attribute DOMString     authorHref;
16     readonly attribute DOMString     description;
17     readonly attribute DOMString     id;
18     readonly attribute DOMString     name;
19     readonly attribute DOMString     shortName;
20     readonly attribute Storage       preferences;
21     readonly attribute DOMString     version;
22     readonly attribute unsigned long height;
23     readonly attribute unsigned long width;
24 };
```

As shown in the Widget object interface description several webinos-specific attributes are added. The meaning of each new attribute is described in the following list.

- distributor attribute: The distributor attribute provides read only access to the distributor element's content of the config.xml if available. Otherwise this attribute is NULL.
- distributorEmail attribute: The distributorEmail attribute provides read only access to the distributor's email adress if available. Otherwise this attribute is NULL.
- distributorHref attribute: The distributorHref attribute provides read only access to the distributor's URI reference if available. Otherwise this attribtue is NULL.
- versionName attribute: A human readable version name. If not set in the configuration document it contains the same value as the version attribute.
- validfor attribute: A numeric value represented in milliseconds that indicates how long the application can be used before usage should be prohibited.
- validuntil attribute: A numeric value represented in milliseconds since midnight of January 1, 1970, according to universal time that indicates until when the application can be used before usage should be prohibited.

The following specification for distributed webinos applications will add more extensions to the Widget application interface as well as to the Widget packaging and configuration specification which are described more comprehensively and have their own sections in the document.

Distributed Webinos Applications

Webinos allows the creation of multi-device or distributed applications not only from the execution environment point of view but also from the deployment and packaging point of view. Webinos allows developers to design their applications in a way that applications (or only parts of them) can be automatically or programmatically deployed on to other devices, e.g., because they are more suitable for the execution of the application code because of the application design, the feature access or because of performance reasons.

The webinos distributed application functionality allows the packaging of any number of sub-applications, referred to as child applications, within a main webinos application package, referred to as the parent application. Child applications are also full webinos applications and follow the webinos application packaging specification.

Application code encapsulated in child applications can be code where the developer decides that it makes sense to create specific application modules for performing certain tasks, the functionalities provided by the modules can be used by the parent application and also can be shared between other applications. Thus, application distribution is not only about outsourcing code to other devices but also about using functionalities provided by one application across others and de-coupling application logic.

Use Case Examples

1 Smart Text Input:

Using a smartphone as text input device for applications running on a TV set. Here, the smartphone not only sends key-codes to the “main” application, it also shows some appropriate graphics in order to support the text input. In addition, the outsourced code running on the smartphone may check the text input in order to prevent sending of unnecessary or incorrect data to the “main” application.

2 Smart sensors:

Assume that an application wants to be informed when remotely available sensor data (real sensor or any another webinos enable/compatible device) crosses a specific measurement threshold. The application could check the sensor reading periodically and take some action based on this. Since this would produce unnecessary traffic and needs the primary application to run continuously, it would be better to only get a sensor event if the threshold is reached. To achieve this, the application may outsource a piece of code to the desired sensor or device. The code locally checks the sensor/requested data until the threshold is reached. The outsourced code informs the “main” application via an event system about this so that the application can perform a specific action.

3 Component sharing:

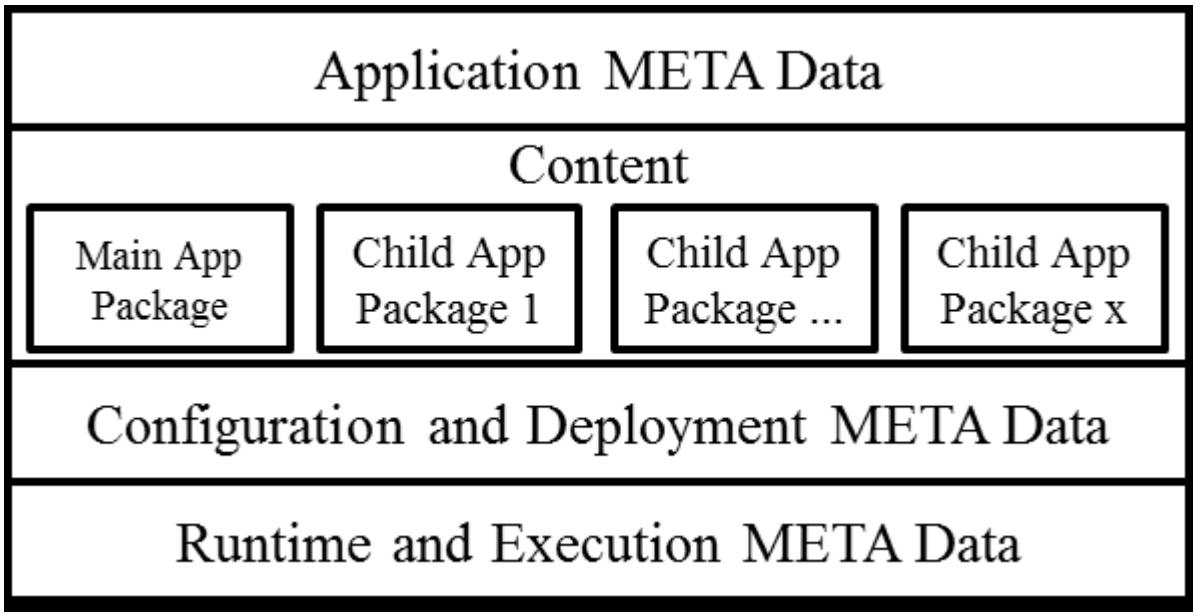
The Android Intent programming paradigm (see http://developer.android.com/reference/android/content/Intent.html) allows easy sharing of application components between other applications. For example a tiny application is only designed for picking a location from a map and

providing the attached geographic location for the selected map position. Since this task is common across multiple applications it would be an ideal candidate for cross application sharing of services. Thus, the application is able to be invoked from other applications and could send back the result to the calling application. Other possibilities like these borrowed from Android Intents could be provided to webinos applications by using the application distribution mechanism not only for distributing applications but also for sharing application features.

Packaging of Distributable Applications

Applications that want to make use of webinos distributed application features have to declare the child applications which should be made available. Three child application deployment options can be considered:

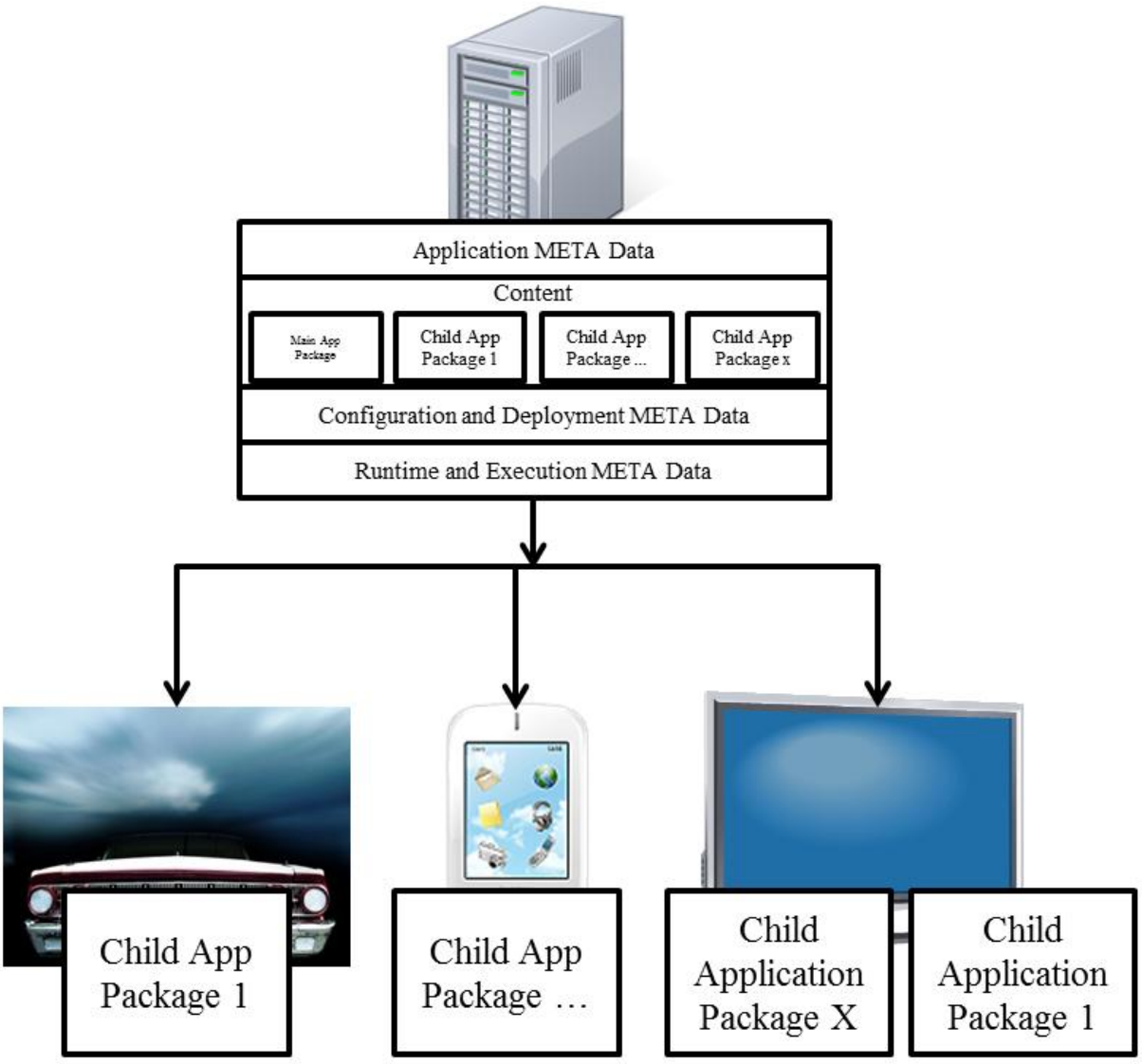
- An application can outsource functionality to the child application for any application specific reason, on the same device or other device, to be accessible using the webinos discovery service (if shared functions are declared).
- An application can share functions with other applications to allow reuse of existing code between multiple applications without allowing access to the whole application but to a dedicated component.
- An application can package multiple child applications in one application package in order to ship and install multiple applications at once.



To declare code as being a child application, two steps are needed:

1. Package the components to make them distributable.
2. Declare those components as being distributable.

Packaging a child application as a distributable application works exactly the same way as packaging a full webinos application. Thus, a child application could be as content-rich as a full webinos application but with a dedicated small single purpose. Applications which do not declare any child application do not need any distributed application related processing by the WRT. They will be installed as described in [Widgets].



Note: In the current release webinos, supports only pre-defined child applications which are available in the parent's application package during application installation. Future versions may support dynamic creation of application packages during application runtime by providing appropriate APIs. Both the parent and child applications must be signed by the same authority.

Packaged child applications must be placed in the webinos-specific 'components' folder which must be located in the root folder of the parent's application package.

Exemplary application structure:

- root
  - config.xml
  - app.html
  - icon.png
  - Scripts
    - app.js
  - Styles
    - small.css
    - big.css
  - Components
    - child1.wgt
    - child2.wgt
    - child3.wgt

Application Installation on multiple devices

Automatic Deployment

During installation of a parent application its child applications, if any, can be automatically installed on the same device as the parent application. The WRT may provide the possibility to also install child applications on other devices if not prohibited by the child application definition in the configuration file (config.xml).

The child application that should be installed on other devices must provide a child element in the application's configuration document. This will trigger the webinos runtime to install the related child application. If the element is absent, the related child application will not be automatically deployed.

Example of Usage:

```
1 <widget xmlns="http://www.w3.org/ns/widgets"
2    xmlns:webinos="http://www.webinos.org/webinosapplication"
3    id="http://exampleapp.org/app1" webinos:type="container">
4        <webinos:child webinos:install="any">child1.wgt</webinos:child>
5        <webinos:child webinos:install="local">child2.wgt</webinos:child>
6        <webinos:child>child3.wgt</webinos:child>
7 </widget>
```

- Line 3: The application package is defined as type 'container'. This means that no parent application exists in the application package but, if defined in the manifest, child applications can be installed when the WRT processes the application package, it is a convenience mode for installing multiple applications using only one package. After the WRT finished processing the package it can be removed from the device because there is no application that can be executed. If no parent and no child applications are defined nothing has to be installed and the application package can be removed. The type attribute is optional. If the type is set to 'container' the content element is not evaluated and can be absent. The other way round, if the container attribute is absent, the content element must be declared in the manifest.

- Line 4-5: The install attribute specifies whether a child application should be installed directly after installing the main application package (set to 'any' or 'local') or not (any other value or absent). If not set to one of the allowed values, child applications can be installed later using the webinos Widget API. Before starting installation of a webinos application the WRT has to check if any declared child applications are available. If not the installation process must be cancelled and the user must be informed about an invalid application package. If *install* is set to 'any' the WRT must show a native WRT dialog to the user containing a set of available devices where the child application can be installed on to the user. Available devices could be every device of the user's personal zone or other devices accessible to the user. The WRT must show information about the child application available in the configuration document (e.g., author, title, version, description,...) to the user based on the application package source and destination device, to allow the user to double check what will be installed. If 'any' is specified, the user may select one, more or all available devices for installing the child application. Afterwards the WRT has to install the application on the selected device as specified in section Inter-Runtime Application Deployment. If set to 'local' the related child application must be installed on the same device as the main application was previously installed. The user does not have to select a target device, but should be informed that this will install only locally.

- Line 6: Each child application contained in the application package must be advertised using the child element. Child applications contained in the package but not advertised in the configuration document are rejected by the WRT and, thus, cannot be automatically installed during application installation phase or using the webinos Widget API. Before starting installation of a webinos application the WRT has to check if any child application declared in the configuration document is also physically available in the package. If not the installation process must be cancelled and the user must be informed about an invalid application package.

Note: In future versions more advanced automatic deployment mechanisms may be introduced which allowing stating a number of filters within the remote-install element of the configuration document in order to define appropriate devices for application deployment.

On Request Deployment

Webinos allows remote installation during installation of the parent application and it is also possible to deploy code on demand at the application runtime. Webinos provides an application deployment API using the Widget interface which takes the application ID of the child that should be installed. The application ID must be specified in the configuration document of the child as specified in [Widgets]. Following the WebIDL specification for deploying child applications:

```
 1
 2 [Callback=FunctionOnly, NoInterfaceObject] interface DeploymentSuccessCallback {
 3     //called if a child application was successfully installed
 4     //childID is the application id which was used during deployChild
 5     //serviceID is the unique application id that can be used to explicitly address
the deployed service within webinos service discovery
 6     void onSuccess(in DOMString childID, in DOMString serviceID);
 7  };
 8 [Callback=FunctionOnly, NoInterfaceObject] interface DeploymentErrorCallback {
 9       void onError (in DeploymentError error);
```

```
10  };
11
12 interface DeploymentError {
13       const unsigned short INSTALLATION_CANCELED_BY_USER = 101;
14       const unsigned short PERMISSION_DENIED_ERROR  = 102;
15       const unsigned short NOT_REACHABLE  = 103;
16       const unsigned short UNKNOWN_APPLICATION_ID  = 104;
17       const unsigned short ALREADY_INSTALLED = 105;
18       const unsigned short INSTALLATION_ERROR_OTHER = 106;
19       readonly attribute unsigned short code;
20       readonly attribute DOMString applicationID;
21 };
22
23 interface Widget {
24          //Deploys a child application known to the WRT through the definition in
the application s manifest
25         //file on another device. If local = false or not specified the WRT has to
provide a list of available
26           //devices to the user where the application should be installed on, if
local = true the WRT has to
27         //install the selected child on the same device as the API is bound to.
28                    void  deployChild(in  DeploymentSuccessCallback  onSuccess,  in
DeploymentErrorCallback onError, in DOMString childApplicationID, in optional boolean
local);
29 }
```

To install a child on a selected remote device the webinos device discovery API can be used to find available Widget API services on other devices where deployChild can be used remotely.

Exposing Application Functionalities as Service to other Applications

As introduced in the beginning of this section, webinos applications may share functionalities across other applications. To make functions available to others the shared element containing a number of shared-function and shared-api elements must be added to the application's configuration document. The content of the shared-function element must be a JavaScript function defined in the JavaScript part of the application which will be accessible to other applications using webinos discovery services and searching for services with the type defined in the id attribute of the widget element. To group functions and allow exposing of multiple APIs at the same time the shared-api element can be used. The shared-api element must have an api-name attribute which uniquely identifies the exposed API. The service can then be instantiated by using the webinos discovery service while using the api-name as input parameter for the service type of the service that should be discovered.

Example of Usage:

```
1 <widget xmlns="http://www.w3.org/ns/widgets"
2    xmlns:webinos="http://www.webinos.org/webinosapplication"
3    id="http://exampleapp.org/app1">
4        <webinos:shared>
5         <webinos:shared-api api-name="http://www.w3.org/ns/api-perms/geolocation">
6         <webinos:shared-function>watchPosition</webinos:shared-function>
7         <webinos:shared-function>getCurrentPosition</webinos:shared-function>
8         <webinos:shared-function>clearWatch</webinos:shared-function>
```

```
 9              </webinos:shared-api>
10           <webinos:shared-function>exampleFuntion</webinos:shared-function>
11         </webinos:shared>
12         <content src="widget.html"/>
13 </widget>
```

The visibility of the shared functions can be restricted to be only accessible by a parent application, a child application or an application running in the same personal zone. To define this, the visibility attribute of the shared-function element can be used. For example:

```
<webinos:shared-function        visibility=”parent”>function1</webinos:shared-function>
allows only a parent application to access function1.
<webinos:shared-function visibility=”child”>function2</webinos:shared-function> allows
only a child application to access function2.
<webinos:shared-function          visibility=”personal-zone”>function3</webinos:shared-
function> allows only applications running in the same personal zone as the service
application to access function3.
```

To define whether the service is permanently available (always running) or only after the application was started by the user or using the Applauncher API the available attribute of the shared element can be used. If it is set to 'permanent' the application is always running, thus, the exposed functions can be found by using the discovery API at any time the hosting device is connected. All other values or the absent of the attribute results in unavailability of the service unless the application that exposes the service functions is started.

```
1 <widget xmlns="http://www.w3.org/ns/widgets"
2    xmlns:webinos="http://www.webinos.org/webinosapplication"
3    id="http://exampleapp.org/app1">
4        <webinos:shared available="permanent">
5             ...
6        </webinos:shared>
7        <content src="widget.html"/>
8 </widget>
```

To use functions exposed by webinos applications a reference to the application is needed. To get a reference to an object that provides access to the exposed functions the webinos JavaScript discovery API can be used while providing the query with the unique identifier of the API or application. The object returned by the webinos runtime is enriched with the Widget interface containing the application name, description, author, and version name beside of the features provided by the queried interface. For example accessing function 1 from the above example would in JavaScript look like:

```
 1   function showMap(location){
 2           //doing some logic
 3   }
 4
 5   function successCB(myLocationService) {
 6           alert('Service ' + myLocationService.displayName + ' ready to use');
 7
 8           //invoking a function exposed by the application
 9           myLocationService.webinos.extensions.getCurrentPosition(showMap);
10   }
11
12   function successCB2(myservice) {
13           alert('Service ' + object.displayName + ' ready to use');
14
```

```
15          //invoking a function exposed by the application
16          myservice.webinos.extensions.exampleFuntion();
17   }
18
19     window.webinos.findServices({api:'http://www.w3.org/ns/api-perms/geolocation'},
{onFound:successCB});
20                    window.webinos.findServices({api:'http://exampleapp.org/app1'},
{onFound:successCB2});
```

The actual functionality, the signature of the functions as well as the function names of the exposed functions must be known to the developer. A semantic description of the functions behaviour is out of scope of the specification. As you can see in the above example the default name space where an API is attached to in the returned object is webinos.extensions. Each exposed function is available using this prefix. If it is needed to make the exposed functions available under a specific path then this can be optionally defined using the api-path attribute of the shared-api element (e.g., if the application exposes a well known API, such as geolocation, that has a defined way of accessing it).

Example of Usage: Defining an API path for an exposed API

```
 1 <widget xmlns="http://www.w3.org/ns/widgets"
 2    xmlns:webinos="http://www.webinos.org/webinosapplication"
 3    id="http://exampleapp.org/app1">
 4        <webinos:shared>
 5                            <webinos:shared-api  api-name="http://www.w3.org/ns/api-
perms/geolocation" api-path="window.navigator.geolocation">
 6         <webinos:shared-function>watchPosition</webinos:shared-function>
 7         <webinos:shared-function>getCurrentPosition</webinos:shared-function>
 8         <webinos:shared-function>clearWatch</webinos:shared-function>
 9             </webinos:shared-api>
10        </webinos:shared>
11        <content src="widget.html"/>
12 </widget>
```

Using the above example makes the API available under myLocationService.navigator. geolocation.getCurrentPosition() instead of myLocationService.webinos.extensions. getCurrentPosition (showMap) when the api-path is not declared.

Interfacing between Child and Parent Applications

Since webinos supports a distributed application design the possibility of communication between application parts must be assured by the system. Webinos supports this by providing information about deployed child applications. As introduced in the previous section webinos provides a unique service ID for deployed child applications. This service ID can either be used for instantiating a remote binding to the child application in order to use exposed functions (if any) or it can be used to uniquely address an application within the webinos event API. Thus, a parent application can communicate with its child applications.

For the other way around, to let the child application know the unique identity of its parent application, the child application can be explicitly informed about the parent's identifier. Thus, after the parent receives a success callback it should instantiate the client and call a function which takes the parent's identifier using the getServiceId function from the ServiceDiscovery API. An exemplary flow could be:

```
 1
 2 function bindcallback(Service service){
 3
service.setParent(window.webinos.discovery.getServiceId("http://www.webinos.org/webino
sapplication"));
 4 };
 5
 6 function onError(in DeploymentError error){
 7        alert(error.code);
 8 };
 9
10 function onSuccess(in DOMString childID, in DOMString serviceID){
11       var service = window.webinos.discovery.createService();
12       service.bind({onBind:bindcallback},childID);
13 };
14
15 window.widget.deployChild(onSuccess, onError, "http://www.exampleapp.org/app1");
```

Hosted Webinos Applications

Apart from supporting installable applications, webinos supports the execution of hosted Web applications which does not require a permanent installation. Hosted applications can also make use of webinos-specific functionalities if the needed features are declared in the related application manifest. To attach metadata and configuration data to hosted applications, the W3C packaging and configuration is extended. The src attribute of the content element in the application configuration document of a widget file (.wgt) is allowed to point to an absolute path outside of the application package. In this case there is no need to include any other content in the package but it is still allowed. Thus, mixing local installed content and remote content is possible.

```
1 <widget xmlns="http://www.w3.org/ns/widgets">
2   <content src="http://www.hostedapps.com/hostedApp1.html"/>
3 </widget>
```

Making the hosted application available to the user works the same as installing a packaged webinos Web application. A .wgt package file is provided to the WRT and processed. If the WRT detects that an application is a hosted application the WRT has to add links to the application in order to make them accessible to the user. This, for example, can be done by placing the referenced application icon in an application gallery or by maintaining a bookmark list. The WRT has to inform the user about unavailability of a hosted application if there is no internet connection available.

In addition to the absolute path of the start document of a hosted application the scope of the application must be defined using the access element in order to allow access to needed documents and to apply related policies.

The access element is defined in the Widget Access Request Policy [WARP] which is applied for hosted applications.

Example of Usage: Application is defined using a root path

```
1 <widget xmlns="http://www.w3.org/ns/widgets">
2   <content src="http://www.hostedapps.com/hostedApp1/run.html"/>
3   <access origin="http://www.hostedapps.com/hostedApp1/"/>
4 </widget>
```

Example of Usage: Application is defined using absolute paths to the applications documents

```
1 <widget xmlns="http://www.w3.org/ns/widgets">
2   <content src="http://www.hostedapps.com/hostedApp2/run.html"/>
3   <access origin="http://www.hostedapps.com/hostedApp2/run.html"/>
4   <access origin="http://www.hostedapps.com/hostedApp2/style.css"/>
5   <access origin="http://www.hostedapps.com/hostedApp2/main.js"/>
6 </widget>
```

Formal Specification Webinos Web Runtime Environment

This section specifies a number of functional and non functional requirements related to the WRT itself. This includes how webinos applications are shared between devices, how the application life cycle is handled, how webinos applications can be installed on WRTs, and which Web technologies must be supported by webinos WRTs in order to define a common set of available functionality.

Inter-Runtime Application Deployment

Webinos provides the ability to exchange webinos applications between devices including export of applications to the file system to be manually installed on another device and automatic installation using provided WRT functionalities. This includes whole application packages as well as child application packages which are included in a main application package. If a WRT is asked by the user to install a select application on another device the WRT has to provided a meaningful UI for selecting a target device which includes, e.g, discovery of nearby devices or of devices in the same personal zone.

For installing application on other devices within webinos the following events are defined and must be supported by the WRT. The same mechanism is applied if an application for remote deployment is selected using the JavaScript Widget API for code deployment or the automatic deployment during application installation time is applied.

Requesting a Remote Installation on another device:

Event type must be: http://webinos.org/events/application/request-installation
Event payload must be a JSON object with following attributes:

```
 1 {
 2     name: [the name of the application that should be remotely installed]
 3       description: [the description of the application that should be remotely
installed]
 4     version: [the version of the application that should be remotely installed]
 5     author: [the author of the application that should be remotely installed]
 6     id: [the application ID of the application that should be remotely installed]
 7     size: [the size of the application package in number of bytes]
 8        uri: [optional, an URI where the WRT that is requested to install an
application can retrieve the application code]
```

```
 9        payloadInstallation: [optional, TRUE or FALSE while true means that an
application transmission using the webinos event mechanism is requested, e.g, if an
uri cannot be provided]
10 }
```

Answer to a remote installation request:

Event type must be: http://webinos.org/events/application/request-installation-response
Event payload must be a JSON object with following attributes:

```
1 {
2       payloadInstallation: [optional, TRUE if an application transmission using the
webinos event mechanism is accepted]
3      code: [optional if payloadInstallation is used, error code as specified in the
webinos widget specification extension or 1 for a successful installation]
4     id: [the application ID of the application that should be remotely installed]
5         uniqueID: [optional if payloadInstallation is used, a unique id of the
application that represents exactly this application installation which can be used
for service discovery or remote application launch]
6 }
```

Sending application using payload:

Event type must be: http://webinos.org/events/application/payload-installation?id=[application id, must be the same as previously requested in request-installation otherwise the event must be dropped]
Event payload: [the binary data of the application]

After installation another request-installation-response must be sent to the initiating device.

Application Life Cycle

Webinos applications can be made available through a number of different deployment methods including installation via Web sites, application stores, file system, other webinos Web runtimes, simple application sharing and application advertisement, and execution of hosted applications (no installation). Upon installation of a webinos application package, the Web runtime must process the package as specified in the webinos application packaging specification. Additionally a webinos Web runtime must implement the following requirements.

WRT-05: The WRT **MUST** sent a User Agent Identification containing vendor, name, version of WRT with each HTTP/HTTPS request to be used for identifying available features provided by the WRT.

WRT-06: If possible the webinos WRT **MUST** catch content which is of MIME type application/widget in order to install the application or execute the application if already installed and up to date.

WRT-07: If possible the webinos WRT **MUST** catch the invocation of files with a .wgt extension in order to install the application or execute the application if already installed and up to date.

WRT-08: The WRT **MAY** check the applications configuration document for compatibility with the features provided by the runtime.

WRT-09: The WRT **MUST** provide means to install locally available applications on another device which can be selected by the user in conformance with section 2 of this specification.

WRT-10: The WRT **MUST** provide a list of application available to the user for local installation or remote usage.

WRT-11: The WRT **MUST** delete all data specific to an application if the application is uninstalled. This includes the un-installation of any child applications if they are depended on the parent application.

WRT-12: The WRT **MUST** use secure storage for webinos applications that are marked as copy protected. This should not allow to view, export, modify, or any other access of the application by other applications or the user (WAC).

WRT-13: The WRT **MUST** use encrypted storage for webinos applications in case that external or general accessible storage space is used (WAC) for storing application data.

Notify Widgets to Web Browsers

It is possible to notify the availability of widgets to Web browsers using the HTML <link> element in common Web sites. Thus, common Web Browsers may show the availability of installable applications to the user including the possibility of installing them into the system. In advance, the browser should check if a Widget handler, e.g., a webinos WRT, is registered in the system. The <link> element's type attribute must be set to "application/widget" to define the MIME type of the linked resource. The *title* attribute should be set to the widgets title. The *href* attribute must contain the link to the application package. The link type defined by the *rel* attribute must be set to "alternate".

```
1 <link rel="alternate"
2   type="application/widget"
3   title="Application Title"
4   href="http://www.applicationhost.com/application.wgt">
5 </link>
```

Life Cycle API

Webinos provides some APIs to allow developers to manage their application's life cycle during application execution. While the application itself cannot influence its execution life cycle status Webinos allows for registering callbacks in case the application will be destroyed and can't be resumed, will go to background/foreground or will be stopped/started again or resumed.

```
 1 interface NotifySuccessCallback{
 2
 3     //Called if an event was accepted by the user. If provided, the notification id
is passed in to link the success to a specific event
 4     onSuccess(in DOMString id);
 5 }
 6
 7 interface Widget {
 8
 9     //allows an application to trigger calling destroy from the runtime
10     void exit();
```

```
11
12      //sends the application to background if possible so that it is not visible to
the user anymore
13      //if possible by the platform the application execution goes on
14      void hide();
15
16      //asks the WRT wheather the application is currently hidden (not visible to the
user) or not
17      //if the application is hidden it may notify an event to the user using notify
18      boolean isHidden();
19
20       //Triggers the WRT to notify occurrence of an event, as described using the
parameters, to the user
21       //The user can click the event. If the application is in background and the
user accepted the event
22         //, e.g., by clicking on it, the application must be brought back to
foreground. The notify success
23      //callback is then called after onForeground was called.
24      void notify(in NotifySuccessCallback onSuccess, in NotifyErrorCallback onError,
in DOMString title, in optional DOMString shortDescription, in optional DOMString id,
in optional DOMString icon);
25
26       //To cancel a previous notify because it is updated or expired (if ongoing /
not clicked by the user)
27      void cancelNotify(in DOMString id);
28
29      //Callback function which is called if the application will be shut down by the
WRT. All application memory
30        //assigned to the application will be freed after returning out of this
function.
31      void onDestroy();
32
33         //Callback function which is called after the application was put to
background, e.g., another application
34      //goes to foreground and the application is not visible any more. After calling
onBackground the application
35      //is still running but not visible anymore.
36      void onBackground();
37
38      //Callback function which is called if the application goes to foreground after
previously going to background.
39      void onForeground();
40
41       //Callback function which is called if application execution is stopped by the
WRT.
42      void onStop();
43
44      //Callback function which is called if application execution is continued after
previously interrupted.
45      void onStart();
46
47 };
```

Application Update

[WidgetUpdate] defines how packaged Web applications (Widgets) can be updated over http. For hosted Web applications without any content in the application package there is no need for local

update checks. All updates are applied directly remotely on the hosting Web server and locally applied at the next execution of the application.

The webinos WRT **MUST** be capable of updating webinos application packages as defined in W3C Widget Updates over HTTP [WidgetUpdate].

Child applications may also be updated using [WidgetUpdate]. Another option for can be re-installing the child application which is initiated by it's parent application which was previously updated using [WidgetUpdate]. Here, the new version of the application that should be re-installed must be different from the version currently installed. Otherwise its rejected.

Application de-installation

The WRT must provide functionalities to permanently remove applications from the device on demand of the user. It may appear the child-applications on other devices become non-functional without their parent application. The developer can declare this within the configuration document in the child element. If the optional attribute *parentneeded* is set to true the WRT has to store the application instance IDs provided by the remote installation process. These IDs must be used to request de-installation on remote devices in case the parent application is deleted.

Example of Usage:

```
1 <widget xmlns="http://www.w3.org/ns/widgets"
2    xmlns:webinos="http://www.webinos.org/webinosapplication"
3    id="http://exampleapp.org/app1" webinos:type="container">
4        <webinos:child webinos:parentneeded="true">child2.wgt</webinos:child>
5 </widget>
```

Events for remote de-installation

Event type must be: http://webinos.org/events/application/request-deinstallation

Event payload must be a JSON object with following attributes:

```
1 {
2     id: [the application ID of the application that should be deleted]
3      uniqueID: [the unique instance id of the application that represents exactly
this application installation]
4 }
```

Event type must be: http://webinos.org/events/application/request-deinstallation-response

Event payload must be a JSON object with following attributes:

```
1 {
2      code: [error code as specified in the DeploymentError of the webinos widget
specification or 1 in case of successfull de-installation]
3     id: [the application ID of the application that should be deleted]
4      uniqueID: [the unique instance id of the application that represents exactly
this application installation]
5 }
```

In case a remote de-installation was not successful the WRT has to inform the user about this and about that a manual de-installation may be needed.

Automatic execution of Applications

Beside user driven application execution, webinos applications can be automatically initiated by events coming from webinos itself or from other applications. Webinos WRT allows registering for event types which triggers the execution of the application. Events for automatic execution can be application specific ones or predefine webinos events. Subscription to application execution events can be made programmatically using the webinos Event API and described using a new webinos element as child of the <content> element in the applications configuration document.

Example of Usage: Descriptive registration for application execution events

```
1 <widget xmlns="http://www.w3.org/ns/widgets"
xmlns:webinos="http://www.webinos.org/webinosapplication" webinos:type="background">
2   <content src="http://www.hostedapps.com/hostedApp2/run.js"/ >
3     <webinos:start>http://webinos.org/events/core/BOOT_UP_COMPLETED</webinos:start>
4
<webinos:start>http://www.hostedapps.com/hostedApp2/eventtypes/event1</webinos:start>
5   </content>
6 </widget>
```

Predefined Events that must be supported by the WRT:

http://webinos.org/events/core/BOOT_UP_COMPLETED
If the device and the WRT is ready to execute applications the WRT must auto start applications registered to this event.

Applications without UI

Webinos application can be executed in a no user interaction (UI) mode which means that the application is invisible to the user and the user cannot directly interact with the application. After starting a background application it will be executed and is responsive to potential incoming request as long as the application is running (not stopped by the user or by the service itself). To express that an application is a no UI application type *type* attribute is added to the <widget> element of the application's configuration document. If *type* is 'background' the application is marked as no UI / background application. In any other cases the application is handled as normal application. The WRT must provide means to manage background applications by the user. E.g., start a background application, show running background applications and terminate background applications.

Example of Usage: Declaring an application as background application

```
1 <widget xmlns="http://www.w3.org/ns/widgets"
xmlns:webinos="http://www.webinos.org/webinosapplication" webinos:type="background">
2   <content src="http://www.hostedapps.com/hostedApp2/run.js"/>
3 </widget>
```

Extensions

The webinos runtime must support the NPAPI standard. The specification to build NPAPI plugins and a NPAPI compliant runtime is out of scope of this document. The NPAPI plug-ins and a NPAPI compliant runtime are specified in [NPAPI-Browser-Side-API], [NPAPI-Plugin-Side-API], and [Npruntime].

This specification covers how to declare a NPAPI plug-in as an extension in the application manifest, how the installation should be handled by the webinos runtime and how functions of an extension can be made available to other applications. Security aspects of extensions are covered in WP 3.5

Bundeling extensions to an application package allows us to manage them in the same way as regular applications (cf. LC-ASP-ISMB-112). Furhtermore it allows us to expose functions of the extension using the same mechansims as for exposing and sharing functions of applications and features (cf. LC-DWP-ISMB-009).

Integrating a NPAPI plug-in into an application package

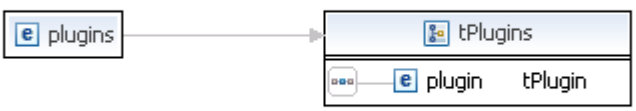
Webinos allows you to integrate a NPAPI plug-in into your application package. The extension is then distributed with the application itself. The installation of the plug-in is handled by the webinos platform. In order to enable the runtime to handle the installation of the plug-in, some metadata has to be specified inside the application manifest including the name, location of the binary files, and supported platforms. This meta data is needed for the lifecylemanagement of the plug-in.

The following example illustrates how an application description making use of this feature looks like.

```
 1 <webinos:plugins>
 2     <webinos:plugin>
 3         <webinos:name>foo</webinos:name>
 4         <webinos:platforms>
 5             <webinos:platform>
 6                 <webinos:name>win32</webinos:name>
 7                 <webinos:path>plugins/win32/foo.dll</webinos:name>
 8             </webinos:platform>
 9             <webinos:platform>
10                 <webinos:name>linux</webinos:name>
11                 <webinos:path>plugins/linux/foo.o</webinos:name>
12             </webinos:platform>
13         </platforms>
14     </webinos:plugin>
15 </webinos:plugins>
```

The plugins element

The plugins element lists all plug-ins, which are part of the application package. An application package can contain more than one plug-in.



Context in which this element is used: As a child of the widget element.

Content model: complex type of tPlugins
Sequence: zero or one

The complex type of tPlugins is defined as followed

```
1 <complexType name="tPlugins">
2     <sequence>
3         <element name="plugin" type="webinos:tPlugin"></element>
4     </sequence>
5 </complexType>
```

The plugin element

The plugin element defines the information about a single plug-in.

Context in which this element is used: As a child of the plugins element.
Content model: complex type tPlugin
Seqeunce: one ore many
Attributes: none

The complex type for a tPlugin is defined as followed:

```
1 <complexType name="tPlugin">
2     <sequence minOccurs="1" maxOccurs="1">
3         <element name="name" type="string"></element>
4         <sequence minOccurs="1" maxOccurs="unbounded">
5             <element name="platforms" type="webinos:tPlatform"></element>
6         </sequence>
7     </sequence>
8 </complexType>
```

name element of plugin:

Defines the name of the plugin.

Context in which this element is used: As a child of a plugin element.
Content model: string

Occurrences: one
Attributes: none

platforms element of plugin

The platforms element is the container for the defintion of the supported platforms by the plug-in

Context in which this element is used: As a child of a plugin element.
Content model: complex type tPlatforms
Occurence: one
Attributes: none

The complex type tPlatforms is defined as followed:

```
1     <complexType name="tPlatforms">
2         <sequence>
3             <element name="platform" type="webinos:tPlatform"></element>
4         </sequence>
5     </complexType>
```

platform element of platforms

Defines a supported platform for plugin inlcuding the platform specific binary.

Context in which this element is used: As a child of the platforms element.
Content model: complex type tPlatforms
Occurence: one or many
Attributes: none

The complex type tPlatform is defined as followed:

```
1 <complexType name="tPlatform">
2     <sequence minOccurs="1" maxOccurs="1">
3         <element name="name" type="string" minOccurs="1" maxOccurs="1"></element>
4         <element name="path" type="string" minOccurs="1" maxOccurs="1"></element>
5     </sequence>
6 </complexType>
```

name of platform

Defines the platform name of the binary

Context in which this element is used: As a child of a platform element.
Content model: String
Sequence: one

path of platform

Defines the path to binary for the specific platform relative to the location of the application manifest.

Context in which this element is used: As a child of a platform element.
Content model: String
Sequence: one

XML-Schema for plug-in packaging

The following code shows extension specific application meta data in form of a XML schema.

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2         <schema         targetNamespace="http://www.webinos.org/webinosapplication"
elementFormDefault="qualified"                xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:webinos="http://www.webinos.org/webinosapplication">
 3     <complexType name="tPlugins">
 4         <sequence>
 5             <element name="plugin" type="webinos:tPlugin"></element>
 6         </sequence>
 7     </complexType>
 8     <element name="plugins" type="webinos:tPlugins"></element>
 9     <complexType name="tPlugin">
10         <sequence minOccurs="1" maxOccurs="1">
11             <element name="name" type="string"></element>
12             <element name="platforms" type="webinos:tPlatform"></element>
13         </sequence>
14
15     </complexType>
16     <complexType name="tPlatform">
17         <sequence minOccurs="1" maxOccurs="1">
18                                 <element   name="name"   type="string"   minOccurs="1"
maxOccurs="1"></element>
19                                 <element   name="path"   type="string"   minOccurs="1"
maxOccurs="1"></element>
20         </sequence>
21     </complexType>
22
23     <complexType name="tPlatforms">
24         <sequence>
25             <element name="platform" type="webinos:tPlatform"></element>
26         </sequence>
27     </complexType>
28 </schema>
```

Remote usage of plug-in functions

The remote usage of the plug-in can be enabled by building a JavaScript wrapper around the plug-in itself and exposing the functionality as described in Exposing in Application Functionalities as Service to other Application.

The following example illustrates how a NPAPI method is being made available to other applications running in the same personal zone.

Application manifest

```
 1                    <widget                      xmlns="http://www.w3.org/ns/widgets"
xmlns:webinos="http://www.webinos.org/webinosapplication"
id="http://exampleapp.org/app1">
 2     <content src="widget.html"/>
 3     <webinos:copy-restricted webinos:restricted-to="personal-zone"/>
 4     <webinos:shared>
 5                                  <webinos:shared-function    visibility="personal-
zone">myExposedPluginFunction</webinos:shared-function>
 6        </webinos:shared>
 7        <webinos:plugins>
 8         <webinos:plugin>
 9             <webinos:name>foo</webinos:name>
10             <webinos:platforms>
11                 <webinos:platform>
12                     <webinos:name>linux</webinos:name>
13                     <webinos:path>plugins/meego/foo.so</webinos:name>
14                 </webinos:platform>
15             </platforms>
16         </webinos:plugin>
17     </webinos:plugins>
18 </widget>
```

The application packages conatins a NPAPI plug-in called foo (cf. line 9), which is supported on linux (cf. line 12). Furthermore the application exposes a JavaScript function callled myExposedPluginFunction (cf. line 5). This JavaScript function is wrapper function for an exposed NPAPI function as shown in the following code snipplet.

content.html

```
 1 <html>
 2     <head>
 3     </head>
 4     <body>
 5         <script>
 6             var myPlugin;
 7                    if  (navigator.mimeTypes["application/webinos-extension-foo"]  &&
navigator.mimeTypes["application/webinos-extension-foo"].enabledPlugin != null){
 8                     document.write('<embed type="application/webinos-extension-foo"
id="plugin">');
 9                 myPlugin = document.getElementById('plugin');
10             }else{
11                 myPlugin = null;
12             }
13             function myExposedPluginFunction(){
14                 if(myPlugin != null){
15                     return myPlugin.getSomething();
16                 }else{
17                     return throw "method not supported";
18                 }
19             }
20         </script>
21     </body>
22 </html>
```

In line 7 the application checks, if a plugin exists for a speicifc MIME type. If so, an embed object is added to the DOM (cf. line 8), which makes it possible to call methods of a NPAPI plugin from JavaScript. In line 15 the exposed NPAPI method doSomenting is called. If the myExposedPluginFunction() is called and the MIMEType is not supported, the method throws an error (cf. line 17).

Installation and execution of an extension

During the installion process of the application, the webinos runtime determines the correct NPAPI binary for the platform and stores it on the local machine, so that rendering engine can iniate the plugin, when executing the application.
Since NPAPI plug-ins are indified by the supporting MIME-Type inside a Web rendering engine, the webinos runtime must be able to associate each NPAPI plug-in to a webinos application and make only those installed plug-ins available which are associated to an application.

Security considerations

We recommend that only applications using plug-in are only executed if they have been signed and approved by some entity. The signing process is defined in WP 3.5

Web Technologies

Note: The following section is preliminary and may be subject to change in future versions of the specification.

A Web runtime which is able to render webinos applications and wants to claim to be compliant to the webinos specification must support the following Web technologies, whereas the concrete mandatory version and feature set of each one is further defined in the Wholesale Application Community Core Specification Version 2.0 Section 2 [WACWebTech].

- XML and HTML Markup Language
- JavaScript (JS)
- Cascading Style Sheets (CSS)
- XmlHttpRequest
- Scalable Vector Graphics (SVG)

Beside of the definition of Web technologies to be supported [WACWebTech] defines URI schemes to allow launching of specific applications. The following URI schemes must be supported by webinos runtimes:

- "http/https" in order to launch a Web browser

- "tel" in order to initiate a call
- "sms" in order to send a short message service (SMS) message
- "mmsto" in order to send a multimedia massaging service (MMS) message
- "mailto" in order to send an email message
- "data" in order to directly embed content (e.g., /9j/4AAQSkZJRgABAQAAA...)

Authentication and Identity

The authentication and identity mechanisms within webinos form the critical foundations upon which the many of the more sophisticated functions are built.

In the preceding sections the concepts of the personal zone (both hub and proxy) have been explained. In this section the details of the authentication mechanism shall be first gently introduced (in the conceptual architecture section) then formally specified. In deliverable D3.5 the reasoning behind these architecture decision and threats to security are discussed in detail.

Within webinos, the authentication and identity issues need to be addressed at two distinct levels.

1. Intra webinos authentication: the mechanisms via which users and devices are authenticated by the personal zone
2. Extra webinos authentication: which describe some utility capability, where the personal zone hub can assist in the authentication against multiple external Web applications and services, which are not necessarily webinos based.

We have decided to distinguish these two levels, as within webinos the zone-based authentication integrates perfectly in the architecture and it meets all the authentication requirements. However, on the open Internet, strong authentication which involves a third party to prove the identity of a user is required. Thus different means of authentication will be implemented which can also be used for legally binding agreements on the Internet.

In addition to that, authentication is done in two steps: first the user is authenticated to at least one of his devices. Second the device communicates on behalf of the user identifying itself with its public key and its certificate.

We will deal with intra webinos authentication in this specification. Extra webinos authentication will be dealt with in phase II of the webinos project. But before we do so, let us remind ourselves of the roles and responsibilities of the PZP and PZH, with respect the the authentication and identity problem.

Personal Zone:

The personal zone is a conceptual entity, that has meaning to an end user, but from an implementation perspective is built up of one (and only one) personal zone hub, and many personal zone proxies.

From end users perspective the personal zone hub has the following qualities:

- Web addressable: the personal zone hub is identified by a unique URI.
- Permanently addressable: the personal zone hub should be "permanently" addressable on the open internet. It is presumed to be highly available.
    - SEC-NOTE: it there fore must be robust to denial of service attacks

- Outgoing messages: the personal zone hub becomes the intelligent agent through which all outgoing webinos messages are proxied, if there is no peer-to-peer communication set-up between two devices after authentication
- Incoming messages: the personal zone hub is the entity to which all incoming messages are directed, and therefore takes responsibility for directing the messages to the right place.
    - in the cases where devices have set-up a peer-to-peer connection, incoming traffic is only routed through the PZH during authentication
- Secure intermediary: the personal zone hub is the end user's policy enforcement point. It should be aware of and police the end user preferences, with respect to any security relevant action.
- Secure perimeter: as far as the user is concerned, the personal zone hub is a secure perimeter. People, devices and applications that are granted access to the personal zone are presumed to be trusted.
- Authentication: the personal zone hub is the entity against which all devices, users, applications must authenticate (identify) themselves in order to be granted "inter-zone" rights

Each device in the personal zone has its own unique public/private cryptographic key pair. Once the user is authenticated to a device, the device reveals access to the secure storage which keeps the private key in a protected environment. The PZP is the only entity which can access the private key. The PZP uses the private key for mutual authentication within the zone and across zones, and for integrity and confidentiality of communication between devices.

Personal Zone Hub:

The personal zone hub is a server based entity that orchestrates the behaviour all of the personal zone proxies, in order to deliver the functionality expected from the personal zone hub.

The PZH acts as a master repository for critical data that must be synchronised between hub and proxies, in order to deliver the required on-line and off-line functionality.

When being messaged from devices on the open internet, the PZH acts like a DNS server, finding the most relevant end device application, to which the incoming webinos messages should be routed.

The PZH acts as certification authority (CA) for the entire zone. Each device has a certificate which is issued by the PZH once the device joins the zone.

Personal Zone Proxy:

An application rarely interacts with the PZH directly, in most cases the application interacts with a PZH via the PZP.

The PZP therefore intercepts and either directly deals with or forwards the authentication requests.

The PZP intercepts and forwards all outgoing messages.

The PZP receives and forwards on all incoming messages

The PZP synchronises key information with the PZH to allow it to perform its functions. This data includes

- critical user identity information (to assist discovery) e.g. email, first name, last name etc
- webinos PZH authentication tokens
- extra webinos, 3rd party service authentication tokens (webinos Phase II)
- identity tokens for trusted devices
- identity tokens for trusted applications
- identity tokens for trusted people
- identity tokens for services that can be accessed from other people
- all policy description files
- context data
- session data
- application data

The webinos-related tokens are represented by certificates. Tokens for 3rd-party services are authorisation tokens which are issued by an identity provider. Synchronisation is performed as soon as a PZP established a secure channel with the PZH and whenever data changes while being connected.

Intra webinos authentication

Intra webinos authentication covers all communication within one personal zone or across multiple personal zones. Authentication is always the same. First the user is authenticated to the device. Second the device authenticates to one or more other devices. These other devices can be in the same or in another zone. A device proves its affiliation to a zone by holding a certificate which is issued by the PZH of that zone. In addition, the device possesses the private key of the key pair of which the public key is contained in the certificate.

PZP installation process

The PZP is a fundamental and trusted component of a personal zone. For proper security of the entire personal zone, we must address the issue of how does the PZP get installed upon the device.

The PZP needs to be obtained from a trusted source. The PZH is the most suitable source as it also hosts the user data which is to be synchronised to the PZP later. On the device which is to be joined to the

zone, the PZH needs to be identified and authenticated. Thereafter, the PZP is obtained and installed. Before and after installation the integrity of the PZP needs to be validated.

During the installation process, the PZP generates its new and unique public/private key pair. It stores the private key in its secure storage and submits the public key to the PZH. The PZH stores the public key and issues a certificate for the PZP. Once the PZP obtained the certificate, it can prove to other entities that it is a legitimate affiliate of the zone.

Before installation, it is required to ensure that the PZP talks to the desired PZH and vice versa. Similar to the case where communication between two personal zones is set-up, a separate channel is used to verify that the communication end points are the desired ones. First, the user authenticates at the PZH and executes the *add device* feature there. Second, the user enters the identifier (URI) of the PZH on the new device. Third, the PZH and the new device establish a secure connection via Diffie-Hellman key exchange.

Fourth, the PZH displays a random number which it also sends to the new device. The user has to verify that both are the same. Fifth, the user downloads and installs the PZP as described above. Since the new device may not have a PZP yet, all the steps can be performed in an ordinary Web browser on the new device.

The PZP is conceptually and architecturally distinct from the webinos Web runtime. A specific implementation may bind the two elements together in a single executable. For a clear delineation of the roles and responsibilities of the PZP and webinos WRT see webinos key architectural components

More on bindings between device, PZP and application, as well as attestation and further related topics can be found in the deliverable D3.5.

Trusting a Web application

A particular Web application may be either "in zone" or "out of zone". The first opportunity to trust an application should be at install time. Either the user must be explicitly asked, or there must be a policy rule that states that this application is "zone trusted".

The results of this trusting process, is that the application identifier is placed into the trusted application cache of the PZP which is synchronised with the PZH.

The set of applications that are deemed zone trusted are defined within the policy definition.

Identifiers

In webinos devices and applications are identified independently.

Application Identifiers

Applications need to be uniquely identified. The application identification mechanisms will be based upon the frameworks discussed in the foundations section. Applications are identified as defined in W3C

Widget. The application ID is defined by the developer plus optional developer signatures are possible. Either the raw identifier, or a hashed token based upon the identifier must be stored in the PZP trusted application cache.

Device identifier

Each device needs to be uniquely identified. When possible, webinos can exploit already available unique identifiers, e.g.

- mobile phone
  - IMEI
  - IMSI
- PC
  - mac address
  - hard drive ID

If a trusted, uniquely defined device identifier proves impossible to define, PZH can issue a unique identifier at the point of PZP installation.

These identifiers could be useful also for authentication purpose, but should be used in a security-wise process, to ensure protection against fake identifiers. An address is hard to forge only if contains some part of (or derives from) a shared secret. Moreover, even if not easy to forge, it is not completely secure if transferred along an insecure channel or re-used more then one time, since it is exposed to the risk of being sniffed and thus re-used by a malicious user.
For these reasons, for example, Mac addresses and hard drive IDs are not hard to spoof identifiers. However, on the base of the security requirement for the scenario (e.g. it is not a privacy problem to expose universal identifiers) and on the authentication mechanisms (i.e. the identity will be confirmed by some challenges based on a some direct or indirect established trusted relationships, e.g. certificates signed by a trusted third party) the overall architecture remains robust and secure

User Authentication levels

Within webinos a user will be authenticated to different levels. At least three levels will be supported by default. The policy definition can support more. The supported levels are:

- Anonymous: the lowest level, where no user authentication has happened. The PZP exist and is running, but it has no idea who is using the device it is on
- Device inherited authenticated: here it is implied that the PZP has integrated with the authentication mechanism on the device already. For a PC this could be the user name password challenge. For a mobile phone this could be PIN lock. The device inherited level will ascribe a *low* or *high* authentication confidence depending upon the mechanisms active
- Elevated webinos authentication: this is where the PZP directly challenges the user, using the authentication plug-in architecture defined below.

The device inherited authentication carries with it no specification, but a list of recommendations or requirements. These requirements define what is minimally required for inherited authentication in webinos. Any device which wishes to be compliant to webinos **MUST** implement these requirements. These requirements are:

- The device has built-in secure storage. Sensitive data, such as the private key of the PZP is stored therein. The secure storage at least **MUST** encrypt data which is stored there with a strong encryption algorithm and a suitable key length. ASE256 **SHOULD** be used. The storage **SHOULD** be a combination of hardware protection and software protection (i.e. encryption). Combining the software protection with tamper-resistant hardware protection (such as the SIM card in a mobile phone) provides the best possible protection of the data and is preferred therefore. For secure storage, see also deliverable D3.5.
- There is an easy-to-use remote command to delete the content of the secure storage for the case the device is stolen or lost.

Technically, the remote delete command at the PZH should result in the PZH deleting the cached authentication token of the PZP-device.
This in turn leads to the PZH-PZP synchronisation deleting the tied authentication tokens

The behaviour of a PZP, is defined so that it its PZH authentication token is deleted, it **MUST** delete all cached user/PZH data from the cache and revert to an anonymous PZP.

- The device **MUST** authenticate the user by one or more commonly accepted state-of-the-art authentication mechanism. This authentication **MUST** prevent unauthorised entities from using the device and from accessing resources thereon. Authentication can be done by any method which is useful and suitable for the device. Examples are:
    - username and password: the password has to be long enough and complex enough
    - PIN protection: the PIN has to be long enough and not easy to guess; a low number of maximally permitted attempts is to be set
    - biometry: a finger print or iris scanner which is difficult to fool.
    - posession of a token which is attached to the device while using it
- There **MUST NOT** be any way to bypass authentication
- The authentication mechanism implements an interface which is exposed to the PZP, where the PZP can:
    - request user authentication
    - determine the state of user authentication (authenticated, not authenticated, method by which the user was authenticated, what time ago the user was authenticated)
    - event-based notification of user authentication each time the user authenticates without request by the PZP

- Only the PZP has access to the secure storage

A user always authenticates with one or more of the user's devices belonging to the personal zone. Any communication performed by the devices happens on behalf of the authenticated user. Devices authenticate themselves as the personal zone, backed by the personal zone's certificate. If the user is not authenticated, functionality of the device is limited to local applications and to the use of services which do not need authentication. Examples for non-authenticated usage are watching TV, browsing the Internet, listening to locally stored music. As soon as remote webinos services are used or PZH communication is required, the user **MUST** be authenticated.

An internal plug-in interface will be defined on the PZP (and optional protocols to the PZH) to allow for new webinos authentication mechanisms to be defined.

JavaScript APIs are defined, to allow an application to both

1. query the current authentication level
2. request a (re-)authentication

Authentication Process

The user authenticates towards the device by means which are provided by the device. Once the user is authenticated, the PZP has access to the private key of the public/private key pair which identifies the device. As soon as the PZH has issued a certificate that proves that the device belongs to the PZH, it implies that the device belongs to the personal zone.

The PZP public/private key pair is generated by the PZP itself, then a certificate request (along with the PZP public key) is submitted to the PZH, which issues a certificate (with the required attributes and the public key of the PZP) signed by PZH private key and give it back to requesting PZP. In this way, the PZP private key never left the local device secure storage, so not exposing this key to chance of disclosure while in transit.

The PZP can then establish a secure communication between itself and the PZP of another device. In webinos -- as it is based on Web technologies -- there is a TCP connection whenever two devices communicate. On top of this connection, a TLS channel is set-up. This channel authenticates both the endpoints of the communication, and it ensures integrity and confidentiality of the communication. Since the PZP needs its private key to establish the TLS channel, the user has to be authenticated in order to enable access to the secure storage where the private key is stored. On top of this secure communication channel, the application can use any application layer protocol (e.g. HTTP, XMPP, RTP, RTCP).

Once the TLS channel is established, applications can call APIs of services via this channel without the need to deal with authentication. Authentication is completely transparent to the developer. The fact that the TLS channel exists implies that the user is authenticated to at least one of the devices, that the devices belong to the same or to a trusted personal zone and that the devices are authenticated. The policy will take care of access control on the API level.

The same kind of secure communication channel is also set-up between each device's PZP and the zone's PZH. This is done by default whenever both the device can reach the PZH and the user is authenticated to the device.

Apart from the TLS protocol, there is no need for additional protocols or messaging for authentication between PZPs or PZP and PZH.

User identifiers

For user visible identifiers we use:

- the URI of the PZH, or
- some piece of data (e.g. the user's email address) that reliably resolves (e.g. via Webfinger) to a single PZH URI.

For the internal implementation of the PZH <-> PZP trusted user cache and synchronisation, the URI which identifies the PZH of a user is mapped to the cryptographic key of the PZP. This is stored in the trusted user's cache on the PZP/PZH.

Trusting new users

Within a webinos personal zone, applications need to be verified against the zone. Devices also need to be verified against the zone, but this is implicit in the PZP installation process described above. The PZH holds a list of installed applications per device. an application is only added to this list if it was installed and verified properly. User consent is required for installation and the security policy needs to permit installation.

Some of the more interesting use cases, however are only possible, when we can communicate between zones. This requires the establishment of trust between zones, i.e. users.

The mechanism via which trust is established between two people is a protocol that requires the permission from both parties in the transaction. First, they need to make sure that their devices (each belonging to one user's personal zone) are indeed communicating with each other. This is done by verifying the two certificate's fingerprints via a separate channel (e.g. by reading it out on the phone) or by typing a randomly generated PIN. Second, a secure communication channel is established. Third, if both users accept, the certificates are exchanged and stored in the PZH. When trust has been established between users, then each PZH will cache the other PZH's certificate in the trusted users cache. This defines the other PZH and thus the entire personal zone as trusted.

Even though a trust relationship exists between two people (i.e. certificates are mutually exchanged), every subsequent communication must still be subject to the security policy in force on both participating personal zones.

The initial step of making sure that the expected devices indeed communicate with each other is supported by webinos' social relationship concept. Users typically know another already and they also know some of their identifiers (e.g. a phone number, an email address, the nickname in a social network). This information can be used in webinos to reveal the URI of the user's PZH. The PZH then knows the user's devices are to be reached. When devices of two users are starting to communicate for the first time, each user has to permit the other user to access the list of devices on the PZH to directly communicate to the given device thereafter. This is a preparatory step for personal zone binding in the case of no local proximity in an easy-to-use manner.

In case of local proximity, personal area network technologies or near field communication can be used to initiate communication of the two devices of the two different zones. For example, Bluetooth, a local wireless network or RFID chips built into the device can be used.

In both cases, social relationship and local proximity, users need to verify that they communicate with the intended devices by exchanging some information via another channel, as described in the process above.

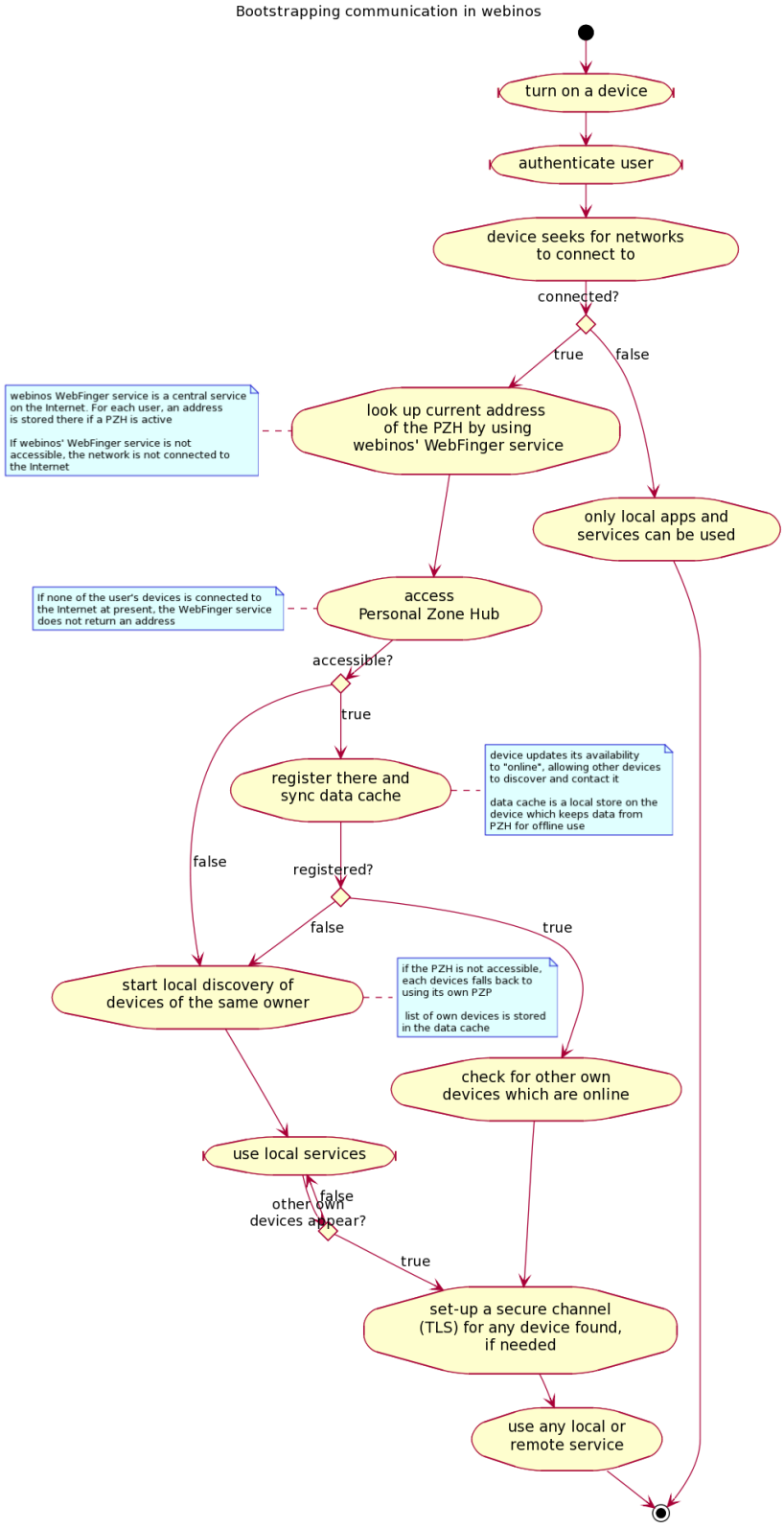
Technical Use Cases

The following discrete technical use cases help us understand the precise technology that is required to implement the webinos functionality.

Attached to each use case is a set of sequence diagrams that explores the logical flows and helps describe the implied roles and responsibilities of each architectural entity. Some of the ones presented here (namely, scenario 2, 4 and 9) accordingly to the current architecture are very similar to other cases, so they are no longer discussed here.

Overview

The activity diagram shows how communication between devices in a zone is bootstrapped once the device is turned on.



The following sequence diagrams describe the logic flow with emphasis on the authentication exchange description. So, the flow is quite at high level in some part. It is expected that to have a full picture, the reader knows other part of this deliverable, in particular Discovery for more precise discovery exchanges, and high level architecture (e.g., for synchronization discussion between PZP and PZH). The service provisioning is just a possible example of service, details of that are out of this section's scope.

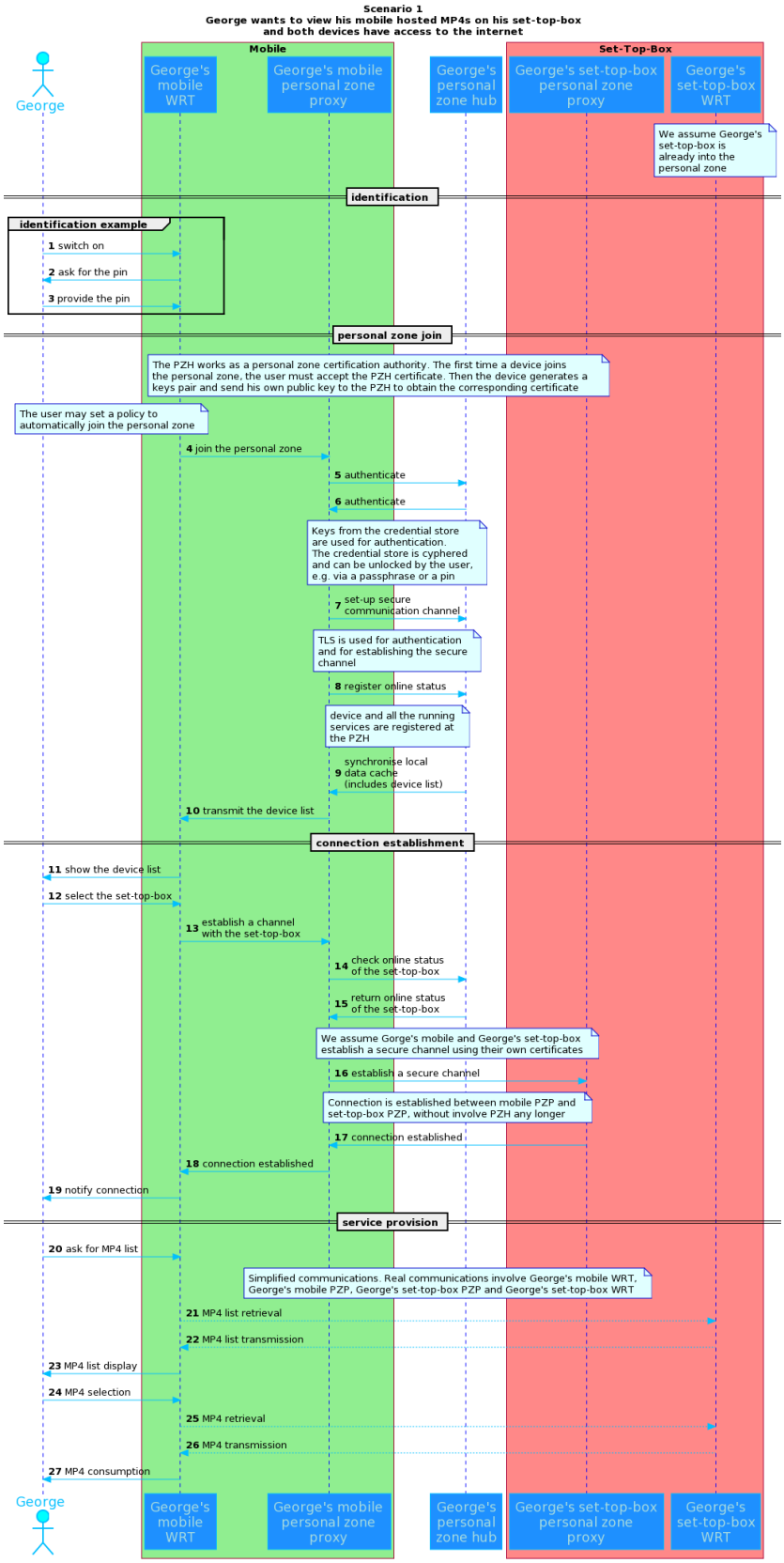
Scenario 1 - Same Personal Zone

George wants to view his mobile hosted MP4s on his set top box and both devices have access to the internet

Assumptions

- George **owns** both devices
- George is going to trigger and control the playback from his mobile device
- The Settop box therefore acts as a slave, and exposes (remotely) stop, playback ffwd, rwd, and status functions
- The data will be streamed from mobile to set top box
- We will assume both devices have access to the public internet

Sequence diagram analysis



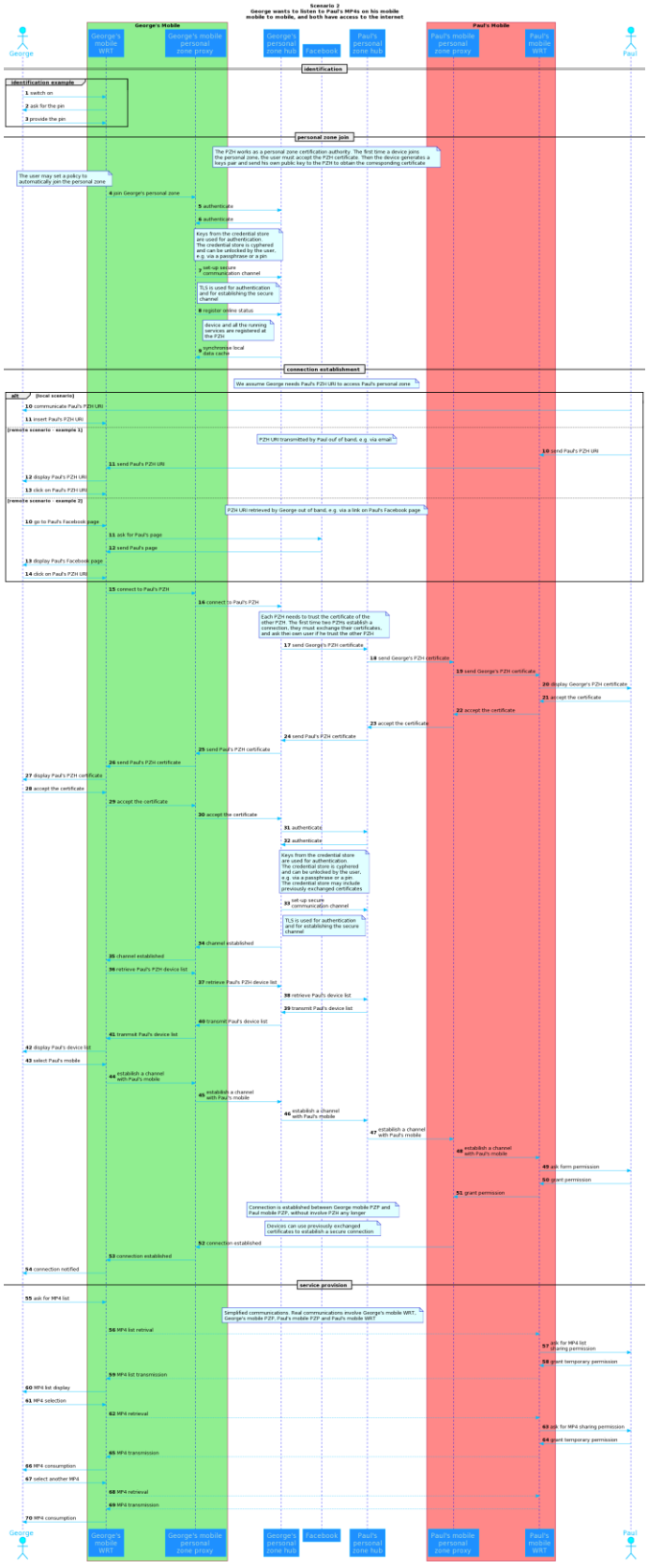
Scenario 2 - Across Multiple Personal Zones

George wants to listen to Pauls MP4s on his mobile - mobile to mobile, and both have access to the internet

Assumptions

- There is access to open identity servers on the internet
- This is a sharing scenario: permission must be granted for a resource to be consumed by another person
- This requires both users identity to be attested

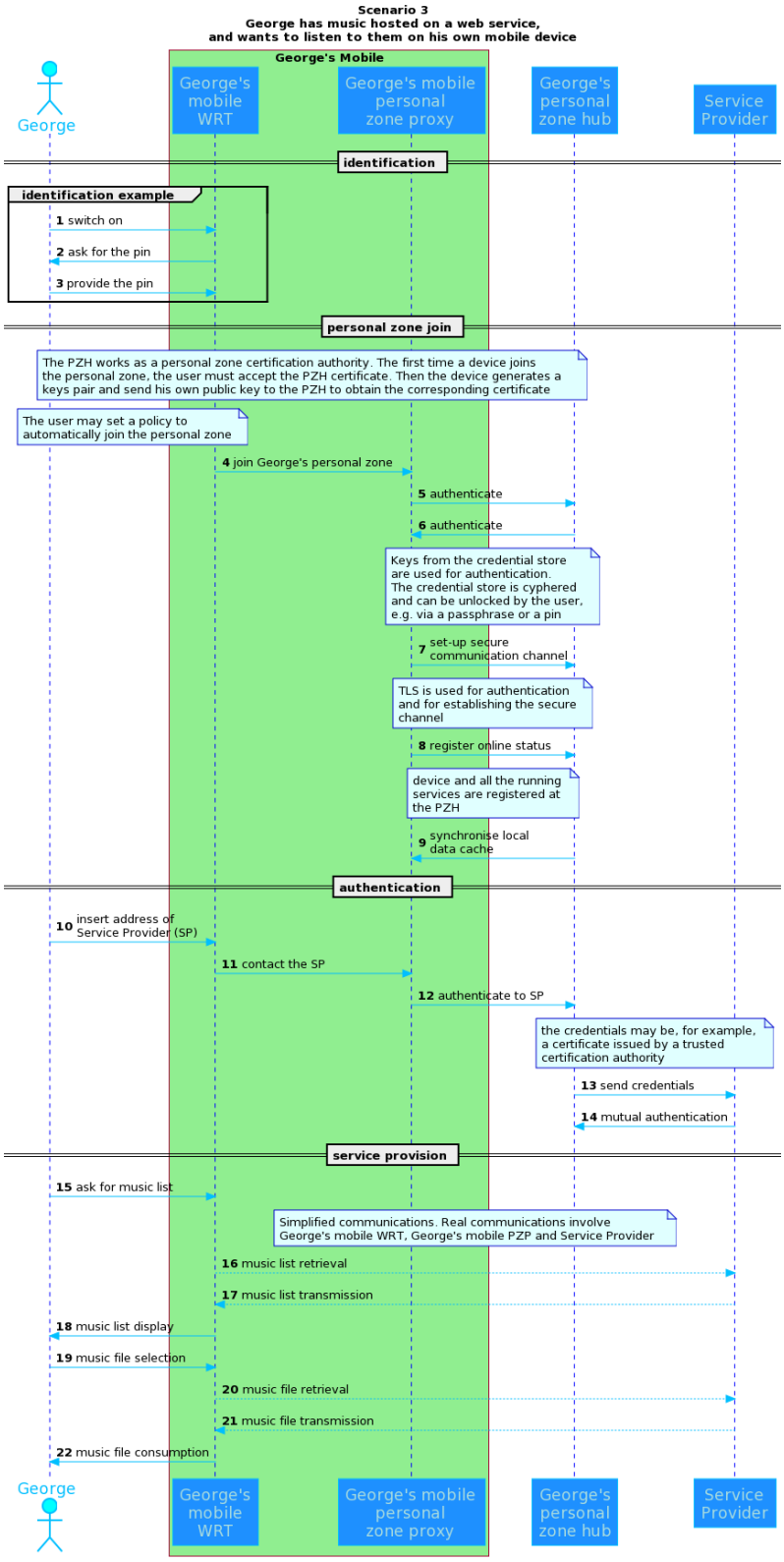
Sequence diagram analysis



Scenario 3 - Personal Zone and Service Communication

George has music hosted on a Web service, and wants to listen to them on his own mobile device

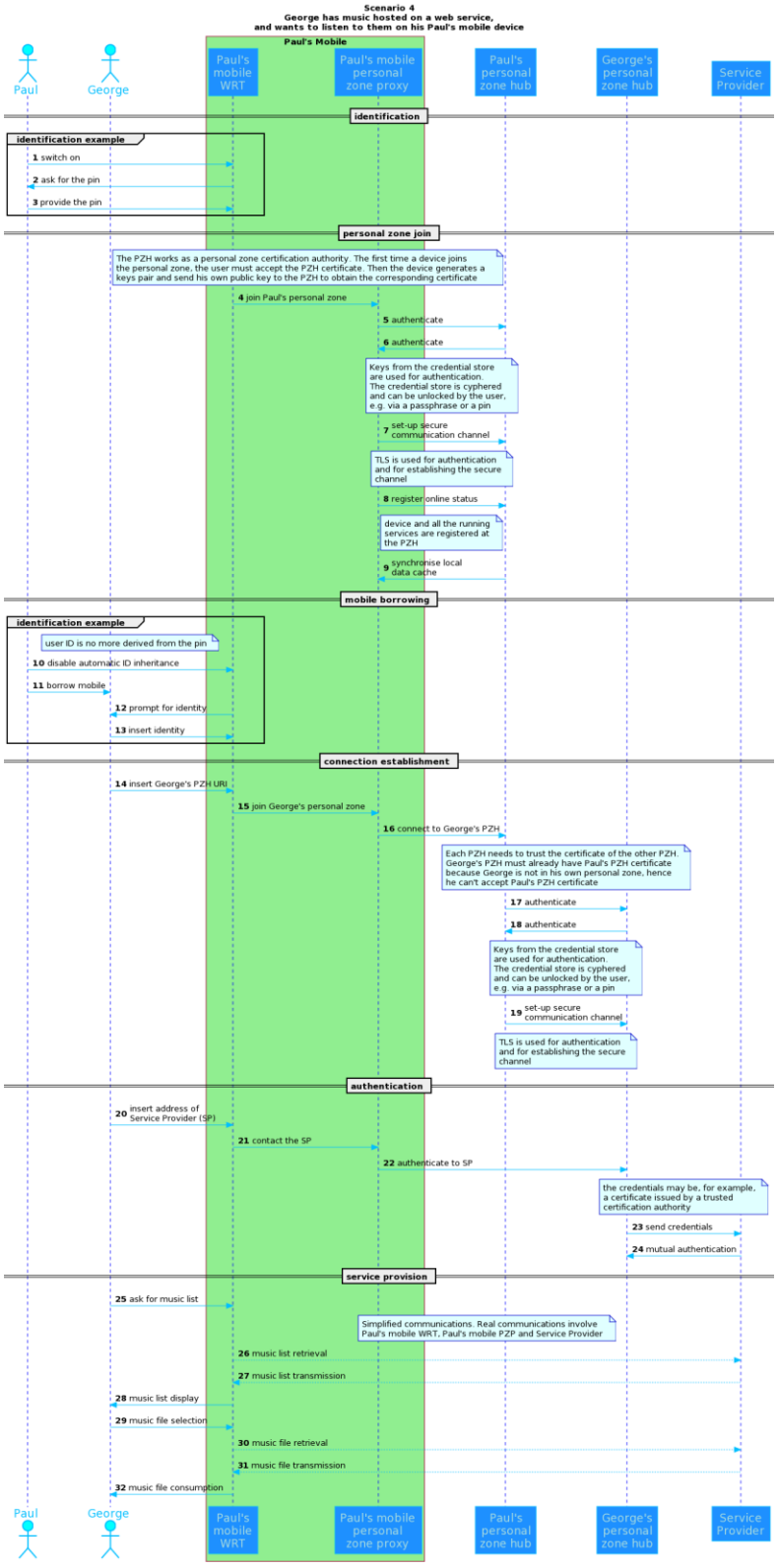
Sequence diagram analysis



Scenario 4 - Service Used by Multiple Zones

George has music hosted on a Web service, and wants to listen to it on Paul's mobile device

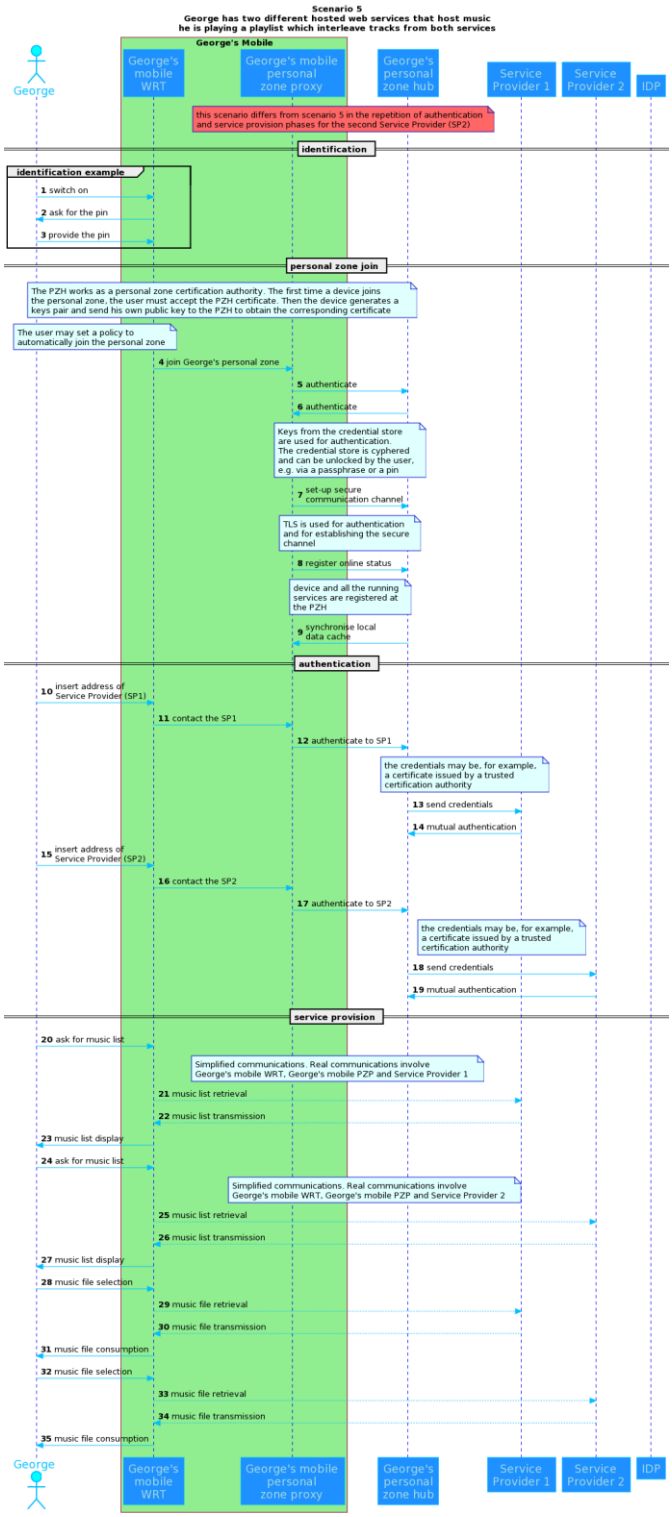
Sequence diagram analysis



Scenario 5 - Multiple Service Usage

George has two different hosted Web services that host music - he is playing a playlist which interleave tracks from both services

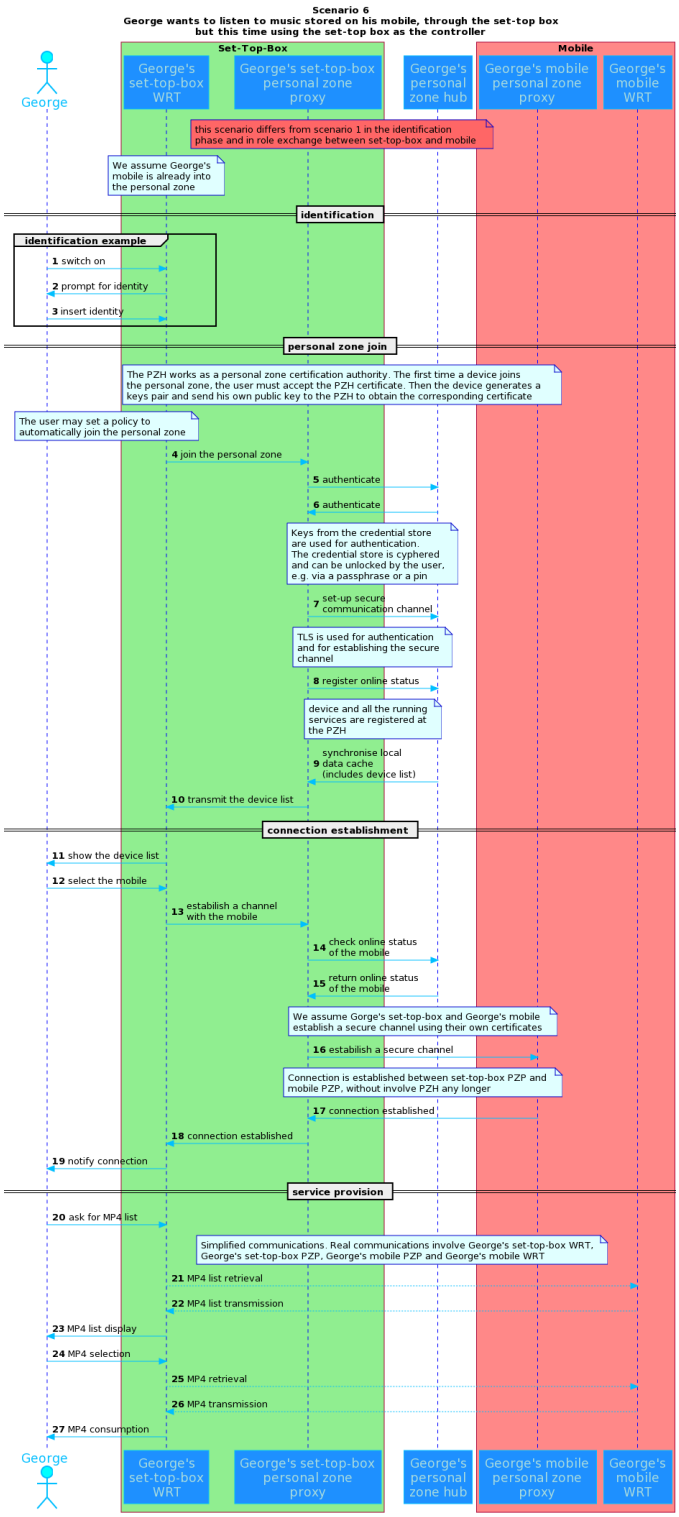
Sequence diagram analysis



Scenario 6 - Same Personal Zone

George wants to listen to music stored on his mobile, through the set-top box - but this time using the set-top box as the controller

Sequence diagram analysis



User Identifiers

Specification of friendly identifiers

For the sake of webinos identification, there is a unequivocal relation between the PZH URI and the user who owns the Personal Zone. Not necessarily this device have to be used in all device exchanges, E.g. when anonymity is required, an anonymous identifier could be used. (Anonymous authentication is a webinos phase II issue).

Specification of user tokens

X.509 certificates are a typical format of credential tokens. However, webinos does not restrict to this specific format. X.509 certificates are used in a TLS based session to ensure security properties. This is why the webinos specification currently uses X.509 only.

User authentication levels

The API by which the status of the user's authentication level can be determined and by which the user can be (re-)authenticated is specified in deliverable D3.2 in the JavaScript APIs section.

Inherited device authentication levels

Applications inherit at most device authentication level, according to least privilege principle. However, when further privilege is required, due to a sensitive function (e.g. access to highly sensitive data), applications can perform a re-authentication (possibly, with a stronger authentication process, e.g. two-factor authentication instead of one-factor).

webinos plugin authentication

Before installation, each webinos component ensures integrity and authenticity thanks to an electronic signature.

Session establishment

Typically, for performance and usability choice, authentication procedure is valid for an entire session. In some circumstances (e.g. access to the critical part of the session) an authentication process could be performed again, to mitigate session hijacking problems.

Session authentication leverages on SSL/TLS security protocols.

User Authentication and Secure Storage

It is up to the developer of the device how to identify and authenticate the user. Any webinos-enabled device **MUST** meet the security requirements which are discussed in the User Authentication Levels section above. It is the device manufacturer's obligation to ensure that user authentication cannot be bypassed.

The device **MUST** implement the Authentication API, which allows the PZP and any Web application to query the level of user authentication on the device, to determine if the user is currently authenticated and to explicitly advise the device to (re-)authenticate the user. This function **MUST** be accessible without any restrictions by the PZP. This function **MUST** be accessible by any application with prior permission check by the policy. This function **SHOULD** be implemented in native code for access by the PZP and a wrapper **SHOULD** be implemented in the WRT for access by the applications via JavaScript.

User authentication **MUST** be implemented in such a way that the secure storage (which contains the private key of the device, among others) can only be accessed by the PZP and only if the user is authenticated successfully. Once the user is no longer authenticated (e.g. by putting the device in stand-by mode or after some time of inactivity of the user), the user **MUST NOT** be considered as authenticated any longer and access to the secure storage **MUST** be blocked. Protection of the secure storage **SHALL** be a combination of tamper-resistant hardware and encryption. The content of the secure storage **SHALL** be encrypted by using credentials of the user as (part of) the decryption key. These credentials **SHALL** only be known to the device while the user is authenticating. Without user authentication there **MUST** be no way to decrypt the content of the secure storage.

Personal Zone Proxy (PZP)

The PZP is the component which implements most of the authentication functionality in webinos.

Once the user is authenticated successfully, the PZP is triggered by the device's user authentication module to perform further authentication. The PZP is triggered by the authentication module in a way that applications cannot do. The PZP **MUST NOT** be triggered by a message on the network stack, as this message could also originate from an application. Means of IPC are to be used which may vary from device to device and from operating system to operating system. In a Linux environment, PAM is suitable.

Next, the PZP tries to access the PZH over the network. If the PZP reaches the PZH it authenticates and sets-up a secure communication channel for any further communication. The PZP-PZH communication **MUST** use TLS, **MUST** use mutual authentication, and **MUST** perform any communication using the secured TLS-based communication channel. TLS version 1.0 or newer **MUST** be used. During the handshake phase of the TLS protocol, the PZP **MUST** send its certificate which was issued by the PZH of the same zone. The PZP **MUST** verify the certificate which is sent by the PZH during handshake. The certificate of the PZH must be the self-signed certificate of the PZH which is the root certificate of the personal zone. If the PZP cannot validate the certificate or if the certificate is not the expected one, communication **MUST** be aborted. The PZP **SHOULD** fall back to initial installation mode, where all the involved certificates are exchanged between PZP and PZH. As soon as the communication channel is estabilised with TLS, all communication is confidential and integrity of each message is validated. This is done by the TLS implementation. The PZP **MUST** not negotiate with the PZP and disabling of encryption or integrity validation. Once the TLS-based communication channel is set-up, it is kept alive for any further use until the device is turned off, turned into stand-by mode or the user is logged-off.

The PZP then registers its online status with the PZH. For doing so, the PZP sends a status update message through the established communication channel.

An overview of the format and content of the PZH-PZP communication is to be found at http://dev.webinos.org/redmine/projects/wp3-1/wiki/webinos_key_architectural_components.

The PZP then synchronises its local data cache with the PZH by using the synchronisation algorithm and protocol specified in the Synchronisation section.

Thereafter, the established communication channel is ready for use by any application which wishes to communicate with the PZH or which needs to route traffic through the PZH.

If the PZH is not reachable on the network, the PZP **SHOULD** try to access the PZH each time when there are changes on the networking. If another network bearer is used, or if the network address changes, or if routing changes, the PZP **SHOULD** try to access the PZH. If there are no changes on the network, the PZP **MAY** periodically try to access the PZH. It is up to the device manufacturer to decide the time interval. In general, it is intended to be connected to the PZH as frequent as possible, in order to be reachable via the PZH and in order to keep the data cache synchronised. The time interval **SHOULD** be balanced between this requirement and power consumption of the device.

If the PZH is not accessible, the PZP is accessible to other PZPs on the local network. The PZP **MUST** respond to other PZPs who intend to establish communication. The PZP **MUST** only accept TLS connection with mutual authentication, as described above. Any communication between PZPs **MUST** use the secure TLS-based communication channel. The PZP must consult its policy to determine which kind of communication is permitted via this communication channel.

In any setting, whenever an application of the device intends to establish any kind of communication with any remote application or service, the PZP **MUST** intercept this establishment attempt and **MUST** establish the secure communication channel. Once the secure channel is established, the application can continue establishing its communication. The PZP **MUST** ensure that all the traffic which is generated by the application/service is exclusively routed through the secure communication channel. It is up to the device manufacturer to decide for how long the established secure channel is to be kept alive. It is also up to the device manufacturer to decide which events will cause an immediate termination of the secure channel. These events **MAY** be: switching off the device, turning on stand-by mode of the device, logging off the user.

Personal Zone Hub (PZH)

The PZH is the component in the personal zone which organises trust relationships, issues and maintains certificates, establishes communication channels to other devices and performs authentication on behalf of the user.

During the installation process, the PZH has issued a certificate for the PZP. Whenever the PZP is about to establish a communication channel with the PZH, the PZH validates the certificate which is sent by the PZP. The PZP **MUST** validate the certificate and it **MUST** verify that it is one of the certificates which was issued by the same PZH. If the certificate was issued by the same PZH, the PZH establishes a TLS

connection with mutual authentication. During this authentication process, the PZH **MUST** send its own self-signed root certificate to the communication peer. If the PZP sends a certificate which is issued by another PZH, the PZH checks if the certificate is stored in the trusted users cache. If it is stored there, the certificate is considered trusted and a TLS connection with this PZP **MUST** be established. If the certification chain cannot be validated, the PZH **MUST** not continue communication with the PZP. The PZH **MAY** enter installation mode or the PZH **MAY** enter mode in which new certificates are introduced to the PZH and stored in the trusted user cache. Both these modes **MUST NOT** be entered and **MUST NOT** be processed without explicit user consent.

The PZH functionality is summarised in the webinos component overview http://dev.webinos.org/redmine/projects/wp3-1/wiki/webinos_key_architectural_components

Trusting New Users

Establishing trust to another user in webinos means that the PZH stores the certificate of the other user's PZH in its trusted users cache. This is described above. To better distinguish the two involved PZHs, the PZH of the own personal zone is called *own PZH* and the PZH of the other user's personal zone is called *foreign PZH*. The process is as follows:

1. Obtain the URI of the foreign PZH. This can be done by asking the user and typing it, by receiving it by any electronic means (e.g. e-mail) or by using Webfinger.

2. Initiate a TLS connection with the foreign PZH. Both PZHs determine that they do not trust the certificates which are sent by the peer PZH. Thus initiating the TLS connection is aborted. Both PZHs notify their users that there is a communication attempt. Both PZHs also display the certificate of the peer PZH to the user. The user is prompted for action.

3. The user **MUST** verify the validity of the certificate. This cannot be done automatically, as there might be a DNS spoofing attack or a middle-person attack which would not be detected by automatic means.

    1. Each user reads out the fingerprint of the certificate of the own PZH.

    2. The other user compares this fingerprint with the one which is displayed on their PZH. This is the one of the foreign PZH. The users do this via a separate channel, e.g. by telephone or in person, where both users also are convinced that they really talk to the expected person.

    3. Both users accept the certificate of the peer PZH if the fingerprint is valid and if they intend to communicate with this user. In any other case, the user declines the certificate.

4. If the user accepts the certificate, the certificate is stored in the trusted users cache. Since this is a security critical step, the user has to authenticate again in order to perform this step. The PZH **MUST** require user authentication before storing the certificate in the trusted users cache. The PZP **MUST** only store the certificate if the user has authenticated successfully.

5. The own PZP again tries to establish a TLS connection. THis time, both PZHs know the peer's certificate. They can mutually authenticate and thus they do establish the TLS connection.

6. Now that the TLS connection is established, APIs of devices of the foreign zone can be accessed. The PZP of each of the devices in a zone takes care of access control. The policy defines which resources and APIs the newly trusted user is permitted to access.

The PZH is not required to have a user interface. All the process can be performed by one of the PZPs in the zone. Once the PZP has added the certificate to the trusted users cache, it **MUST** synchronise the cache with the own PZH.

Note that the policy **MAY** define that the user is not to be prompted for action in step 2 above. In this case, the PZH **MUST** terminate communication with the foreign PZH without involving the user. However, if the users intend to establish the trust relationship, they must disable this rule temporarily.

Dependencies on other components

PZP accomplish authentication process on the base of local Policy Infrastructure, in particular interacting with the decision wrapper described in the policy architecture (PZP has also a number of other functions, as described in Security section)

Implementation Architecture

The basic authentication architecture deals with

- Electronic signature signature (according to W3C widget signature standard) and correlated cryptographic primitives
- management of X509 certificate type
- use of TLS/SSL protocol and correlated ciphersuites
- XMPP for PZP-PZP and PZP-PZH communication

webinos architecture do not restrict to these security means, but implies them as a baseline

Discovery

Webinos discovery defines interfaces to detect devices and services either through remote network access or via short range connections (e.g Bluetooth, WiFi) or local bearers such as USB. It is responsible for:

- Advertising or publishing device/service's availability and capability
- Querying or finding device/service based on certain criteria
- Binding with device/services that meets the searching requirement
- Discovering PZH

In remote scenario, discovery aspect happens after connecting with Personal Zone Hub (PZH). Typical steps in remote discovery:

- Publish presence information to PZH along with services and publish/subscribe information.
- Gather service information based on userid/jabberid
- Establish connection with other nodes

Facing the challenges of a wide variety of discovery protocols used these days due to multiplicity of networking technology, Webinos discovery proposes a solution by providing two levels of APIs:

- Low level APIs based on specific interconnecting technologies for 3rd party developers
- High level generic APIs using JavaScript libraries for Web developers.

Technical Use Cases

Assumptions:

- No particular device is considered, it could be mobile or TV or automobile or PC
- There could be more than one owner for the same device (e.g. TV)
- Connectivity happens seamlessly for both local and remote discovery. In case of remote device connection to local device which does not run on IP, PZP will act as a bridging point.
- Policy management and security mechanism apply while connecting and in some cases after connecting.
- Local discovery applies for finding USB, BT, WiFi, UPnP and ZeroConf devices

- In technical scenario, Personal Hub could be XMPP server. XMPP is used for connecting to server and gathering information about connected devices. It does not support discovery directly but provides information about connected devices.
- All communication happen over IP for remote discovery, IP addresses are assigned through DHCP, if not they are assigned based on link local address. In local discovery it could be IP based or non IP based for short range bearer devices.
- PZH is assumed to be in cloud and PZP will act in case of PZH missing, as it has cached information about the devices.

Use Case 1:

George wants to access his own device located remotely on public IP.

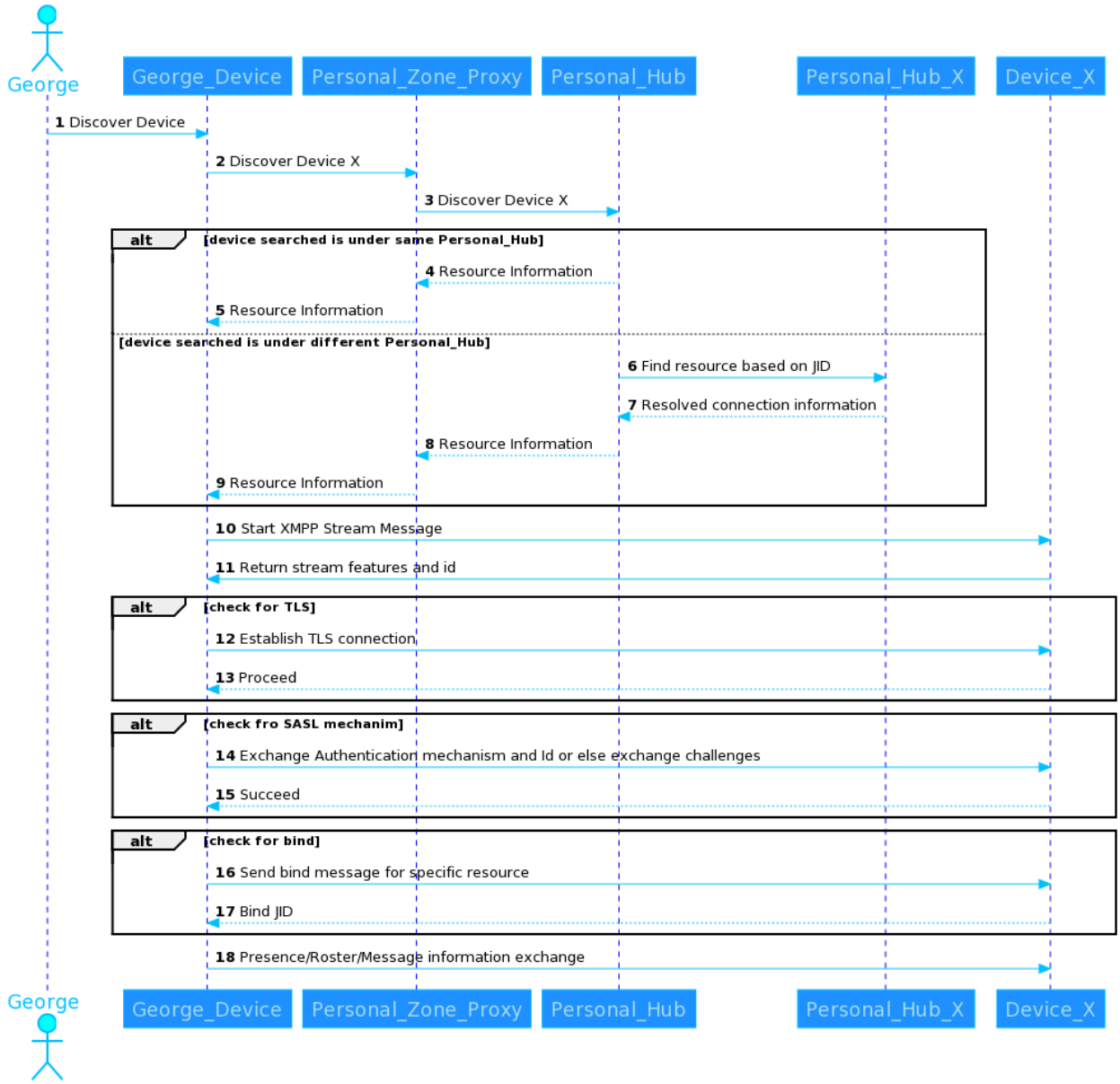
Description:

George is travelling and wants to access his home TV or set top box to access the content on the device. Based on the selection of device, a connection is established and real time communication takes place between devices. This would involve establishing transport layer security, exchanging user credentials with personal hub. If personal hub indicates device is online, connection is established between the devices. Identification of device happens through JID (Jabber ID) used in XMPP protocol. Usage of protocol is described in Protocol Description

Main challenges:

- Find own device and establish secure communication over device.
- Know about the presence of the device and if it is available to be used.

Sequence Diagram - Use Case 1



Use Case 2:

George wants to access device located locally using link local address.

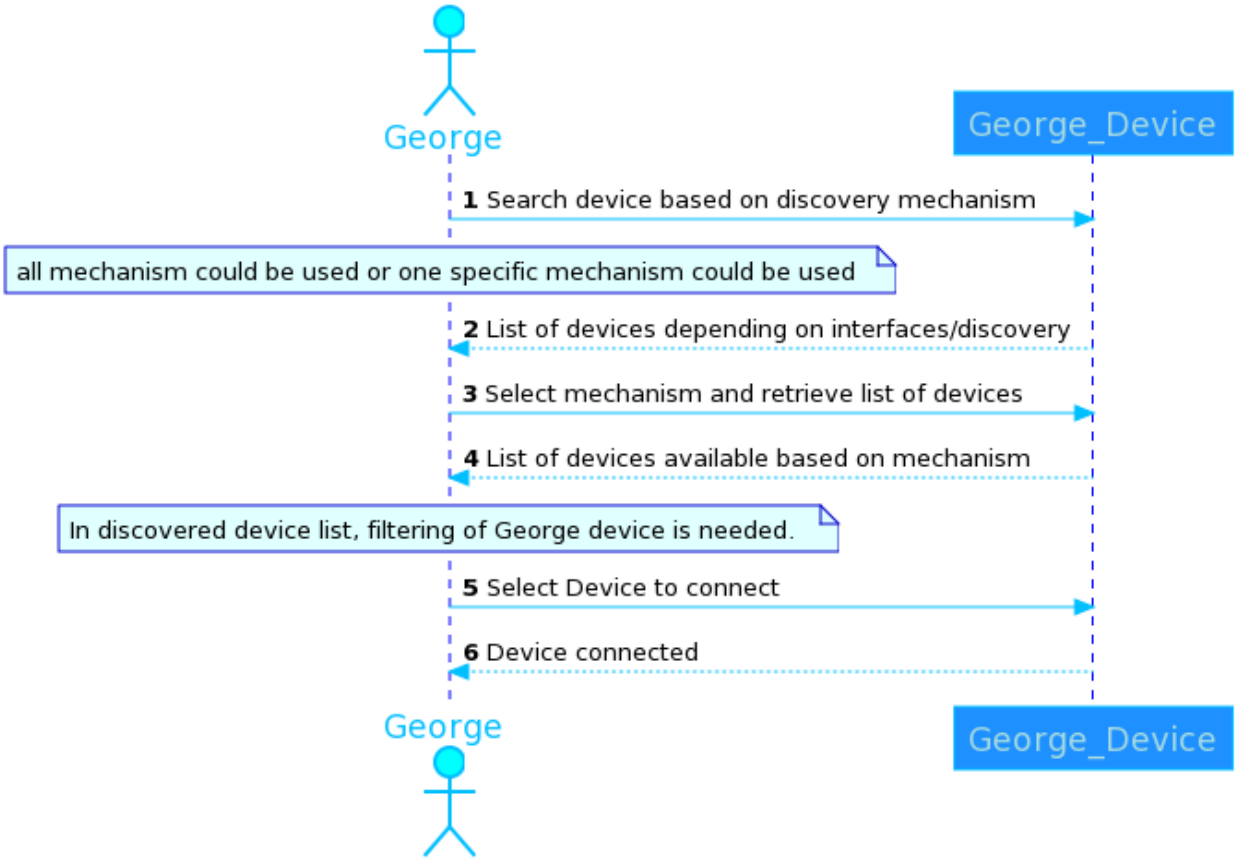
Description:

George is at home and wants to access TV from his mobile phone. User has to first select the discovery mechanism he wants to use; he could either use all or use a specific device to find other devices.

Main challenge:

- Identify device from set of device and authenticating to access device
- If device is being shared, what policy rules apply regarding access? Which user owns the device?

Sequence diagram - Use Case 2



Use Case 3:

Paul wants to access his friend George's device and connectivity is using public IP.

Description:

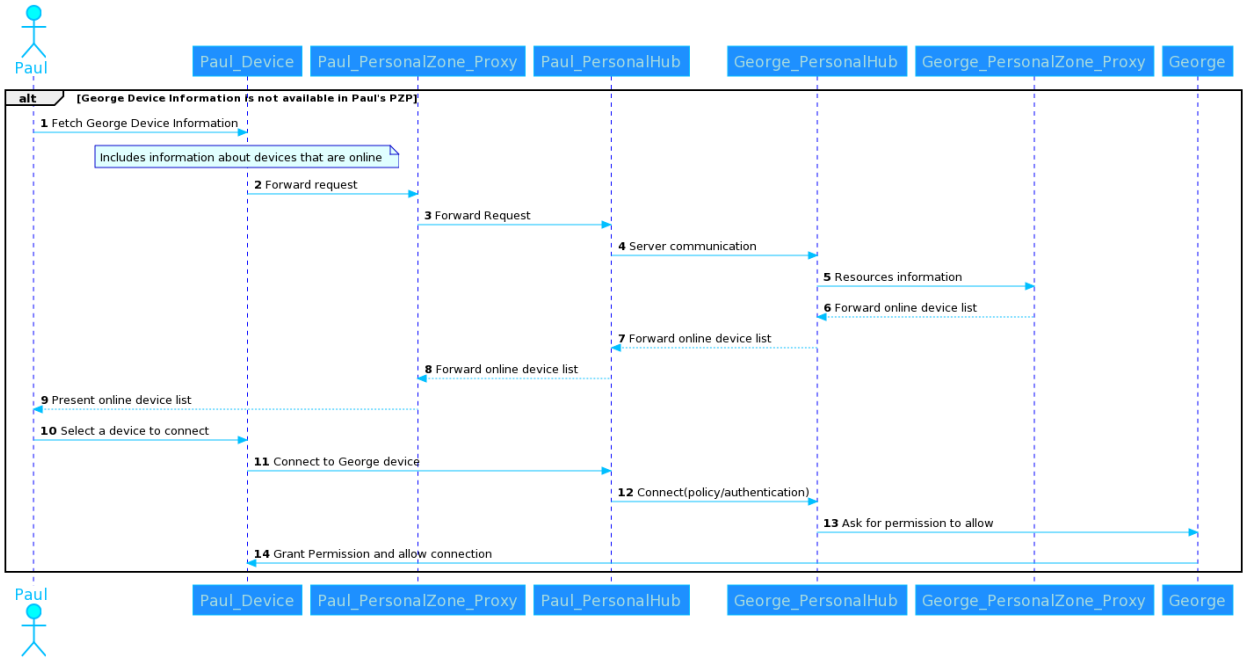
Paul has his friend's id which in XMPP terms would jabberid@domain and wants to connect with a particular device. Information about the George's device that are online has to be first retrieved if Pauls's device does not have information and then connection is established based on George's permission to access device functionality. To support connection it would involve communication between PZH and PZP. It is assumed in this Use Case, PZP in Paul Device itself does not have George's PZP availability.

Here it is assumed that PZP holds information about local devices such as USB and BT. Consider PZP to be MP, USB and BT are local but having this information is not available to PZH directly, it is not possible since they are not IP based and cannot be updated about their availability. PZP will act as a bridge to update information. In the example, when PZP is coming online, it includes information about local devices to it. If there are other BT device all in near proximity it is PZP that updates PZH about them. So when device comes online, it is through PZH, PZP will be able to obtain information.

Main Challenge

1. Policy rule to allow what features to be accessible to other user devices
2. Authentication before connecting two devices

Sequence Diagram - Use Case 3



Use Case 4:

Paul wants to access George device using local link local address or local address assigned using DHCP

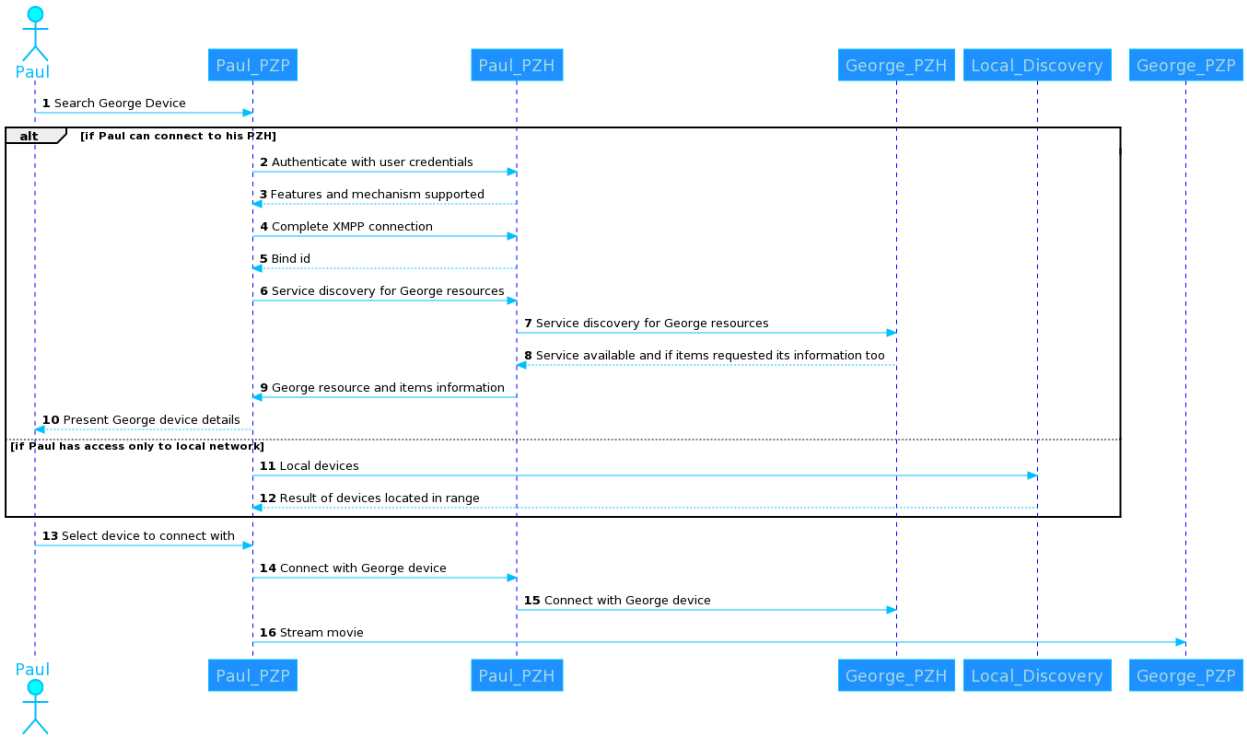
Description

Paul is visiting his friend's house and wants to show movie on his phone on George TV. In order to do that Paul has to first authenticate with George's personal hub, once authenticated Paul should be able to establish connection with George's TV based on policies of George.

Main Challenge

1. Policy setting and at the same time ease in using setting to allow and stop communication with friend device
2. Paul if he is a family member and wants to access George device, how to handle access all the time
3. In absence of personal hub, how to impose policies and authentication

Sequence Diagram - Use Case 4



Formal Specification

This section covers both high level and low level API for Web developer and third party interface developer respectively.

Protocol definitions

This section highlights protocol, data structure that third party developer could use to implement webinos discovery module.

This section provides outline specifications for device and service discovery module. Device discovery aspect is not limited to just one protocol but provides support for discovery on more than one interface and allows usage of more than one discovery mechanism.

Discovery module aims to provide wrapper interface for high level API and low level discovery specific API. It will provide a uniform interface for Web developers to obtain addressing information on devices and services across a wide range of interconnecting technologies. The discovery module shall provide the following functionalities:

- Advertise or publish device/service's availability and capability
- Query or find device/service based on certain criteria
- Bind with device/services that meet the searching requirement

To describe above functionalities below section covers data structure and functional methods for discovery mechanism. All aspects covered below differ for local and remote discovery mechanism.

Note: Webinos will provide information about interfaces based on BT and WLAN. Discovering WiFi/BT devices and connecting to such devices is outside scope of webinos and is underlying OS functionality. Assigning IP address after connection establishment is also underlying OS functionality. Though webinos can provide these functionalities but it will be a replication of functionality.

Advertising/Publishing Information

Devices advertise their services and allow other devices to discover device's service. Each discovery mechanism has different way of advertising services and allow communication establishment with the device. Mechanism of advertising differs in local discovery and remote discovery.

In local discovery mechanism, service availability/publishing has to be multicast and discovery happens by looking for a service and establishing communication with the device based on the service details. In remote discovery, connection is established with PZH and then information is exchanged to discover other devices.

Local Discovery

The interface that will be provided to search local device in webinos will be USB. WiFi and BT device discovery as mentioned above will be discovered and connected through underlying OS. Support to retrieve information about connection to such interfaces will be supported.

Reason for including USB is to find all devices that are connected to a device to be searchable and allow remote device to connect to the device, for usage through USB over IP.

All interfaces are to be handled as resource and should be separately addressable for remote device to discover it.

Device needs to advertise about the service it offers using ZeroConf or UPnP. This advertisement provides information about capabilities of the device and the information to establish communication with the device.

```
#define MAC_LEN 6

// Discovery structure searches for device based on interface and discovery mechanism
// Note it is not needed to use all mechanism all the time, it is user configurable option.
struct discovery{

  // List of USB connected interfaces
  USB *usb;

    // Holds list of printers found through SLP
  SLP *slp;

   // Devices published/advertised information as well as device found through ZeroConf resolve method
  ZeroConf *zero_conf;

  // UPnP device discovered list
  UPNP *upnp;

  // DLNA discovered device list
  DLNA *dlna;
};

// UPNP based devices found are hold in this structure
typedef struct _UPNP{

  // Unique id of the device
  char *uuid;

  // Device type along with manufacturer details
  char *device_type;

  // Address of device
  char *address;

  // Port on which service is available
  int port;
```

```
 37
 38   // Description of service
 39   char *description;
 40
 41   // Holds another UPnP discovered device
 42   struct _UPNP *next;
 43 }UPNP;
 44
 45 // Discover printers based on SLP
 46 typedef struct _SLP{
 47
 48   // Service type
 49   char *srvtype;
 50
 51   // Service URL
 52   char *srvurl;
 53
 54   // TO hold other SLP discovered devices
 55   struct _SLP *next;
 56 }SLP;
 57
 58 // Structure to hold ZeroConf information
 59 typedef struct _ZeroConf{
 60
 61    // Publish device details, this is first step to be done since devices need to
start service before publishing service
 62   ZeroConf_Publish *publish;
 63
 64   // Discovered ZeroConf based device details
 65   ZeroConf_Resolve *resolve;
 66 }ZeroConf;
 67
 68   // This structure holds information that will be used for publishing service for
webinos based device
 69 typedef struct _ZeroConf_Publish{
 70
 71  // Name of the device such as hostname@machinename
 72  char *name;
 73
 74  // Port on which service will be available
 75  int port;
 76
 77  // Advertised service type such as _presence._tcp
 78  char *service_type;
 79
 80  // TXT in form of pair (type=value)
 81  char **txt;
 82 }ZeroConf_Publish;
 83
 84 // This structure holds information of ZeroConf device available in the network
 85 typedef struct _ZeroConf_Resolve{
 86
 87   // user assigned name or default name
 88   char *name;
 89
 90   // service name of the type we are connecting to
 91   char *type;
 92
```

```
 93   // human friendly service description
 94   char *nice;
 95
 96   // numeric network interface index
 97   char *iface;
 98
 99   // true => IPv6, false => IPv4
100   bool ipv6;
101
102   // IP address
103   char *address;
104
105   // Port on which service is running
106   int  port;
107
108   // additional name/value pairs
109   char *txt;
110
111   // Holds next ZeroConf Resolve data;
112   struct _ZeroConf_Resolve *next;
113
114 }ZeroConf_Resolve;
115
116   // This struct will hold information about searched USB devices
117 typedef struct _USB{
118   // Unique product number of USB device
119   unsigned long product_number;
120
121   // Vendor/Manufacturer id
122   unsigned long vendor_id;
123
124   //USB standard defined class of device
125   unsigned long device_class;
126
127  // Each device class is divided into multiple device sub class
128   unsigned long device_sub_class;
129
130   // Device number
131   unsigned long device_num;
132
133   // Bus on which device is running
134   unsigned long bus_num;
135
136   // Text description of the vendor name
137   char *vendor_name;
138
139   // Hold details of next USB device
140   struct _USB *next;
141 }USB;
142
```

Devices searched are presented to the user and once user selects device to connect with, a bind operation is performed on the device.

Bind

Bind operation differs on if action is local or remote. In case of remote it will involve communication iwht PZH.

Local Device Bind

Each interface defines different bind mechanism. If binding is based on interface, device will be connected to selected interface. In typical socket communication it would be based getting information based on getaddrinfo/getifaddrd. Once information is retrieved, a socket connection is established.

```
 1 struct device_bind
 2 {
 3   // protocol type
 4   unsigned char protocol;
 5
 6   // interface
 7   unsigned char iface;
 8
 9   // host name of the device
10   char *hostname;
11
12   // IPv6 address
13   unsigned char ipv6[16];
14
15   // IPv4 address
16   unsigned char ipv4[14];
17
18   // port number to connect or the port that will be used if acting as server. This
could be user input or random input
19   int port;
20
21   // MAC addr of the interface, if USB product:vendorid, rest filled with Zero
22   char mac_addr[MAC_LEN];
23 }LocalConn;
24
25 // 1 if successful or if any error depends on iface and socket error
26
27 int connect_local(LocalConn *local);
```

Remote Discovery

Remote discovery happens after connecting with PZH. The PZH is resolved based on a domain name, it involves using of DNS-SRV resolution mechanism, to fetch address information and establish socket communication with PZH. The information needed from user is userid along with domain name, resource (optional), and password. Based on this information connection will be established with PZH. User can discover other users and their devices and establish communication with the resources.

There are several mechanisms that supports remote discovery, it can be done through resource distributed as distributed hash tables or using XMPP. To communicate with remote device, it requires having IP address or domain information to establish socket communication with the device. First step is to gather address information.

XMPP is used for remote discovery in Phase 1, as it allows connecting with different resources, finding services, finding different items of his own and user friends. For example, userid alice@example.org/camera has alice as a userid, example.org is the domain that holds information about user alice and camera is the resource that user wants to connect. Using same id but with different resource, user can connect to XMPP server, alice@example.org/phone.

A typical XMPP node can hold three pieces of information:

- Resource/Nodes: Entity that can be addressed
- Services: Commands or function resource can use
- Items: These are nodes under a Resource.

To perform service discovery of user own devices or other user devices, XMPP Service Discovery (XEP-030) will be used. It defines protocol mechanism to retrieve information about info-nodes and items-node associated with a jid. Info-nodes are similar to full jid (alice@example.org/camera) and item-nodes are not addressable directly, the information about them is retrieved using info-nodes. Query tag of XMPP messages supports both items and info node. Message exchanged uses type get and result between requesting and sending entity respectively. See below diagram for full call flow.

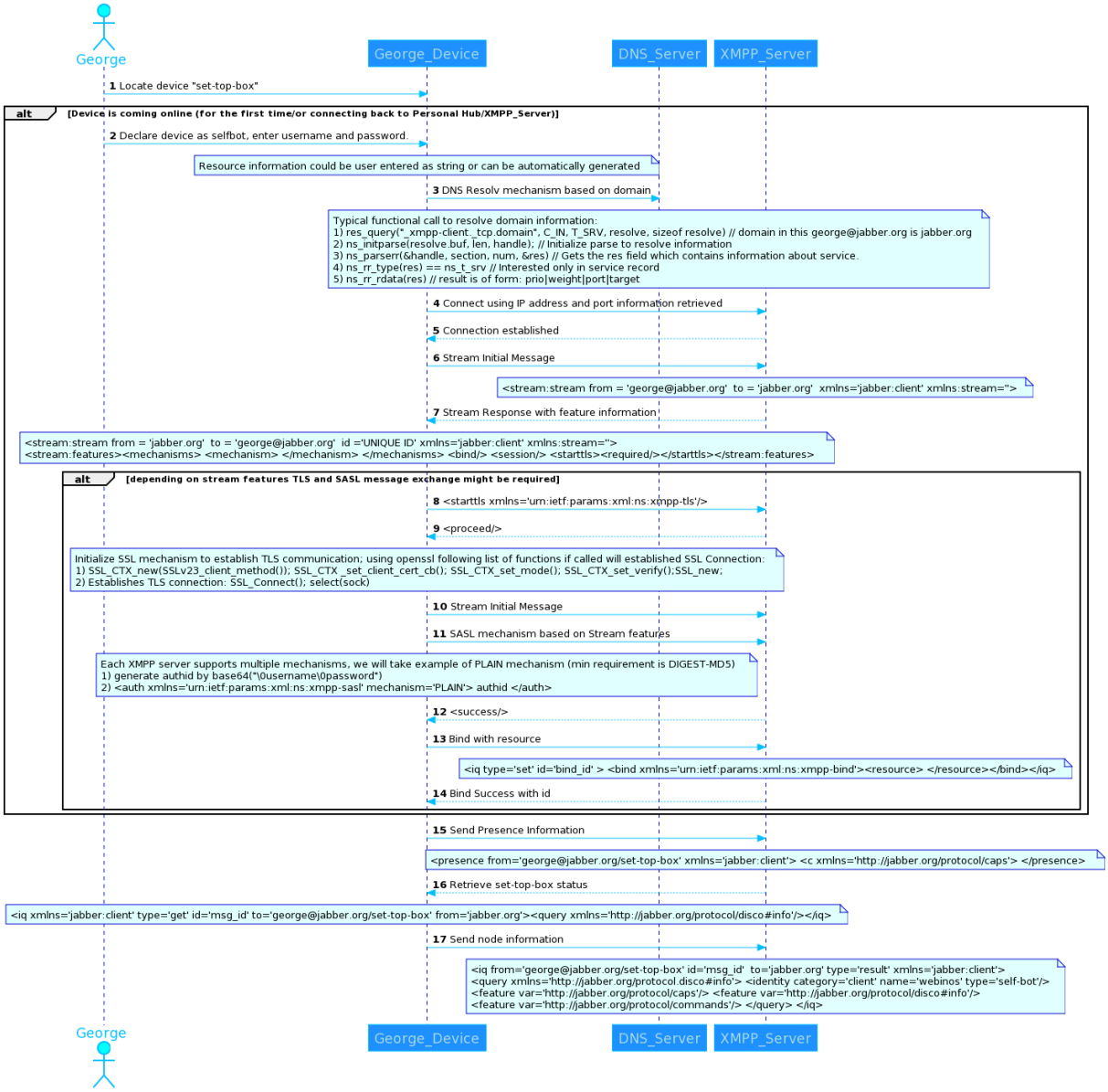
In a personal zone, not all devices will be IP capable. To handle this scenario a device can include all the local devices connected to it (e.g. bluetooth, usb) as items-node. Any device searching for George devices will first fetch information about info-nodes and then each info-nodes can then be queried to get all items-node information.

As part of discovery, services have to be discoverable. After XMPP connection has been established, it could fetch service information based on URI mechanism or using adhoc command functionality, where a particular node holds information about advertise service. Both mechanisms are supported in XMPP.

The bridging mechanism between local devices and remote devices will be a functionality provided by webinos's PZP. As each node cannot be directly communicable they will be first communicated to Personal Zone Proxy (PZP) and then PZP will act as a mediator between devices. The communication mechanism is dependent on how services are defined. This could be either based on adhoc commands or based on config.xml defining services.

Next step is finding information about devices whether they are online. User can add different user he knows in a roster (XMPP terminology for friend list) and can define different groups. Depending on presence preference, roster information will be updated once friend and his devices are online. All the information displayed will be based on policy management rules.

User should be able to get all the resources that belong to him indicated as devices in his personal zone. All devices of the user will be listed as resources/info-nodes. It should be guaranteed that each resource is unique. Once the user select device he wants to connect, depending on the preference connection will be established.



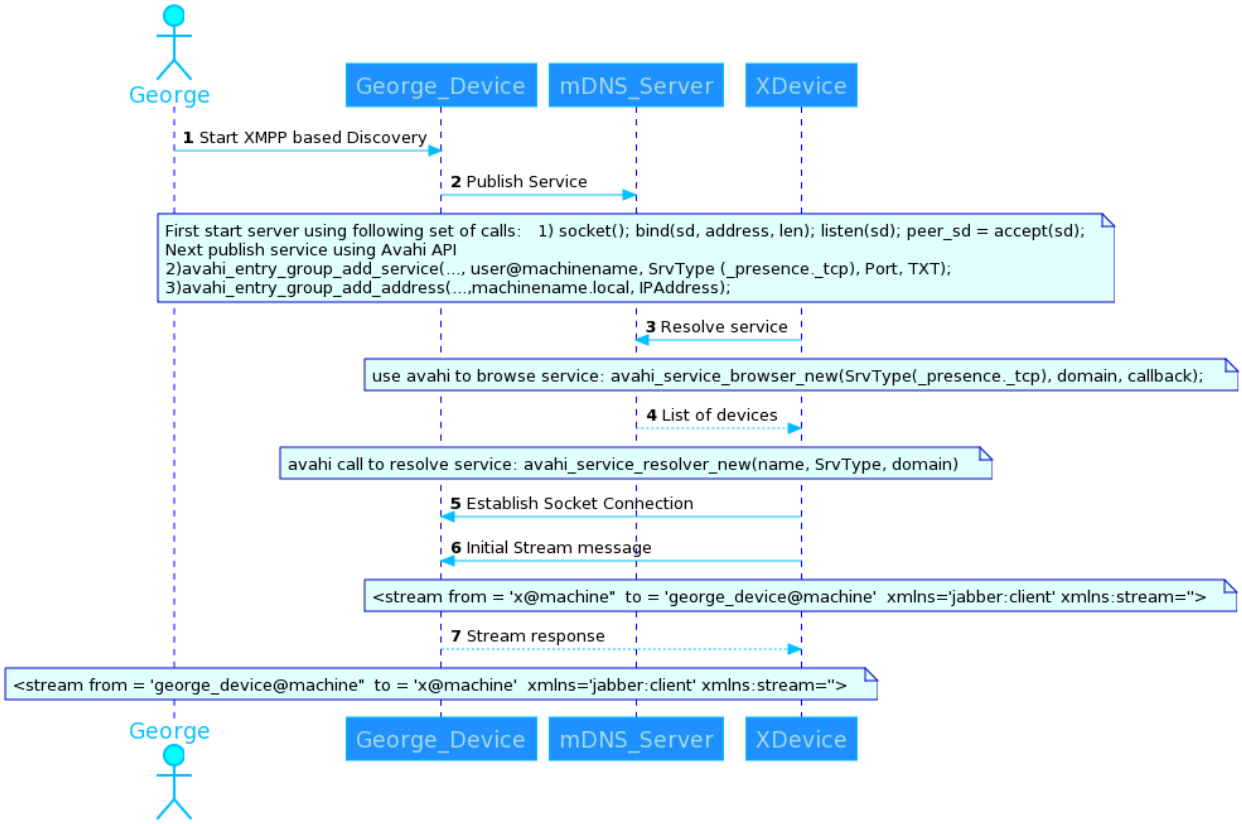
Other XMPP mechanism that is relevant for webinos is using it in serverless environment i.e. lack of PZH with no public IP connection possible. XMPP Serverless Messaging (XEP-0174) uses ZeroConf as underlying discovery mechanism. It provides information for two devices to establish socket connection, and once socket connection is established it uses XMPP streams and iq message to exchange messages between devices. Devices can extend stream message exchange by performing feature exchange and authenticate with each other before receiving messages.

XMPP defines ZeroConf service type "_presence._tcp" and requires information about specific port and IP address. All these information maps to A record, SRV and PTR field of DNS field. Any extra information can be filled in the TXT field of DNS which is in form of name=value, where first field needs to be "txtvers=1".

Publishing a service requires device to have server thread running in accept mode to accept connection from other devices using the port number advertised. A thread will handle new connections with each client. Message exchange will be of form xmpp message type, iq.

To use the service, device can resolve services advertised in form of "_presence._tcp". Once the device hosting this service is found, based on user preference, selection or previous decision connection is established between devices. The mDNS resolve contains enough information to establish socket communication with the device and start XMPP session.

A sample implementation overview is described below. It is one of the probable way of implementing communication with XMPP server.



In comparison to other service types available on ZeroConf, XMPP serverless provides service type, IP address and port that helps in establishing socket communication between two devices and then they can exchange messages based on XMPP. The advantage is no need for driver as two devices uses XMPP protocol to exchange messages.

Based on the above description, API and structure relevant to establish remote connection are presented below.

```
 1 typedef struct remote_conn{
 2
 3   // This will hold username in case of connection to XMPP server and nick in case
of connecting to XMPP
 4   // serverless messaging
 5   char *username_nick;
 6
 7   // Domain name where user is registered with and in XMPP serverless as a machine
name
 8   char* domain_machinename;
 9
```

```
10   // Password required for connecting with XMPP server
11   char* password;
12
13     // If same resource is to be used with multiple resources, all resource
information. This could be user input
14     // or information is filled in automatically based on discoverable device
configuration
15   char **resources;
16 }RemoteConn;
17
18 // Error handling based on XMPP error message
19 // This call has to be called in order to establish communication with remote XMPP
server and in case of Serverless
20 // Messaging this will be used to start services locally, advertise service, and
connect to remote device.
21
22 int connect (RemoteConn *remote);
```

Query Information

The data structures defined support interfaces and devices. Each interface and discovery mechanism can be queried to get device specific information. Results might vary depending on device and mechanism used to find the device, for example USB discovery mechanism provides different information and if queried for device found through UPnP or ZeroConf information provided is different.

```
1 // Returns 1 if it meets filter option or else result will be based on interface
results
2 // result will be based on format that developer can understand
3 int query (const char *device_interface, char *filter_option, char *result);
4
5 // Returns the IP address depending on the interface
6 char* getIPAddress(char *iface);
```

JavaScript APIs

Webinos provides a JavaScript Discovery API for Web developers. The API provides functionality to search for services either remotely/locally and to instantiate an implementation of the API that hides the complexity of communicating with the remote service in a trusted manner. The API can be used to find service that are defined by webinos but it is also possible for application developers to add own services and make those services discoverable. A service is made discoverable by an application developer using the webinos:shared element in the config.xml manifest as defined in Exposing Application Functionalities as Services.

The Discovery API is defined in Webinos Discovery specification. The Discovery Interface provides three methods that are used to find and create services:

- findServices(), to query for services
- createService(), to create a service with a given service identity
- getServiceId(), to get a unique identity of service that can be shared between two peers.

The search query is asynchronous and there will be a success callback for every service that is found. The time for finding services can vary significantly depending of which type of discovery method that is used, e.g. Bluetooth service discovery. An application must thereby be able to handle cases where the search query continues for a reasonable amount of time. The default max search query time is defined to 120 seconds.

The second part of the service discovery process is to Bind to the requested services. This method is implemented by the Service Interface as defined in Service Interface.

The Bind operation will mutually authenticate the application/service, verify access/privacy policies and instantiate an implementation of the API in asynchronous manner. Once the service binding has been successfully executed, the API will be ready to use by the application and the service object will act as a proxy to the remote peer. A more detailed example of how to use the API is available here discovery code example

Discovery of Personal Zone Hub

WebFinger will be used to discover the Personal Zone Hubs (PZH) from e-mail addresses, facebook identities or some other well-known identity. When a WebFinger request is issued with an user identity to a Webfinger service, the WebFinger service will return an Extensible Resource Description with known services associated with this user identity. The PZH will be exposed in the XRD document as a host meta data element using the Link element where the ref attribute is assigned to http://webinos.org/spec/1.0. Example of the XRD document with a link to a PZH:

```
1 <?xml version='1.0'?>
2 <XRD xmlns='http://docs.oasis-open.org/ns/xri/xrd-1.0'>
3                         <Link                         rel='http://webinos.org/spec/1.0'
ref='http://phz.example.org/profile/joe.smith'/>
4 </XRD>
```

Dependencies on other components

This section describes interfaces between discovery and following work areas:

- Event Handling
- ID Management

Event Handling

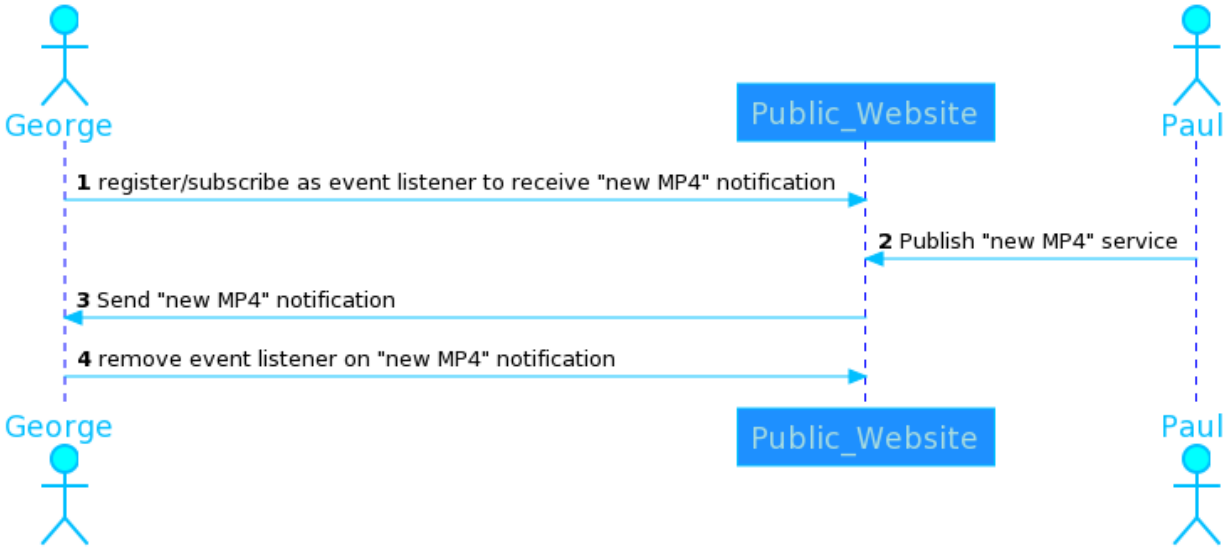
Interfacing with event handling will mainly happen when

- There are changes in status of service or device, or in contextual aspects of a service or device being used
- New service becomes available. The new service could be a brand new one or could also possibly be a better replacement for the existing service.

Discovery requires event handling to provide the following interfaces:

- Device or service requester to register as event listener subscription to a certain event type
- Device or service publisher to send event
- Device or service requester to remove event listener

To be able to receive a certain class of notifications on the changes of status or context information of the service or device being used or to be informed about new services, the service or device requester needs to act as an event listener. To do this, the requester can subscribe to the event source with associated queries. Event source varies on different use cases. For example, in the case that George would like to be notified of the update on Paul's MP4s, a possible implementation is to use publish/subscribe as follows:



In this case, the notification is pushed to George's Personal Zone Hub (PZH). George's PZH will work out whether George prefers this notification being forwarded to the device he is using or to all his devices. Alternatively, in the case that George's media player on mobile would like to know the updates of on the MP4 files on her PC, a Zone API shall be applied.

Some lower level discovery mechanisms have their own event handling mechanisms. For example, in UPnP, a control point can be registered at a service to be informed about device or service status changes. The service sends event messages to all registered control points after changing the value of a state variable. Event messages are transmitted via the Generic Event Notification Protocol (GENA) (COHEN11). In webinos, service advertisement and service status update function shall be exposed as JavaScript APIs or JavaScript Object. These need to be supported by a lower level mechanism. A wrapper on the top of lower level event handling mechanism might be required in this case, for example, exposing an HTTP or Web sockets or similar server as part of a plugin.

ID Management

To discovery another person's zone, in another word, to acquire proxy object of another person's personal zone, a userid (or Zone ID) or information from where userid can be acquired should be

provided. The userid information could be [i] email address; [ii] phone number; [iii] or just a text file etc. In whichever way, when userid is available, user should be able to locate or query service to determine their zone hub, further address devices based on their user information.

Due to the variation of interconnect mechanisms, device identity could be represented in different forms such as UUID in UPnP, MAC address in Bluetooth, some of them also presents friendly description, e.g. Mark's PC, at the same time. The Web developer, however, doesn't want to know mechanism specific names or identities. This requires a genuine Identity/naming identity representation. Some discovery mechanisms, e.g. mDNS, allow user assign meaningful names to devices to replace the default names. A common naming scheme sitting on the top of different bearer schemes, will allows us to hide the under layer implementation details.

Device names should only be unique to the user they are associated. A good example on presenting userid and deviceid is a full Jabber ID: alice@gmail.com/mobile, which is a representation being used in XMPP description.

Implementation Architecture

This section looks at the two levels of implementation APIs for different groups of discovery API users:

- Third party developers who would like to implement interfaces have APIs on the internetworking technologies they are interested
- Web developers who don't want to know the details of the bearers and use prefer to have generic structures for different discovery mechanisms.

Figure Implementation_1 & Figure Implementation_2 present the implementations architectures for both levels of APIs. For low level third party API, when native discovery APIs is not presented in JavaScript, a wrapper is required. Whilst for high level APIs, a wrapper is essential in order to isolate the complexity of different internetworking technologies.

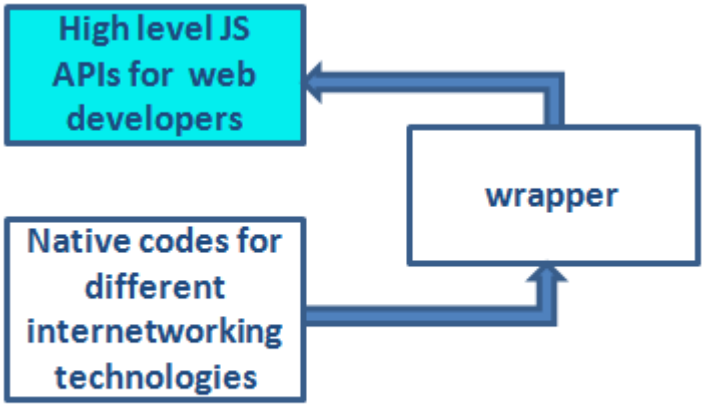


Figure 1: High Level API Implmentation Architecture

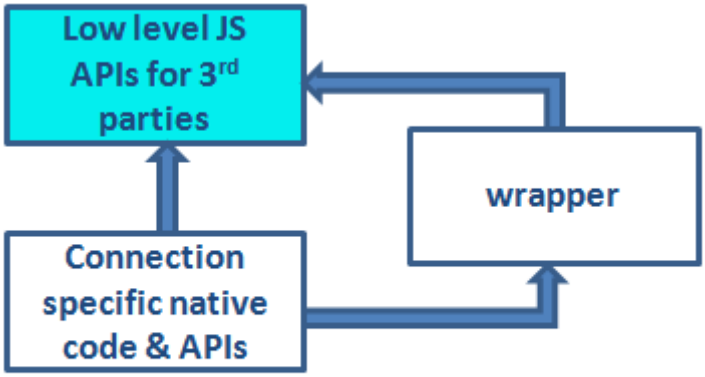


Figure 2: Low level API Implementaion Architecture

These two-level API infrastructure solutions will bring benefits such as

- Giving the freedom for third party developer to work on the specific networking technologies interested.
- Reducing effort on developing applications for Web developers
- Isolating Web developers from changes in networking technologies, addressing schemes and topologies

In this section, we will look at the implementation issues from the following aspects:

- Native code & APIs
- Exposing JavaScript APIs to third party developers
- Exposing JavaScript APIs to Web developers

We take a simple use case, e.g. George wants to view his mobile hosted MP4s on his own or a friend's set top box, to start our story on implementation.

Aspect 1: Native code & APIs

Local Discovery: Set Box can be reached via local connection. George to discover set box via local discovery – Internet access is not essential.

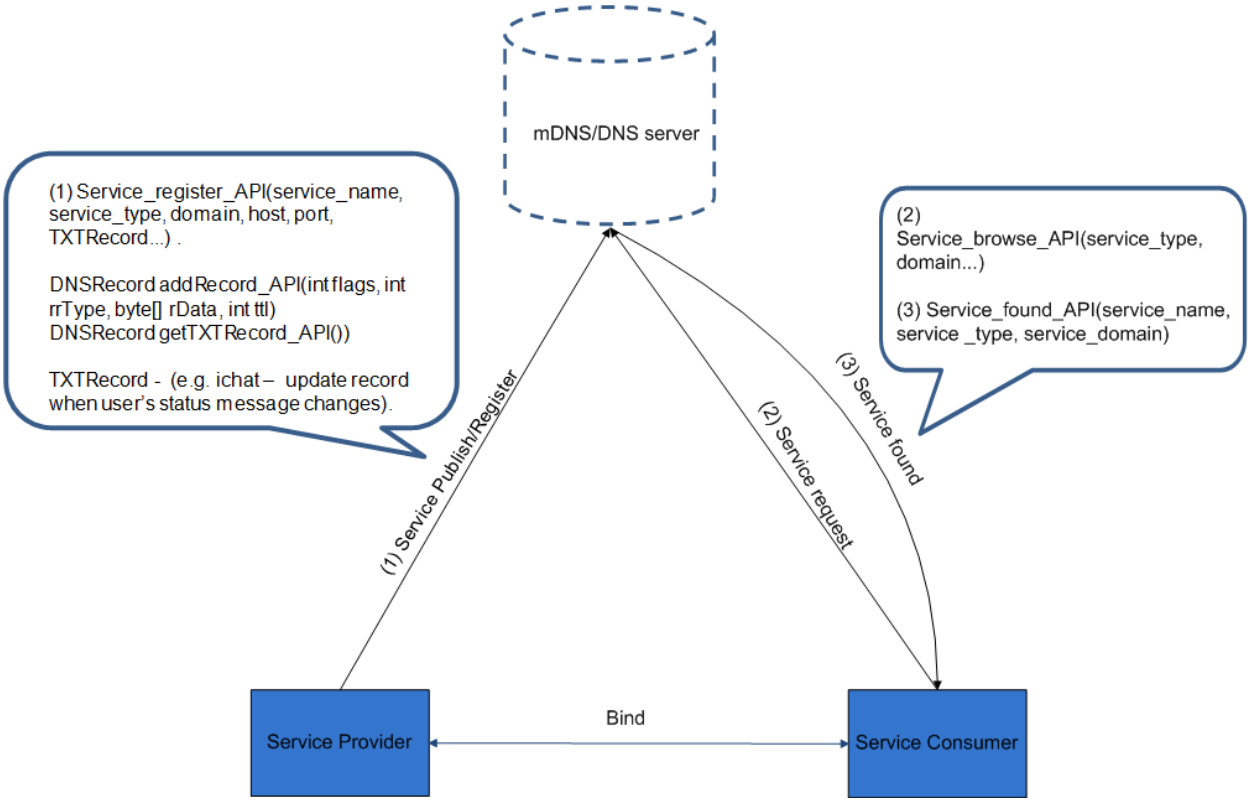
George's mobile scans for the set box with any of the following local discovery implementations:

Avahi - Zeroconf implementation

Avahi is a ZeroConf implementation. It provides a good range of for multicast DNS/DNS-SD service discovery. It is under the GNU Lesser General Public License (LGPL). With the advantage being open source over Apple's Bojour implementation, Avahi had already become the de-facto standard implementation of mDNS/DNS-SD on free operating systems such as Linux.

Avahi provides a set of language bindings, e.g . python, C, C++. It ships with most Linux and *BSD distributions. Avahi source codes are available to download free online.

Following figure illustrated some main APIs used in avahi.



Example code is available in our early demo - [http://www.w3.org/2011/04/discovery.html].

UPnP implementations

There are a few UPnP open source implementations existing for Linux system, e.g. gupnp-tools and upnp-Inspector. These implementations may come with a lot dependencies, e.g. gupnp implementation depends heavily on glib. As SSDP in UPnP is not so complicated itself, an alternative way is to write our

own SSDP codes, e.g. SSDP discovery source is available in our early demo http://www.w3.org/2011/04/discovery.html. It uses scan_ssdp to launch a thread to discover UPnP devices by sending SSDP M-SEARCH message over multicast UDP to 239.255.255.250:1900. It then listens for unicast responses and multicast notifications. When a new location is seen, a further thread is launched to retrieve the XML document describing the device's services. This is then passed back to the Web page script via the browser's pluginthreadasynccall which calls the plugin on the UI thread, triggering a call back to the Web page.

In local discovery mechanism availability of service description differs. ZeroConf publishes and resolve devices based on service type. Devices connect and bind to each other and then can exchange service description or driver to communicate. In contrast, UPnP mechanism makes service description available where UPnP based device has published device information; it is in form of Web server. Though information about services is relevant after connecting, device can look into service information before deciding to connect to it. This feature is not present in ZeroConf. ZeroConf focuses on discovery aspect and defines only service type

In both Zeroconf and UPnP, services are described in an .xml file. This should be mapped in to the service descriptions in webinos config.xml manifest defines the services utilizing the webinos:shared element.

BlueZ - Bluetooth

BlueZ is a widespread Bluetooth stack implementations initially developed by Qualcomm. It is included with the official Linux kernel distributions. As of 2006, the BlueZ stack supports all core Bluetooth protocols and layers. BlueZ provides a set of language bindings such as Python, C, Java, C++ etc.

The Host Controller Interface (HCI) socket in BlueZ provides a direct connection to the microcontroller on the local Bluetooth adapter. For tasks requiring precise control over the Bluetooth controller, such as asynchronous device discovery, HCI socket type shall be used. For a simple discovery procedure, the following APIs shall be available:

- Gets the device ID for a specified local HCI device. e.g. hci_get_route()
- Opens HCI socket associated with the device. e.g. hci_open_dev()
- Performs an HCI inquiry to discover Bluetooth devices. e.g. hci_inquiry().

An example code is available in our early demo - [http://www.w3.org/2011/04/discovery.html].

OpenSLP – SLP

The OpenSLP project is an effort to develop an open-source implementation of the IETF Service Location Protocol suitable for commercial and non-commercial application. It is under the Caldera Systems open source license - a license that is legally compatible with the popular BSD open source license[http://www.openslp.org/]. OpenSLP is hosted by SourceForge.net and source code and binaries are free for download.

Main APIs correspond to service discovery are:

- Registration - SLPReg (..., service_type, service_attribute, callback...);
- Service find - SLPFindSrvs (..., service_type, filter, callback...);

An example code is also available in our early demo - [http://www.w3.org/2011/04/discovery.html].

Local discovery with serverless XMPP - IP address is essential

To use XMPP for serverless messaging, ZeroConf is used to publish and resolve service. Once service is resolved and connected, XMPP is used for message exchange.

A typical implementation for third party developer will involve following function calls:

1. connect(nick_name, last_name, ...);
2. start_server(port)
3. avahi_entry_group_add_service (nick_name@machinename, srv_type(_presence._tcp), port)
4. avahi_entry_group_add_address(...,machinename.local, IPAddress);
5. avahi_service_browse_new(_presence._tcp)
6. socket_bind(ipaddr_port);
7. send_stanza(userid);

Calls are similar with server based XMPP except instead of DNS SRV, mDNS is used to locate services.
There are no identified open source implementations for XMPP Serverless Messaging. Initial implementation demo to experiment with the XMPP serverless messaging: [http://dev.webinos.org/redmine/attachments/662/xmpp_rfc6120_xep0174.zip].

Remote discovery: The set box is remotely located from George's Mobile or local discovery is not available. George to discover the set-top-box via Internet access

XMPP is used in this mechanism. The details described here are low level and protocol level and does not describe how webinos system will interact with Web developer. The information is under hood work done by webinos platform.

A typical call flow for third party developer implementation

1. connect(userid@domain/resource, password);
2. ipaddr_port = resolve(domain);
3. sock = socket_bind(ipaddr_port);
4. bind_id = xmpp_stream_feature_exchange(sock, resource);
5. send_presence_subscription_info(services);

6. send_stanza(userid);

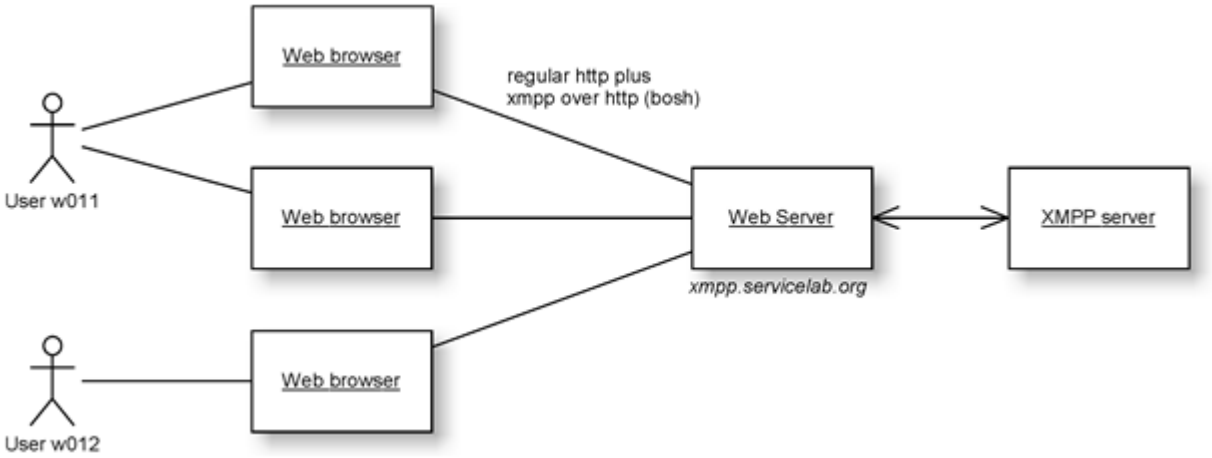
There are several open source implementation of XMPP core, libstrophe is implemented as C library and also as JavaScript library. As part of experimentation with XMPP, a demo initial implementation based on UML sequence diagram described in remote discovery of protocol definition was implemented. It is implementation in C as NPAPI plugin. It is limited in comparison to libstrophe but provides platform as a base for webinos platform.
[http://dev.webinos.org/redmine/attachments/662/xmpp_rfc6120_xep0174.zip].

XmppDemo

Another XMPP implementation for service discovery was implemented in JavaScript. Purpose of this demo was to investigate the use of XMPP for webinos service discovery (as opposed to trying to implement webinos in JavaScript). Firstly, a tiny introduction to XMPP addressing:

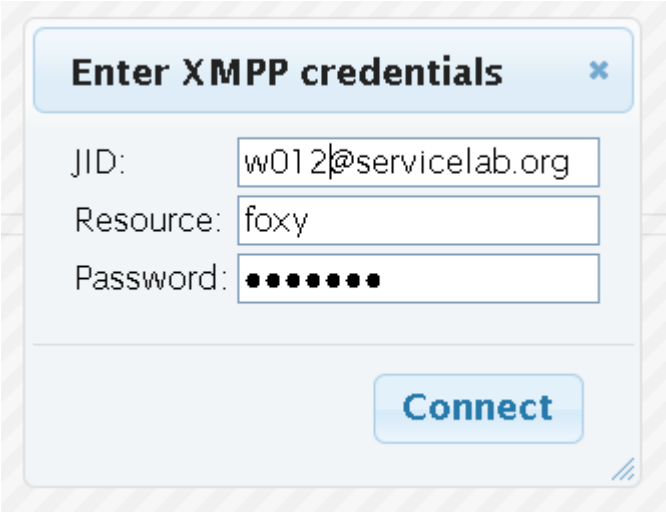
Within XMPP addressing is done using Jabber ID's (in the early days before standardisation XMPP was called Jabber). A full Jabber ID - or JID - consists of a user part, a domain part and a resource, e.g. william@example.com/mobile. Users can be connected through multiple resources at the same time. If one addresses a user without a resource (e.g. send a message to william@example.com) this means that the server should determine at what resource the user is best reachable.

The demo setup is like this:



The above figure depicts two users w011 and w012 of which the first one uses two browser instances (for example one on a laptop and one on a mobile phone) and the second one uses just a single instance. Note that the demo easily supports dozens of people using multiple resources simultaneously, even though the figure contains a mere two users with three resources.

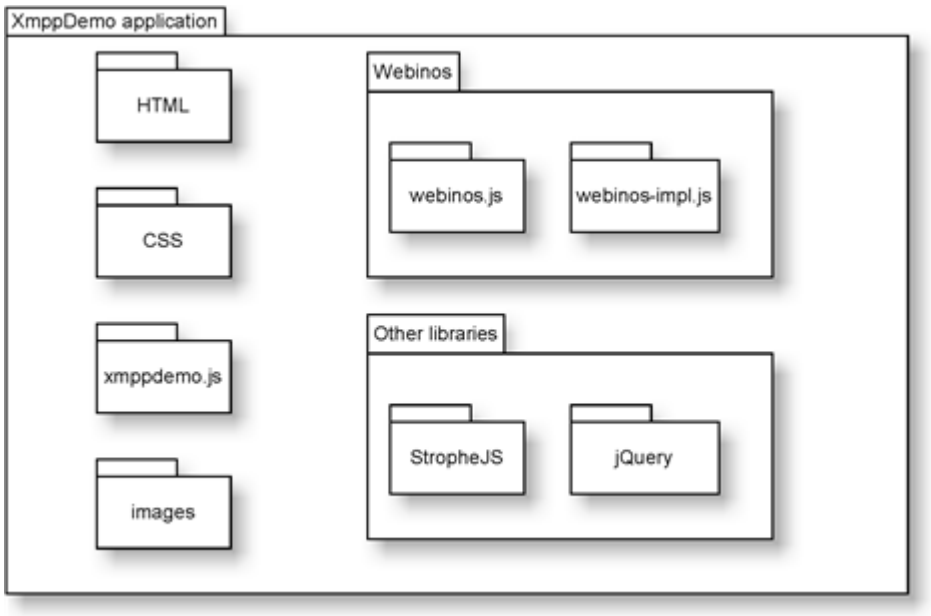
The demo supports freely chosen resource names (which are prefilled but changable):



The user names and buddy lists are preconfigured and could be chosen from:

- w011@servicelab.org t/m w015@servicelab.org (friends group 1, personal zones 1..5 within that group)
- ...
- w081@servicelab.org t/m w085@servicelab.org (friends group 2, personal zones 1..5 within that group)
  Password for all users is webinos.

The browser runs a Web application that is retrieved from the Web server. This application consists of HTML, CSS and JavaScript. The Web server doubles as a BOSH server that converts XMPP over HTTP (from and to the browser) to regular XMPP (from and to the XMPP server). In order to keep things clean and understandable the application is highly modularised as is shown here:



- xmppdemo.js contains the code for the application itself; setting up webinos, handlers and managing the user interface. It leans on the jQuery JavaScript library (see http://jquery.com/).
- webinos.js describes the interface, i.e. the way applications use webinos. In addition, it provides a model for services.
- webinos-impl.js contains the actual webinos implementation. As stated above, in the future webinos is not to be coded in JavaScript but purposes of the experiment it proved quite conventient. For communication with the server (using XMPP over HTTP or BOSH, see) it uses the Strophe library (see http://strophe.im/strophejs/).

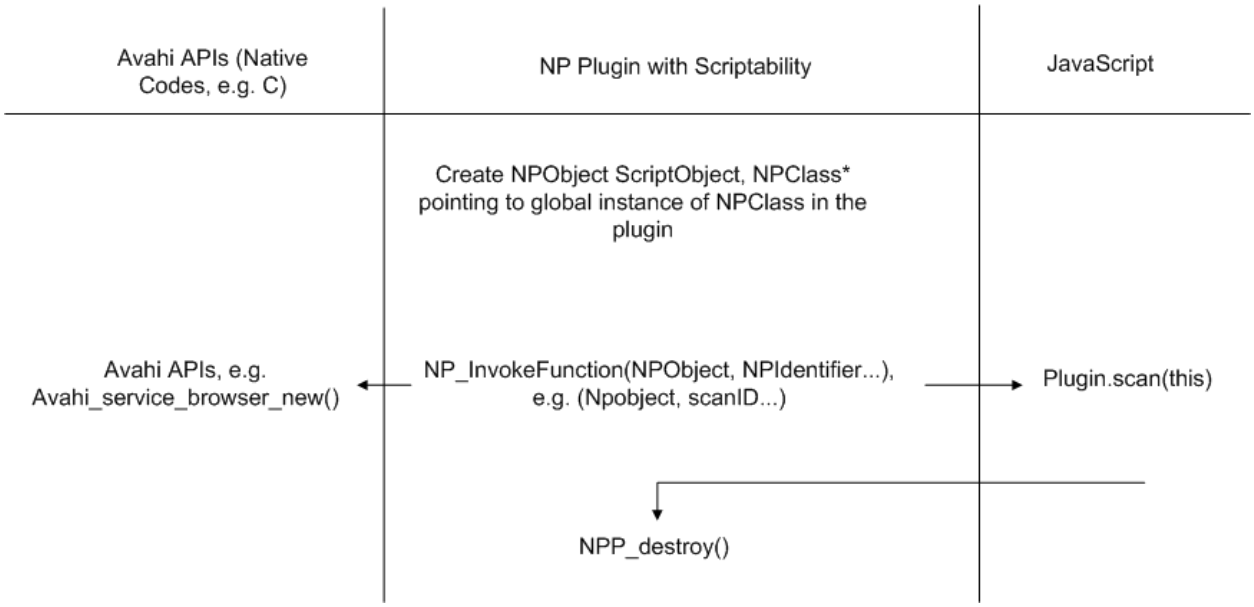
Part of the code (especially the debug console) is based on example code from the book "Professional XMPP Programming with JavaScript and jQuery" (http://professionalxmpp.com/). The full implementation can be obtained by accessing http://xmpp.servicelab.org/xmppdemo/. The JavaScript files are extensively commented.

Finally, some comments on what we have learned from this demo:

- XMPP architecture, naming and addressing map quite good on the webinos requirements
- The example build on the XEP-0115 specification for service discovery (http://xmpp.org/extensions/xep-0115.html), which is elegant and suitable, but
- does not scale very well. Some sort of 'intelligent massive service discovery' extension should be designed specifically for webinos-like applications.

Aspect 2: Expose JavaScript APIs to third party developers

Because most local discovery libraries discussed above do not provide language bindings with JavaScript, a wrapper to map between the native code and JavaScript is required. One solution is to use a cross browser plugin. Following figure shows an example on exposing avahi_service_browser_new() api to a JavaScript object using NPAPI. Refer section on plugin for more details on Browser plug-in/extension handling.



In the case of using XMPP for local discovery, instead of scan, connect function is called with user details such as nickname, lastname, email or any other detail that would like to broadcast.

Aspect 3: Expose JavaScript APIs to Web developers

The high level discovery implementation is to expose discovery interfaces, device and service features for different internetworking technologies to Web developers and end users, ideally in JavaScript. This means to publish, find and bind services via Web page scripts. The scripts listen for requests from other browsers and then respond in some manner. This might involve extensions on the current browsers with a server of some kind. An extension on browser is out of the scope of discovery and will not be discussed here.

Webinos has defined a set of high level service discovery APIs to find and bind services that are exposed locally or remotely. Local services include services exposed via local connectivity's such as BT, WiFI or USB whilst remote services cover services exposed within personal zone and cross personal zones for different users. Basically, the webinos service discovery module enables Web developers to find service (it includes devices) based on a certain search criteria. The found services are listed in a selection list in an asynchronous manner. Once the user selected a service, the implementation of the service is initiated by binding to the service. For further details on these APIs and example codes, please refer WP3.2 Discovery APIs.

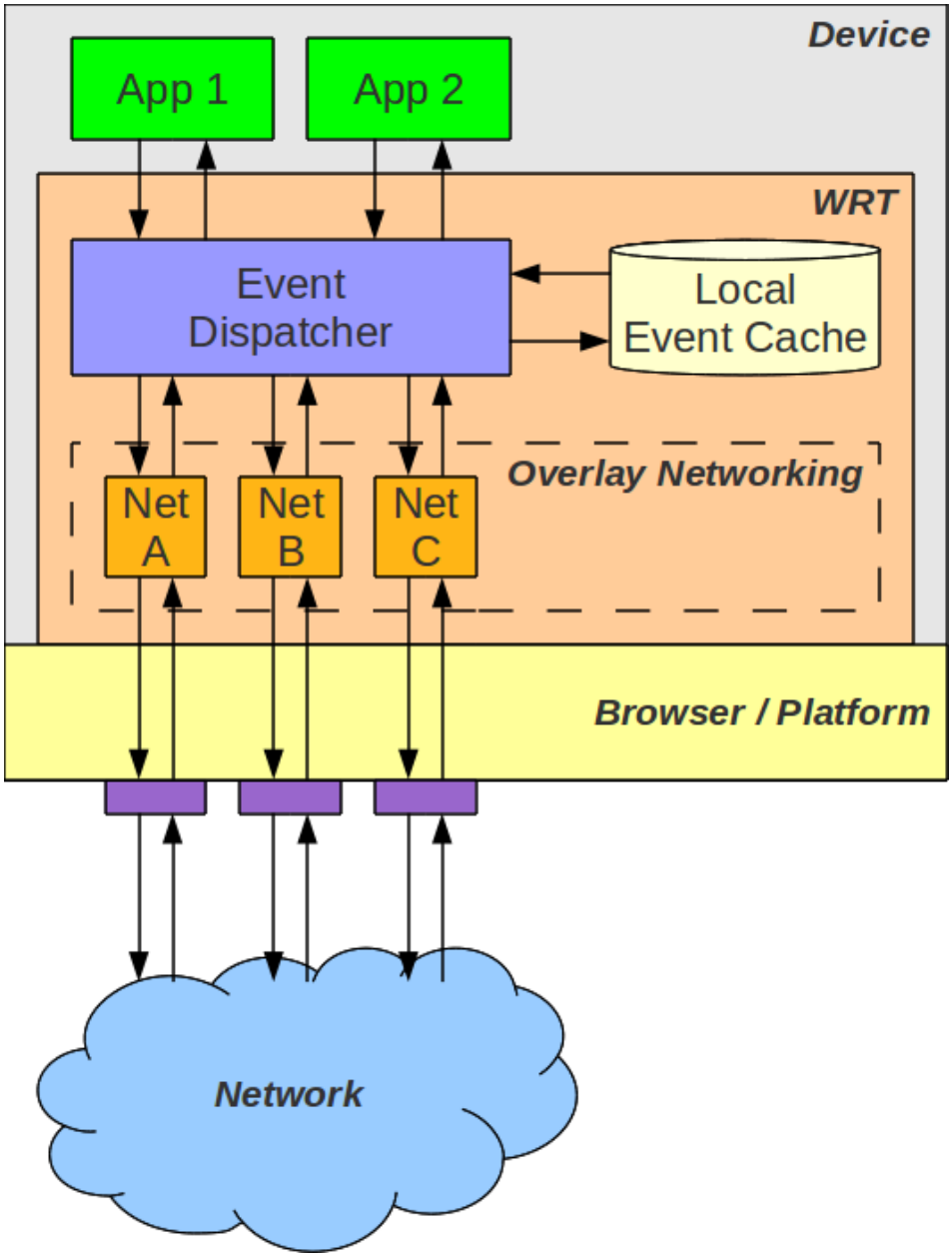
Messaging

The event handling architecture revolves around the concept of having an "event dispatcher", that is a part of the WRT devoted to routing events among addressable entities, network interfaces and part of the non-volatile device-local storage called "Local Event Cache", where not-yet-delivered events can temporarily be stored.

Such event dispatcher is supposed to perform event routing in a very generic fashion, thus not needing to have knowledge of the specific event-based protocols in order to properly work; furthermore, it relies on the interconnect technology abstraction and NAT traversal capabilities provided by the overlay networking system to enable easy and consistent communication with the outside.

Applications are offered a low level eventing API that interfaces with the event dispatcher and allows them to send/forward events to other entities, whether local or remote, and to receive events addressed to themselves or to the addressable entities they control (e.g., a service created by the application) in a seamless way.

Higher level event-based protocols, then, can be easily defined and implemented on top of this basic architecture, possibly abstracting away details of this same architecture that are of little or no interest at an application level, thus keeping their usage as simple as it is intended to be.



Formal Specification

Event object

An event is an object with the following attributes:

Attribute	Description	Required	Data type
Type	Identifies the event type and determines payload semantics	Yes	String that matches the following regular expression: [_a-zA-Z][_a-zA-Z0-9]*

Source	Represents the entity that originally sent the event	Yes	Addressable entity object reference
Destinations	3 distinct sets of objects representing original destination entities, one labeled "to" (primary recipients), another labeled "cc" (secondary recipients) and the last labeled "bcc" (blind-carbon-copy recipients)	Yes	3 distinct sets of addressable entity object references
ID	Event identifier	Yes	String holding the lowercase hexadecimal digest obtained as the result of a SHA-256 hash operation
In response to	Reference to the event that this event is a response to	No	Event object reference
Generation timestamp	Moment in time in which the event is generated by the original event source	No	A suitable date/time representation with millisecond precision or better
Expiry timestamp	Moment in time past which the event is no more valid or meaningful	No	A suitable date/time representation with millisecond precision or better
Delivery notification wanted	Indicates whether the sending entity wants to be notified by the receiving entities about the result of the event delivery	Yes	Boolean value
Addressing sensitive	Indicates whether the computation of the event ID should depend on the value of the "Source" attribute and the "to" set of the "Destinations" attribute	Yes	Boolean value
Forwarding	Addressing information on the entities involved in the last event forwarding with optional timestamp	Only when the event is forwarded	An event forwarding object (described later)
Payload	Event type-specific data	No	String

EventForwarding object

An event forwarding is an object with the following attributes:

Attribute	Description	Required	Data type
Source	Represents the entity that forwarded the event	Yes	Addressable entity object reference
Destinations	3 distinct sets of objects representing destination entities, one labeled "to" (primary recipients), another labeled "cc" (secondary recipients) and the last labeled "bcc" (blind-carbon-copy recipients)	Yes	3 distinct sets of addressable entity object references
Timestamp	Moment in time in which the event is forwarded	No	A suitable date/time representation with millisecond precision or better

This specification does not demand the usage of a specific serialization for transmitting events, yet it presupposes that:

- the actual serialization has a one-to-one mapping with Unicode code points for addresses and payload data;
- a one-to-one mapping between addressable entities and address strings is defined.

The event ID does not need to be transmitted, since it can be computed as the lowercase hexadecimal digest obtained by performing a SHA-256 hash on the following event object to string serialization if the "Addressing sensitive" field is specified as true:

```
type|source|to-dest-1,to-dest-2,...,to-dest-n|in-response-to|generation-
timestamp|expiry-timestamp|payload
```

or otherwise:

```
type|in-response-to|generation-timestamp|expiry-timestamp|payload
```

where:

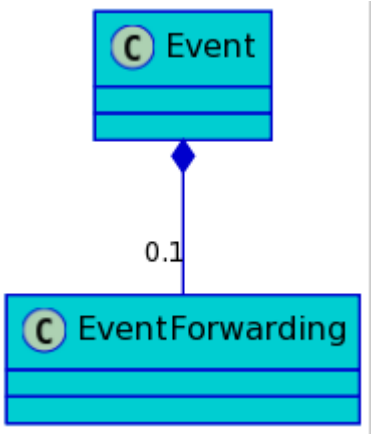
- *type* is a copy of the string held by the Type attribute;
- *source* is the escaped (see below) address string corresponding to the entity the "Source attribute" refers to;

- *to-dest-1*, *to-dest-2*, ..., *to-dest-n* are the escaped (see below) address strings corresponding to the entities in the "to" set of the "Destinations" attribute, sorted by ascending Unicode code points (sorting to be applied before escaping);
- *in-response-to* is the event ID of the event that the In response to attribute refers to (lowercase hexadecimal SHA-256 digest) or the empty string if none is specified;
- *generation-timestamp* is the string representation of the content of the Generation timestamp attribute obtained by base-10 formatting the number of milliseconds since 1970/01/01 relative to UTC or the empty string if none is specified;
- *expiry-timestamp* is the string representation of the content of the Expiry timestamp attribute obtained by base-10 formatting the number of milliseconds since 1970/01/01 relative to UTC or the empty string if none is specified;
- *payload* is a copy of the string held by the Payload attribute or the empty string if none is specified.

The escaped *source* and *to-dest-x* strings **MUST** be escaped so that the resulting strings are computed by applying the following substitutions in this exact order:

- each '\' character is substituted with '\\';
- each '|' character is substituted with '\|';
- each ',' character is substituted with '\,'.

Relationship of these objects



Protocol definitions

This section shortly illustrates how to define and implement event-based protocols and defines two event-based protocols that will be specially handled by the WRT.

Defining event-based protocols

The event handling architecture was designed in a very generic fashion to allow the definition and implementation of custom event-based protocols with ease.

While there are several legitimate ways in which such a goal can be accomplished, in a typical scenario it should be possible to do that by:

- choosing one or more static values for the Type attribute that are unique to the protocol being defined;
- verbally defining rules that describe how protocol-specific data maps to the underlying webinos event handling system, which of its features it is legal to use (e.g., stating that events are only valid if coming from a certain kind of source w.r.t. the protocol being defined) and how protocol-specific events ought to be processed;
- defining the payload semantics;
- defining a JavaScript API that partly or completely hides away the webinos event handling system;
- developing an implementation of the defined API that makes use of the webinos Event Handling API.

Delivery notifications

The delivery notification protocol is part of the core webinos event handling protocol and it provides a handful of ICMP-like functionality to the system.

A delivery notification event **SHOULD** be generated when an entity receives or is about to receive an event having the "Delivery notification wanted" attribute specified as true and such entity is a primary recipient among the last forwarding destinations or, if none, among original destinations, possibly according to specific policy rules. Delivery notifications **MUST NOT** be generated in any other case.

In order for a delivery notification event to be considered valid, the following attributes **MUST** be set as hereby described:

Attribute	**Valid values**
Type	"deliveryNotification"
Destinations	The "to" set **MUST** contain only a reference to the last entity that forwarded the event because of which this notification was generated or, if none, the entity specified by the Source attribute
In response to	Reference to the event because of which this notification was generated
Delivery notification	false

wanted	
Addressing sensitive	true
Forwarding	unset

The Payload attribute **MUST** be specfied as one of the following values:

Value	**Meaning**
"ok"	Event successfully delivered
"duplicate"	An event with the same ID was already delivered to the recipient
"invalid"	The recipient got an invalid event (e.g., transmission error)
"badDestination"	The intended recipient is unknown or unreachable
"expired"	The event expired before the actual delivery, according to its "Expiry timestamp" attribute
"refused"	The event could not be received because of lack of authorization and/or policy settings
"noReference"	The recipient does not hold a local reference to the event specified by the "In response to" attribute

Furthermore, in case of errors, the "Source" attribute of the event **MAY** indicate an entity other than the intended recipient of the original event (e.g., a "badDestination" delivery notification **SHOULD NOT** have the "Source" attribute indicating the unknown/unreachable recipient).

This protocol is meant to be automatically handled by the WRT without requiring any extra effort on the developer side: the Event Handling API **SHALL** prevent from creating events whose type is "deliveryNotification" and the WRT **SHALL** handle those events itself and **SHALL** ensure that no such events are delivered to the actual recipients.

RPC

The RPC protocol provides a common event-based way of performing RPC.

It basically wraps JSON-RPC 2.0 objects and batches into webinos events, hence concepts, terminology, payload syntax and semantics, and behavior are meant to be aligned with that specification.

A JSON-RPC 2.0 request object or request batch is wrapped into a webinos RPC request event that, in order to be considered valid, **MUST** have the following attributes set as hereby described:

Attribute	**Valid values**
Type	"JSONRPC20Request"
In response to	unset
Addressing sensitive	true
Forwarding	unset
Payload	JSON-RPC 2.0 request object or request batch

When an entity offering RPC interfaces receives an RPC request event that does not contain a notification or a notification-only batch and said entity is a primary recipient of that event, it **SHOULD** generate and send an RPC response event, possibly according to specific policy rules.

Similarly to webinos RPC request events, a JSON-RPC 2.0 response object or response batch is wrapped into a webinos RPC response event that, in order to be considered valid, **MUST** have the following attributes set as hereby described:

Attribute	**Valid values**
Type	"JSONRPC20Response"
Destination	The "to" set **MUST** contain only a reference to the entity specified by the "Source" attribute in the RPC request event because of which this RPC response event was generated
In response to	Reference to the RPC request event because of which this RPC response event was generated
Addressing sensitive	true
Forwarding	unset
Payload	JSON-RPC 2.0 response object or response batch

Note: the JSON-RPC 2.0 specification states that notifications **MUST NOT** be replied to, and the same holds true for this specification; nevertheless it is still possible to use the "Delivery notification wanted"

attribute to be informed about the delivery of an RPC request event, even if it contains a notification or a notification-only batch.

This protocol is meant to be automatically handled by the WRT without requiring any extra effort on the developer side: the Event Handling API and the event dispatcher **SHALL** prevent applications from creating and receiving events whose type is "JSONRPC20Request" or "JSONRPC20Response", while it **SHALL** be possible to expose RPC interfaces by describing them in the config.xml file, as specified in the Foundations section of this specification, and the WRT **SHALL** automatically create functions with corresponding names into the local JavaScript object that represents a discovered service, mapping calls to those functions to asynchronous RPC requests and responses operated via this protocol.

Event routing

This section specifies the inner working of the webinos' event handling mechanism at various levels, thus describing the expected behavior of the event dispatcher.

The event dispatcher basically performs four subsequent operations on each event, in the following order:

1. *validation*, in which the event metadata is checked to ensure validity, well-formedness and consistency;
2. *route determination*, in which the event dispatcher determines how each recipient is to be reached;
3. *policy enforcement*, to ensure that local policy settings allow for the event to be further processed;
4. *routing*, in which events are actually exchanged among applications, the Local Event Cache and/or network interfaces.

Event validation failure **MAY** result in the automatic generation and sending of delivery notification events by the WRT, if the event source (forwarding source, or, if none, original source) does not reside on the device and the event has the "Delivery notification wanted" attribute specified as true. This kind of behavior, however, is implementation-defined. Failure to validate an event **SHALL** prevent the event from being further processed.

In the route determination phase the event dispatcher **SHALL** go through the list of recipients (forwarding destinations or, if none, original destinations) and determine whether they are local or remote. In the former case, it **SHALL** ensure that they do actually exist and are reachable (policy settings apart) and, if this condition does not hold true, it **SHOULD** automatically generate and send a "badDestination" delivery notification event to the event source (forwarding source, or, if none, original source, no matter if local or remote). In the latter case it **SHALL** determine whether each recipient is reachable within the local networks the device is connected to or not, and also which network interface is going to be used to reach each recipient, so that this information is available in the policy enforcement phase.

Route determination for events **SHALL** be performed once per recipient, so that individual failures **SHALL** only affect one event/recipient combination. Failures during the route determination phase **SHALL** prevent the event from being further processed for that event/recipient combination.

Local policy rules are also implementation-defined, hence not described in this specification. Again, if the local policy prevents the event from being delivered and the event source (forwarding source, or, if none, original source) does not reside on the device and the event has the "Delivery notification wanted" attribute specified as true, a "refused" delivery notification event **SHOULD** be automatically generated by the WRT and sent back to the source.

Policy enforcement of events coming from remote sources **SHALL** be performed once per local recipient, so that individual failures **SHALL** only affect one event/recipient combination. Failures during the policy enforcement phase **SHALL** prevent the event from being further processed for that event/recipient combination.

Errors during the validation and policy enforcement phases when the event source (forwarding source, or, if none, original source) is local **SHALL**, instead, result in appropriate exceptions being thrown at the application-level.

Application-level event routing

The event handling functionality exposed to the application, at the lowest level, consists of a restricted set of conceptually simple operations: generating and sending events, or forwarding them, and registering listeners for incoming events.

This implies that the event dispatcher **SHALL** be able to somehow create the JavaScript representations of incoming event objects and to properly trigger the execution of the relevant listener callbacks.

Each application **SHALL** be allowed to handle events from/to addressable entities that it "owns" (e.g., the application itself, the services it exposes), obviously according to the constraints imposed by local policy rules and authorizations, hence it **SHALL** be able to both receive events directed to any addressable entity that it owns and to send/forward event on the behalf of any addressable entity it owns.

Other than that, at this level, the behavior of the system is highly dependent on the actual webinos Event Handling API specification, hence that document is to be considered the reference for determining whether a given implementation is conformant or not.

Device-level event routing

Each local addressable entity **SHOULD** be associated with some sort of buffer holding, at least, incoming event IDs, in order to limit the likelihood of receiving duplicate events. Such buffer **SHOULD** be managed by an algorithm that, for each incoming event:

1. checks whether its event ID is already present in the buffer, in which case the event **SHOULD** be dropped, possibly generating and sending back "duplicate" delivery notification events;

2. if the event ID is not present, instead, it is either added to the buffer if there is enough free space, otherwise an existing entry in such buffer is replaced with a new entry corresponding to the ID of the incoming event - it is recommended that the criterion by which an old entry is chosen to be replaced tends to keep inside the buffer entries corresponding to IDs of events that were received more than once.

Such algorithm **SHOULD** also possibly make clever use of the "Expiry Timestamp" attribute of events to keep buffer usage low.

When an event has successfully gone through validation, route determination and policy enforcement, the event dispatcher **SHALL**, for each local recipient:

1. check if an event with the same ID was already delivered to the recipient using the incoming event buffer associated to that recipient, if available;
2. create a copy of the event object for each unique local entity;
3. for each event object copy:
    1. if the event is not a forwarded event (i.e., the "Forwarding" attribute is unspecified), remove all entities from the "bcc" set of the "Destinations" attribute that are not owned by the application by which the addressable entity is owned;
    2. otherwise, remove all entities from the "bcc" set of the "Forwardings" attribute that are not owned by the application by which the addressable entity is owned.
4. call the relevant listeners, if any;
5. possibly generate and send back "ok" delivery notification events.

Network-level event routing

The event dispatcher **SHALL** use the local PZP to receive and send/forward events from/to local networks, while it **SHALL** also rely on the PZH for routing events from/to entities living in remote networks.

In any case, the PZP **SHALL** pass events coming from the network to the event dispatcher and **SHALL** physically transmit outgoing events passed to it by the event dispatcher, leveraging off the overlay networking layer. This means that, according to whether the event source (forwarding source, or, if none, original source) is local or not, the eventing system will behave differently.

Incoming events (remote event source)

In this case the event dispatcher **SHALL** only try to transmit the event to local entities.

In particular, during the route determination phase, the event dispatcher **SHALL**:

1. ensure that the event is directed to at least one local entity;
2. if this is not the case:

1. if the "Delivery notification wanted" attribute of the event is specified as true, it **SHOULD** generate and send back a "badDestination" delivery notification event;

2. the event is dropped;

3. otherwise the event processing **SHALL** proceed as described in the "Device-level routing" section, only for local destinations.

Outgoing events (local event source)

In this case the event dispatcher **SHALL** act as specified in the "Device-level routing" section w.r.t. local destinations, while for remote destinations it **SHALL** determine which can be reached within local networks and which reside in remote networks in the route determination phase.

Then, if the event has successfully gone through the policy enforcement phase, the event dispatcher **SHALL**:

1. given the union of recipients in the "to" and "cc" sets of destinations that are to be reached via local networks, excluding those destinations that reside on the same device as at least one destination in the "bcc" set:

    1. for each involved local network:

        1. if the set of destinations to be reached through that network contains more than one element, the event **SHOULD** be preferably be transmitted through multicast/broadcast facilities, if any are available, in order to save bandwidth, with the "bcc" set unspecified;

        2. otherwise (only one destination or no multicast/broadcast available) the event **SHALL** be transmitted once per destination on that network with the "bcc" set unspecified;

2. for each group of destinations specified in the "bcc" set that reside on a single device, the event **SHALL** be transmitted to that device, with the "bcc" set only containing that specific group of destinations;

3. if there are one or more remote destinations involved, the event **SHALL** be transmitted to the PZH for delivery to remote entities - the data transmitted to the PZH **SHALL** contain a suitable serialization of the event and a supplementary list of destinations, among the event destinations, that the PZH is asked to transmit the event to.

E.g., an event has the following destinations:

- *to*: A, B, C;
- *cc*: D, E, F;
- *bcc*: G, H, I;

and:

- A and G are local;
- B and C reside on device X;
- E, H and I reside on device Y;
- D and F reside on device Z;
- device X and device Z are reached through the same local network N;
- device Y is connected to a remote network;

then, the event dispatcher will act as follows:

- A is delivered a copy of the event with empty "bcc" set, according to what is specified in the "Device-level routing" section;
- G is delivered a copy of the event with the "bcc" set only containing G, according to what is specified in the "Device-level routing" section;
- since all entities (B, C, D and F) on devices X and Z are not in the "bcc" set, and since devices X and Z are to be reached through the local network N, a copy of the event with empty "bcc" set is trasmitted to both devices using multicast/broadcast-like facilities, if available, or otherwise one copy for each device is transmitted;
- one copy of the event is sent to the PZH for delivery to the entities on the Y device, also specifying the event has to be delivered to E, H and I, with the "bcc" set containing H and I.

The actual event transmission to the "next hop" is considered as having successfully taken place when:

1. if the interconnect technology is connection-aware (e.g., TCP over IP), the device has been somehow notified of the successfulness of transmission (e.g., TCP ACK).
2. otherwise (e.g., UDP over IP), as soon as the whole event data has been physically transmitted.

The PZH **SHALL** behave in a similar fashion for the transmission of events.

Event caching

When the actual event transmission to one or more remote destinations fails or is not currently possible (e.g., lack of connectivity), the event **SHOULD** be put in the Local Event Cache by the event dispatcher to attempt later retransmission. The frequency and number of such attempts is implementation-defined.

An event inside the Local Event Cache having the "Expiry timestamp" attribute set to a moment in time in the past **SHALL** be removed from the Local Event Cache, NO more retransmission attempts **SHALL** take place and the result **SHOULD** be reported to the sending/forwarding entity via the Event Handling API as a delivery error.

JavaScript APIs

The event handling functionality **SHALL** be available to application and third-party developers through the webinos Event Handling API.

While the API itself is quite advanced, supporting all of the current specification, a few simple examples are hereby given.

An event can be created using the webinos.events.createWebinosEvent() function, e.g.:

```
1 var evt = webinos.events.createWebinosEvent("chatMessage", {to: remoteChatApp},
"Hello!");
```

which can then be sent using the event.dispatchWebinosEvent() function, e.g.:

```
1 evt.dispatchWebinosEvent();
```

It is possible to register listeners to incoming events using the webinos.events.addWebinosEventListener() function, e.g.:

```
1 webinos.events.addWebinosEventListener(function (evt) { alert(evt.payload); },
"chatMessage");
```

Here follows the full WebIDL interface definition:

```
module events {
        [NoInterfaceObject] interface WebinosEventEntity {
                attribute DOMString id;
        };

        [NoInterfaceObject] interface WebinosEventAddressing {
                attribute WebinosEventEntity source;
                attribute WebinosEventEntity[] to;
                attribute WebinosEventEntity[] cc;
                attribute WebinosEventEntity[] bcc;
        };

        [NoInterfaceObject] interface WebinosEvent {
                readonly attribute DOMString type;
                readonly attribute WebinosEventAddressing addressing;
                readonly attribute DOMString id;
                readonly attribute WebinosEvent inResponseTo;
                readonly attribute DOMTimeStamp? timeStamp;
                readonly attribute DOMTimeStamp? expiryTimeStamp;
                readonly attribute boolean addressingSensitive;
                readonly attribute WebinosEventAddressing forwarding;
                readonly attribute DOMTimeStamp? forwardingTimeStamp;
                readonly attribute DOMString? payload;

                void dispatchWebinosEvent(
                                in optional WebinosEventCallbacks? callbacks,
                                in optional DOMTimeStamp? referenceTimeout,
                                in optional boolean sync)
                     raises(WebinosEventException);
                void forwardWebinosEvent(
                                in WebinosEventAddressing forwarding,
                                in optional boolean withTimeStamp,
                                in optional WebinosEventCallbacks? callbacks,
                                in optional DOMTimeStamp? referenceTimeout,
                                in optional boolean sync)
                     raises(WebinosEventException);
        };
```

```
exception WebinosEventException {
        unsigned short code;
        DOMString message;

        const unsigned short INVALID_ARGUMENT_ERROR     = 1;
        const unsigned short PERMISSION_DENIED_ERROR    = 2;
};

[NoInterfaceObject] interface WebinosEventDeliveryError {
        readonly attribute unsigned short code;
        readonly attribute DOMString message;

        const unsigned short UNKNOWN_ERR                = 0;
        const unsigned short INVALID                    = 1;
        const unsigned short BAD_DESTINATION            = 2;
        const unsigned short EXPIRED                    = 3;
        const unsigned short REFUSED                    = 4;
        const unsigned short NO_REFERENCE               = 5;
};

[Callback, NoInterfaceObject] interface WebinosEventCallbacks {
        void onSending(in WebinosEvent event,
                       in WebinosEventEntity recipient);
        void onCaching(in WebinosEvent event);
        void onDelivery(in WebinosEvent event,
                        in WebinosEventEntity recipient);
        void onTimeout(in WebinosEvent event,
                       in WebinosEventEntity recipient);
        void onError(in WebinosEvent event,
                     in WebinosEventEntity recipient,
                     in WebinosEventDeliveryError error);
};

[Callback=FunctionOnly] interface WebinosEventListener {
        void handleEvent(in WebinosEvent event);
};

[NoInterfaceObject] interface WebinosEventsInterface {
        WebinosEvent createWebinosEvent(
                        in DOMString type,
                        in WebinosEventAddressing addressing,
                        [TreatUndefinedAs=Null]
                          in optional DOMString? payload,
                        in optional WebinosEvent? inResponseTo,
                        in optional boolean withTimeStamp,
                        in optional DOMTimeStamp? expiryTimeStamp,
                        in optional boolean addressingSensitive)
              raises(WebinosEventException);
        DOMString addWebinosEventListener(
                        in WebinosEventListener listener,
                        [TreatUndefinedAs=Null]
                          in optional DOMString? type,
                        in optional WebinosEventEntity? source,
                        in optional WebinosEventEntity? destination)
                  raises(WebinosEventException);
        void removeWebinosEventListener(in DOMString listenerId)
             raises(WebinosEventException);
```

```
        };

        [NoInterfaceObject] interface WebinosEvents {
                readonly attribute WebinosEventsInterface events;
        };

        webinoscore::Webinos implements WebinosEvents;
};
```

Dependencies on other components

Overlay networking

The webinos' event handling mechanism uses webinos' overlay networking model to allow exchange of events on the network regardless of the underlying interconnect technology.

It is needed for the overlay networking model to provide methods to send and listen for events to/from network interfaces that do also handle the specific event serializations for each interconnect technology.

Policy

All communication **MUST** happen according to policy settings, hence policy enforcement is a fundamental part of the event handling system.

Authentication and Identity

The transmission of events on local and remote networks happens via the PZP and PZH components, hence involving the webinos' authentication and identity infrastructure.

Implementation Architecture

While this specification is relatively strict in terms of the behavior of the system, at the same time it is intended to be quite loose when it comes to suggest an implementation architecture.

This is because of two reasons: first of all, while the event handling mechanism borrows significantly from XMPP at a conceptual level, it is however pretty much a novelty, hence many real-world issues are unlikely to be foreseen at this stage; then, its actual scope of application is still to be investigated, also because no performance constraints were explicitly specified beforehand.

This section discusses some of the most crucial implementation choices, also based on some experience gained by developing a small prototype implementation, however with no claim of exhaustiveness.

Event dispatcher and PZP

The event dispatcher unit described in this specification is a logical abstraction that does not necessarily need to be implemented as a single component and it shows significant functionality overlap with the PZP component.

One possibility might be, indeed, to actually let the PZP implement most of the event dispatcher functionality. However, such a choice has a number of implications:

- if the PZP lives in a separate thread or process w.r.t. the Web application, it would be probably impossible to avoid blocking synchronization (e.g., local policy enforcement always happens synchronously), thus introducing new sources of unpredictable delays, and also, if living in a separate process, each event might need to be validated twice for security reasons (e.g., once by the API implementation running in the Web application context, then by the PZP process) - in practice, anyway, even though not ideal, the result might prove to be "good enough", at least for a first implementation phase;
- the PZP may get slower at processing requests, on average, especially if a one-main-loop design is used (e.g., while an event is being processed, all other requests are "frozen");
- since the PZP is a security-critical component, the dispatcher functionality would need to be somehow isolated from the rest of the PZP, thus probably making things a lot more complicated and likely to turn "normal bugs" into serious vulnerabilities, depending on the level of isolation.

Another possibility could consist in having a part that translates event-related requests into more basic lower-level requests for the PZP and/or other components from within the Web application context, thus letting the PZP to handle smaller and simpler requests. While such a solution could solve or mitigate the problems above, it might potentially introduce new ones:

- it could be needed to perform policy enforcement both from within the Web application context and inside the PZP, thus potentially introducing policy synchronization issues;
- while such an approach should theoretically reduce average latency, it might however have significant impact on average throughput due to synchronization overhead (smaller but more frequent requests to the PZP than before).

IPC and synchronization issues

At a device-level, data exchanges among Web applications might happen by just sharing memory references if the involved applications live within the same process. This is, however, not always the case, and also it is very likely that we need process separation in any case for security and reliability reasons, hence we will concentrate on a multi-process scenario here.

Typically, native APIs that allow to control/manipulate JavaScript execution (e.g., NPAPI, Webkit's JavaScriptCore), provide means to add hooks into the JavaScript processing loop (or, otherwise, main GUI loop), from which it is possible to safely perform JavaScript-related work for a Web page. While this allows to, e.g., create JavaScript event objects, the actual data transfer from the sender to the receiver(s) does still need to be performed using some IPC mechanism.

Of the many viable alternatives for IPC, two seem the most appropriate for the taks: Unix domain sockets (or roughly equivalent, named pipes on Windows) and shared memory.

In a few words, Unix domain sockets should make the implementation task a lot easier, also probably leading to less interoperability- and extensibility-related headaches, but on the other hand they might have serious performance implications. An approach based on shared memory, instead, would be much more challenging to be developed, especially w.r.t. security, memory usage and synchronization issues, but would probably lead to a faster and much more responsive system behavior.

The advice, here, is to first make an attempt with Unix domain sockets (or named pipes on Windows) and, in case it turns out that IPC constitutes a major performance bottleneck, to switch to either shared memory or to a hybrid approach of some kind.

JavaScript-related issues

There is a number of JavaScript-related issues that should be taken into account when developing an implementation.

First and foremost, JavaScript is a garbage collected language, and, as of now, typical JavaScript engines do not offer a native API to control this mechanism. This means that it is basically impossible to limit the performance unpredictability due to garbage collection, hence performing time-critical tasks (e.g., soft real-time tasks like real-time DSP) can be actually considered outside of the scope of the language.

This consideration alone gives us a first hint on scope of the event handling system itself, since it is mainly meant to be used by JavaScript applications and libraries through the Event Handling API. Indeed, measurements on our prototype implementation show that the generation and local transmission of a simple event might take from less than 1 ms up to 500 ms.

Furthermore, the need to prevent garbage collection of event objects when they are sent/forwarded, together with "In response to" object references and the overall asynchronous nature of the system might lead to significant memory consumption. It is, therefore, recommended that this particular issue is taken very seriously by implementors.

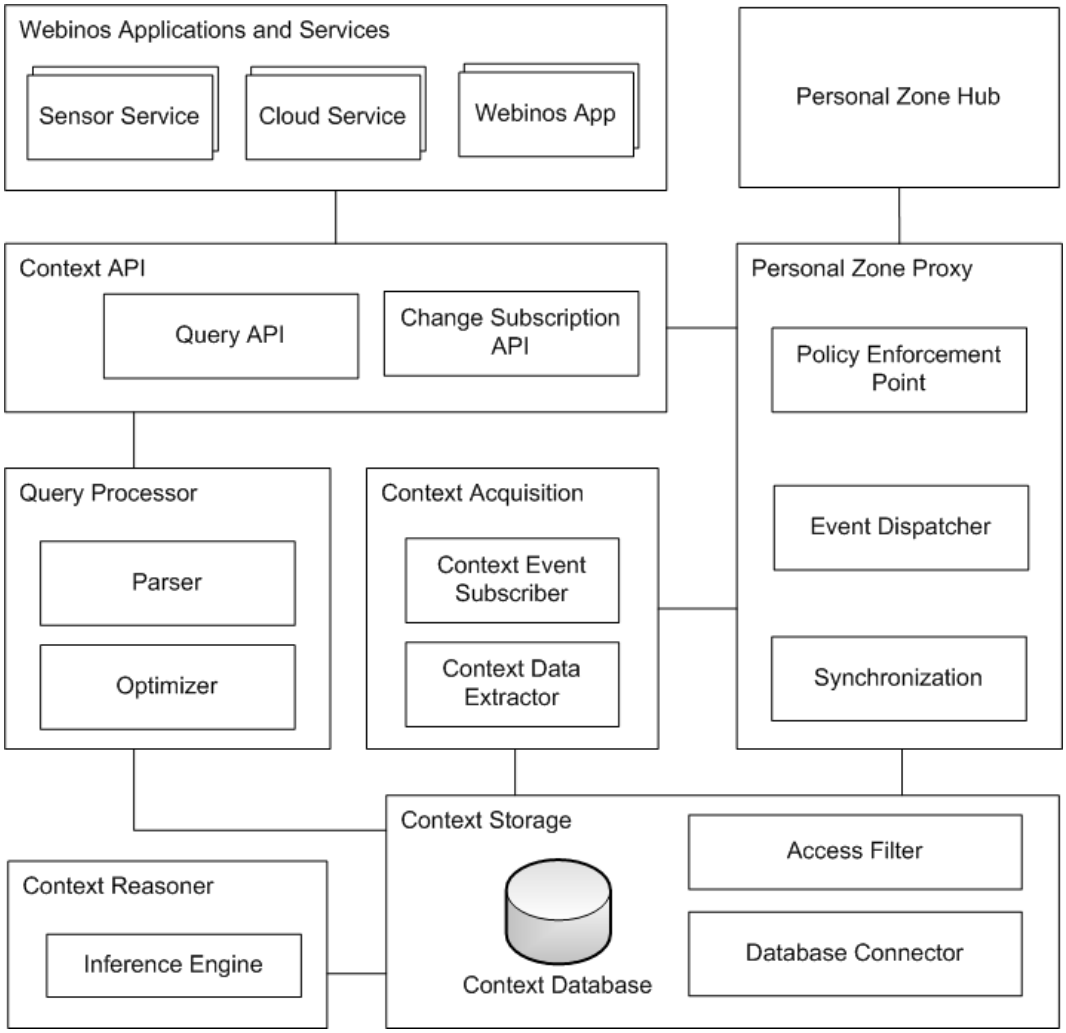
Context

The webinos context framework provides access to contextualized data in order to enable the design and operation of context-driven webinos applications/services.

The framework is responsible for collecting and storing context data (through the identification of specific context related events that happen within webinos enabled devices) and providing applications with a layer to access such data, either by querying against the storage or by being notified in real time for context changes (when specific events happen).

In addition, the context framework is closely coupled with the webinos policy and privacy enforcement framework in order to ensure secure handling of the often highly sensitive context data that is stored.

The following schema describes the conceptual architecture of the webinos context framework and illustrates its connection to the policy/privacy enforcement framework through the Policy Enforcement Point.



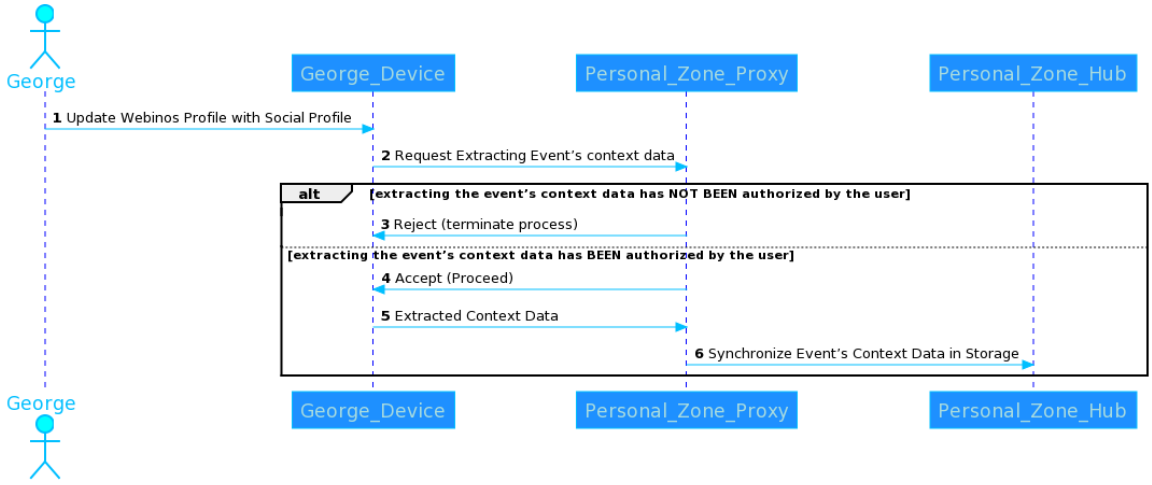
Technical Use Cases

George is a webinos user that has several webinos enabled devices, a mobile, a desktop and a set top box.

The following short use cases describes the context data acquisition process from context related events, Querying for Context Data using the Context Query API and Subscribing to receive context data updates using the Context Changes Subscription API.

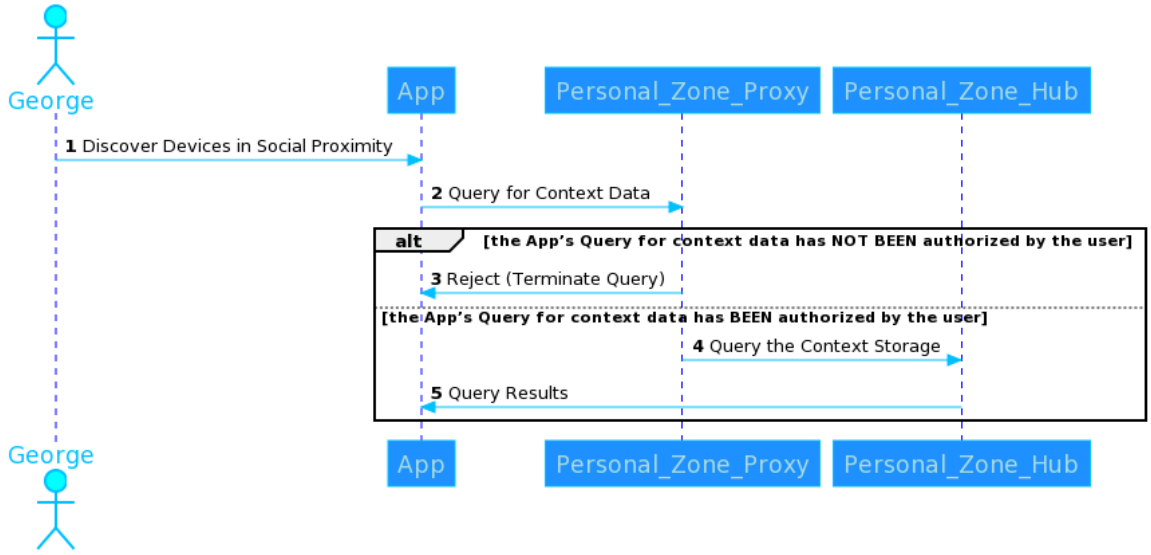
Use Case1: Acquisition of Context Data from Context Related Events

George is updating his webinos profile by adding to it one of his social network profiles (i.e. facebook). This activity constitutes a context related event and triggers a context acquisition process at the device. This means that (meta)data about George's newly added social profile will be added to his context data, provided that George has allowed for such context data acquisition in his privacy settings.



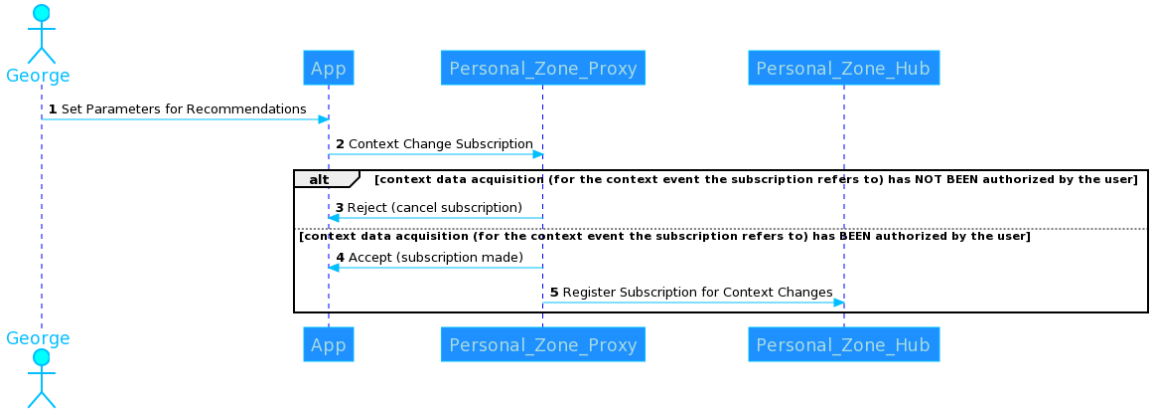
Use Case2: Query for Context Data

George is at a coffee shop with a couple of his friends John and Jos. George has bought (and downloaded in his mobile) a new single of one of his favorite artist and wants to stream it to his friends devices. For that he is using an app in his mobile that allows him to stream mp3s to other (capable for playback devices). He is accessing his app, selects his new mp3 and invokes device discovery using a social proximity filter. The app, using the webinos discovery APIs, discovers several devices around George capable of receiving streaming. In order to prioritize the discovered devices (i.e. put at the top the ones George is most likely to want to stream music to) the app uses the Context API to Query for context data that illustrate if George's device has connected with any of the discovered devices in the past or if George has any social connection with the discovered device owners.



Use Case3: Subscribe to Context Changes Updates

George has a recommendations app on his mobile TV that he has set up to provide him real-time recommendations for alternative broadcasts whenever he is watching an adventure film during primetime (21:00-00:00). The app is using the Context Change Subscription API to get notifications in real-time whenever the user sets on his TV a channel airing an adventure film between the hours 21:00 and 00:00.



Formal Specification

Context Framework

The scope of providing context-awareness within webinos is three-fold:

1. To identify context related events and register (occurring) context data
2. To enable context-aware behaviour for any webinos application by using secure access to context data regulated by user-policies

3. To be able to transform various multi-media formats for web content based on context data (content adaptations)

Therefore, the extent of the context framework in webinos is to define and implement the most suitable mechanisms (in terms of context data acquisition, representation, storage and access) in the webinos context model any webinos applications can start exhibiting context in a secure way.

The following sub-sections summarise key characteristics of the webinos context framework.

Support for creating context aware applications by enabling access to context data

Context-awareness support in webinos will allow applications in particular:

- Listen for context related events happening on device level, either by user activities (at device level) or by applications using the underlying APIs.
- Subscribe to events across applications and devices that bring about changes in context data.
- Retrieve stored context data of identified/subscribed, context-related, events.
- Retrieve stored context data about the current device or a target device where an application is to be executed.
- Retrieve stored context data about a specific user (i.e. device owner) or a particular application (i.e. an app receiving a message on a device).
- Retrieve real-time updates of context data from context related events they have subscribed to.
- Perform reasoning based on the underlying context data, for example evaluate the social proximity of discovered devices based on social connections of their owners or previous activity (i.e. sharing a device feature).

Providing for the Secure and Privacy-aware access to context information through user-negotiated policies

Context-awareness support will be secured by dedicated access control functions provided by a policy manager entity (the “policy enforcement point”). This entity constitutes an access point that provides for secure and authorized access to context information, specifically:

- Allow only the permitted applications to access context information about the user, the device, the application, etc.
- Inform the user about context and personal information that is registered, accessed and shared by applications. This excludes context information that is private for the application itself.
- Provide access control to context structures through user-defined policies.
- Implement the privacy policies that the user and the application negotiate between them.

- Securely store associations between device, user and context information and provide this information based on user preferences.
- Register their intent to access context information provided by the underlying webinos platform, given that the user permits the access to this context information originating from the webinos platform.
- Ensure that the user is offered the possibility to understand and control the access to context information originating from the underlying webinos system for all applications that register an intent.
- Ensure the security and integrity of any context information that is privately created and stored by the application, so that attackers and users can not tamper with the stored or shared context information.

Transform various multi-media formats for web content and Context-driven adaptation support

Devices in general do not support all available file formats for transcoding various multi-media formats. The result is that some files cannot be viewed in some devices, but should the content is presented on another device it would render fine. Context information can be used to identify what file types and codec support a device provides. After obtaining this information, an application may sometimes need to transform/ transcode the content into a new file or media stream. The following support will be provided by the webinos platform to application as part of its context-awareness functions:

- Check device compatibility with certain file formats.
- Provide embedded mechanisms that transform file formats.
- Support the integration of external content adaptation services.
- Use device context and user context as means to adapt the delivery and presentation of content.
- Use context information for pushed content to trigger the installation of the correct application in case the application is not yet installed on the device that is receiving the pushed content.

In order to address the needs of a multi-device and cross-platform applications, the webinos platform defines formalized ways of describing content and layout of the page to be able to provide an integration layer that allows integrating external adaptation services into the platform. The following will be provided by the platform to support variety of existing and future adaptation services:

- Extensible integration layer that allows for third party adaptation services integration.
- Support for input transformation filters that can transform content as per the required format for third party adaptation service.
- Support for output transformation filters that can transform third party adaptation services output to format required by the platform.

- Support for organizing filters into filter chains performing set of transformations over content.

Adaptation service input consists of:

- Device Context information reflecting actual state of the device
- Device user-agent string identifying the device/browser details.
- Content to transform pre-processed by filtering mechanism to fulfill adaptation service requirements.

Adaptation service output consists of:

- Transformed content that can be post-processed to meet requirements of Webinos platform.

A context model for webinos

A model in general can be described as a systematic description of an object or phenomenon that shares important characteristics with the object or phenomenon. It can be seen as a representation of relevant/ important aspects of the target object or phenomenon that is being described.

Context describes aspects of a situation seen from an actor/ entity's point of view. From such a viewpoint, context acquisition in webinos becomes the process of sensing and gathering context information to help enhance the quality and usefulness of the application. A context model represents (or guides) how these (retrieved) data are structured together in a way that makes sense. The approach in webinos is to define a very methodical way of handling context information. We believe this will be important for application developers to be able to add context-awareness to their application to benefit from it. In this section, we present the foundations for representing context information in webinos applications. It is based on a review of existing literature in the field, including some EU-projects that have been working with context-aware applications and services as well as the webinos partners existing experience in the field.

Relating context to how humans sense, think, and act

Our human mind is episodic in nature. We remember artefacts, objects, people, and information easier if we are able to remember/ recall aspects of the situation in which we experienced the entities in. Not only do we organise our thoughts and remember in an eposidic way, also our human communication is full of associations and references to past, present, future and imaginative/dreamt situations. Our cogntion helps us to perceive, remember, recall, reason, and act. In neuro-science and psychology one distinctinguishes between episodic and semantic memory - see for instance Tulving (1972).

Consider how we reflect upon printed magazines in a book section: we might remember them according to the front cover, an article, some photo we saw inside it, the title of an article, a person in a picture, or the place we read the magazine. Some of these ‘triggers’ can be difficult to capture, and record.

Context-aware systems are to some extent trying to mimic this kind of cognitive behaviour based on sensing, thinking, and acting in the situation. If a living organism is acting well within a situation, then

one would typically say that the organism is adapting well to that situation. The notion of adaptivity is therefore related to context-aware systems. Remembering context in a software program is therefore about creating a facility that stores the episodic information captured in the situation.

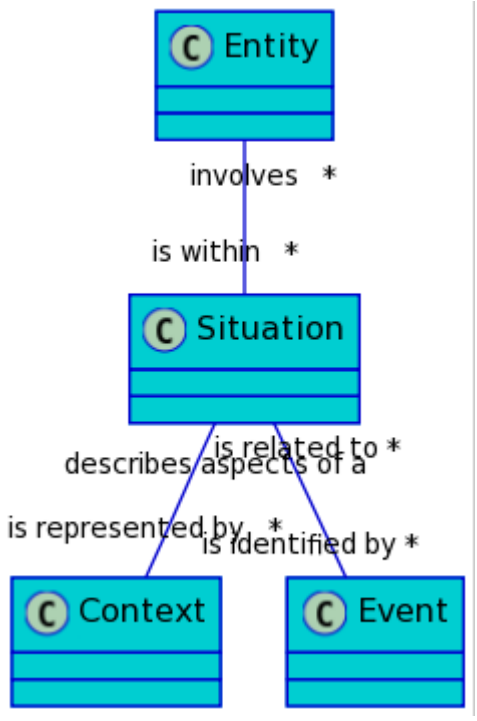
Definition of context

There are many definitions of context. This is due to research in the topic within computer science, information science, and artificial intelligence.

Context is much related to the notion of a situation. In some languages the words 'situation' and 'context' are being used to describe each other. For instance, "it is not the right situation for this yet" could easily be rephrased to "at the moment the context is not right for this" and mean almost the same. For our work, we will use the following definitions: Situations exist in the real world. There are actors/ entities involved in situations. A situation endures over a period of time. This is true whether the time is referred to as a point in time, an hour, a day, at night, in the summer, and so on. We define context as a description of the aspects of a situation.

In this way, a context captures important aspects of a situation when a context related event occurs. This means that someone has decided which particular aspects that are more important/ matters more than others and through which events these aspects can be captured. Context therefore is an explicit representation of a situation. Capturing context information can therefore be compared with taking a snapshot of a situation when a corresponding event happers, where only the most important things are being developed into the preserved "photo" (ref context).

The following figure depicts the relations between Situations, Context data, Events and webinos Entities. An entity may be any person, thing, or system that is being active or passive for that situation.

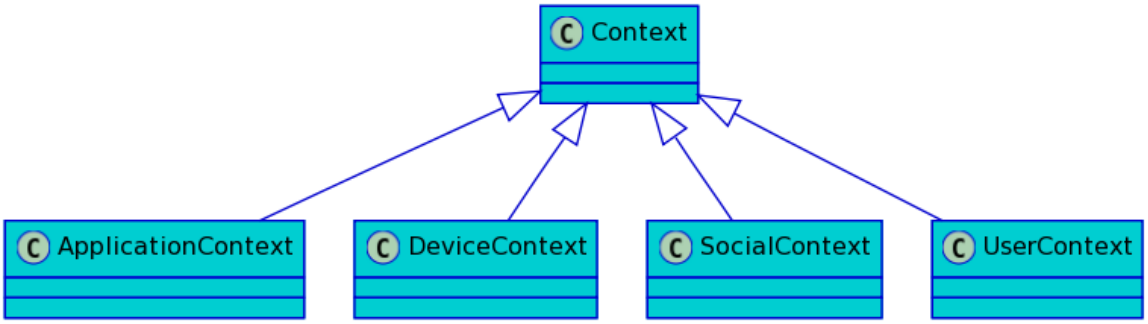


Which entity is the context for?

In literature and in technical implementations there are many different types of contexts mentioned. In general, it can be observed that context means something different for the researchers and the application developers. The naming of the context depends quite much on the entity or actor the context is intended for. When sharing, debating, and reasoning about context, because of its dependency on time, people often discuss how context changes over time. This means that in webinos the naming of contexts will reflect the actor/entity that is involved in the situation as the subject:

- Contexts that describe aspects of a user's situations are referred to as user contexts.
- Contexts that describe aspects of an application's situations are referred to as application contexts.
- Contexts that describe aspects of a device's situations are referred to as device contexts.
- Contexts that describesaspects of a user's social situations are referred to as social contexts. A social context is for this reason considered to be sub-part of a user context.

In webinos, these four kinds of contexts have been referred to in the user stories, use cases, and requirements. In terms of inheritance diagram, these four identified types of contexts can be depicted in UML as:



Please note that a social context is part of a user context, but that we have not depicted this in this inheritance diagram. We will come back to this composition relation (i.e. a part-of relation) in another UML diagram below.

Context representation in webinos

Adressing the needs of each application and the platform

Since the purpose is to enable any application to be context-aware, there will be a wide variety of needs for representing context information in webinos. Some of these needs will be individual for each application. The context-awareness support therefore would need to cater for all these demands as means to be sufficiently generic.

This is because context information that is relevant for one application will most likely not be relevant for another application. Secondly, context information that is individually captured, created and maintained by one application will need data integrity, and should/cannot not be taken out of context or modified by another application.

The exception to this is when the context information originates from the webinos platform itself - for instance from a location sensor. In such cases, since context information is generated by the underlying platform, and it can be personally sensitive, applications will need to negotiate their consent with the user to access this kind of context information. Yet when they are authorized (by the user) to access and use such context data, they should be readily available through the platform mechanisms. The latter is one of the basic objectives of the webinos context framework.

Structured vs. unstructured context information

Having analysed multiple system approaches to modelling and representing context, we have identified two main approaches for the representation of context:

1. Structured context information
2. Unstructured context information

Unstructured context information can be thought of as a collection of attributes or properties that have not been classified or categorised in a data structure. For example the phrases "in the city, tomorrow, meeting, Helen, Alice" is considered unstructured context information. Text queries to a Web search engine can be viewed as unstructured context information. This approach is very suitable when it is difficult to define a context structure due to the absence of data structures.

On the other hand, a structured approach to representing context information reflects a view that it is desirable to have some consistency and stability in the representation, categorisation, and expression of attributes and relations. Context information has been represented as tree structures, property lists, arrays, semantic networks, graphs and so on. These context structures can either be static or dynamic. A light weight way of representing structured context information is the use of a list of property-value pairs. With property-value pairs, we mean the same as attribute-value pairs.

Here are two examples of such a lightweight representations of context information:

1. "color=pink, interior=light-beige, buyer=Peter, car=BMW".
2. "location=home, orientation=vertical, device=Experia Neo, time=yesterday".

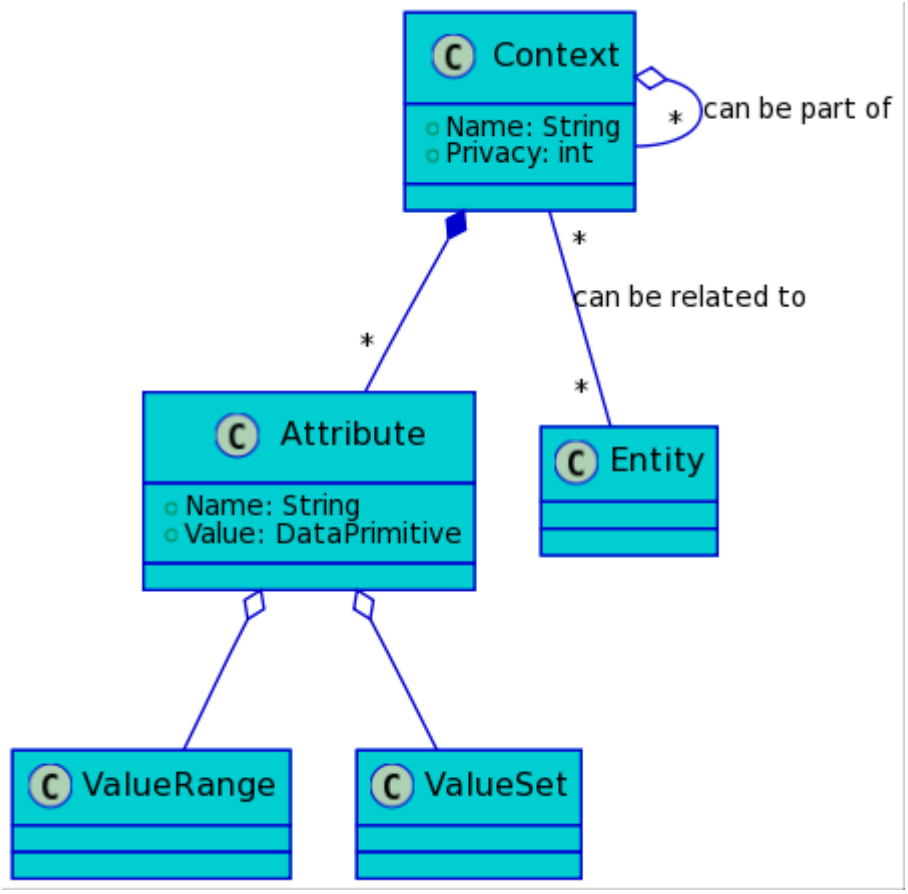
In webinos, we have chosen to represent context as a hierarchical tree-based representation of situations. The first reason for this is that tree-structures can be mathematically proven to be a list of elements. The second rationale is that unstructured information can always be represented in a structure. With this is we gain the possibility to categorise and sub-categorise the context information where unstructured information is just categorised in one category. This gives the developer community the opportunity to better organise and group context information together, and to also handle unstructured context information if needed.

The context strucure in webinos

The class diagram below shows core of all webinos context structures. A context has attributes with values. One can also add relations with other entities.
A context has:

- Name
- Privacy
- Attributes

A context can consist of sub-contexts to better group attributes together, although this would not be needed if there are few context attributes. The below class diagram for context information can described with: context can consist of several sub-contexts, and a context can be part of another context. A context will always comprise one or more attributes.



Each context attribute has a name and a value. The type of the context attribute is a data primitive (e.g. Boolean, integer, float, double, or string). In many cases it will be sufficient to just use string as the data type for a context attribute.

However, it is not always needed to have attributes at the top-levels of the context if the top-level context consists of several sub-contexts. In this case, the attributes of the sub-contexts qualify the top-level context to be a context in its own right. This approach to organise contexts information allows for making meaningful contexts without including attributes at the top-level context - as long as there are attributes at the bottom level contexts in the structure.

The benefit of this is that chunks of context attributes can be copied from one context instance to another. For example, if a device context contains a location attribute, one can copy this entire attribute

for in the user context structure. This makes the reuse of context information across context instances effective.

Attributes can have value sets and ranges

In some cases it is desirable to constrain the possible values for specific attributes. It is desirable on these cases to achieve this through the use of finite value sets or ranges. For instnace, an attribute with name="Capitol", type=string, could be constrained to the values = {"London", "Edinburgh", "Berlin", "Rome", "Madrid", "Istanbul", "Paris", "Athens", "Warsaw"}.

Here are some examples of value ranges:

- [10, 1000], (integer)
- [0.1, 0.12>, (float)
- <-1.0, 1.0>, (complex)
- [false, true], (Boolean)
- [June 1 2010, Aug 31 2010], (date)
- [*], (integer)

Here are some examples of value sets:

- {"Cyan", "Magenta", "Yellow"}, (string)
- {"None", "Little", "Medium", "High"}, (string)
- {"A2", "A3", "A4", "A5"}, (string)
- {10, 15, 20, 25}, (integer)
- {false, true}, (Boolean)
- {Jun 1 2012, Feb 1 2012}, (date)
- {*}, (string)

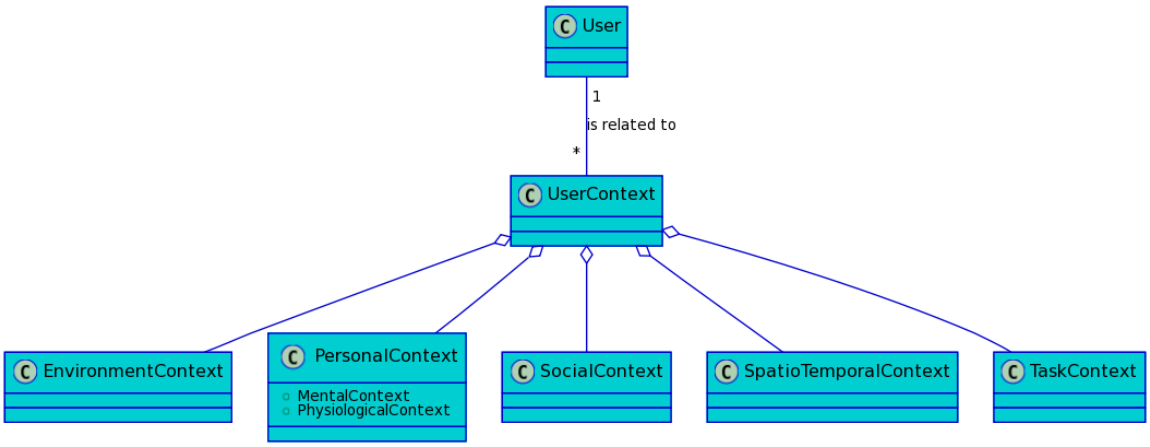
The data primitive of the value (i.e. type) are indicated in brackets.

User context in webinos

The following user context model describes aspects of a situation seen from the user's point of view (Myrhaug and Goker, 2003). This is proposed as a framework for exploiting user contexts within and across application domains. We believe this to be a comprehensive model that can be used with a wide range of applications involving user contexts. The structure is designed to enable effective matching and retrieval of contexts. The user context structure consists of five sub-contexts:

- Environment context
- Personal context
- Task context
- Social context
- Spatio-temporal context

Each sub-context should be considered as a container where an application developer can insert the important context attributes with values and types needed in the particular context-aware application, see the UML class diagram below that shows the various parts of the user context model:



Environment context

This part of the user context captures the entities that surround the user. These entities may be (but are not limited to) things, services, temperature, light, humidity, noise, and persons. Information (e.g. text, images, movies, sounds) that is accessed by the user can be linked to the environment context. The various networks that are in the surrounding area can also be described in the user's environment context.

Personal context

This part of the user context consists of two sub contexts: the physiological context and the mental context. The physiological context contains information such as pulse, blood pressure, weight, glucose level, retinal pattern, and hair colour. The mental context contains information like mood, interests, expertise, angriness, stress, etc.

Task context

This context describes what people are doing. The task context can be described with explicit goals, tasks, actions, activities, or events. Note that this can also include other persons' tasks that are within the situation. For example, considering a car with a driver and passengers, the situation can include the driver driving the car, passengers doing various activities such as reading, watching the car TV, and listening to music on the personal stereo. The task context of the driver and the passengers will be different.

Social context

This context describes the social aspects of the current user context. It can contain information about friends, neighbours, co-workers, relatives, and their presence. One important aspect in a social context is the role that the user plays in the context. A role can for instance be described with a name, the user's status in this role, and connections to the tasks in the task context. A role can, in addition, be played in a social arena. A social arena can have a name like "at work".

Spatio-temporal context

This context aspect describes aspects of the user context relating to the time and spatial location for the user context. It may contain attributes for time, location, direction, speed, shape (of objects/buildings/terrain), track, place, clothes of the user and so on (i.e. the physical extension of the environment and the things in it) .

Validation of the user context model

Below, we relate others' work on context models within the model above.

Environment context captures the entities around the user. It includes objects of the surrounding environment (e.g. buildings, outdoor facilities, infrastructure) and their state (i.e. temperature, light, humidity, noise). It also describes information – what Lucas (2001) calls information context – as part of the environment. Furthermore, devices that reside in the users' vicinity are also accounted for as part of the environment. Attributes of these devices define the extent with which personalisation can be performed (Chalmers and Sloman, 1999). Examples include the processing power, screen size/resolution, colour support, sound capabilities and the kind and number of input devices. Similar to Goker and Myrhaug, (2002), Schmidt, Biegl, and Gellersen (1999) also categorises contextual information about device(s) as part of the physical environment.

Personal context is equal to the user information that is stored in a typical user model (as also discussed later in following section). It can be distinguished into user's physiological context (e.g. age or body weight) and user's mental context (e.g. interest). This is similarly described in Schmidt, Biegl, and Gellersen (1999), which states "information on the user comprises for instance knowledge of habits, mental state or physiological characteristics". Attributes that are represented by the mental context can generally be found in user models. Examples include user's identity, preferences, knowledge and skills as listed in Reichenbacher's context model (2007). User's interest is one of the most important attributes that is commonly modelled in most user models and has been widely applied in personalised information systems (Brusilovsky, 1996).

Task context contains information about what the user is doing or aiming for. It describes "the functional relationship of the user with other people and objects" (Bradley and Dunlop, 2004), including the benefits and constraints of this relationship. It can be modelled as explicit goals, actions and activities. User's activity is a common type of context in a number of context models such as the one described in Reichenbacher (2007) as well as Chalmers and Sloman (1999).

Social context represents the social environment of a user. It may describe users' relationships to like-minded people that are connected to the user. Social filtering systems implement one special kind of social context modelling (Griffith and O'Riordan, 2000). The recommendation output is solely based on a user being similar to other users who rated items previously. This social connection is then exploited to recommend more items. Social filtering systems have demonstrated good results for a range of relevant topics such as music (Shardanand and Maes, 1995) and news (Resnick et al., 1994). Social context may model a user's list of friends or colleagues – perhaps explicitly expressed by that user or implicitly acquired through an email address book or buddy list on a website.

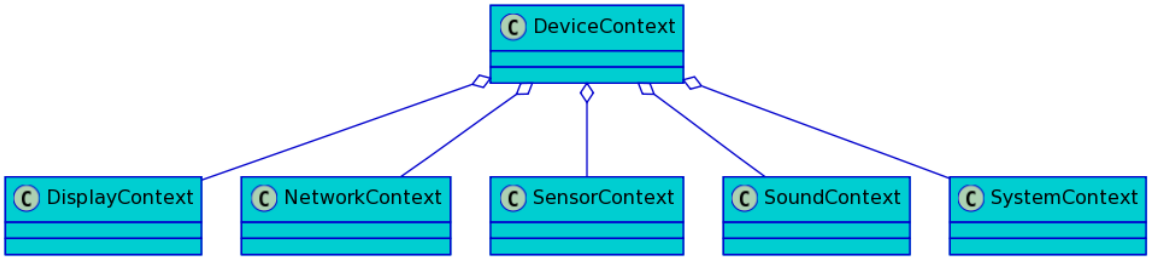
Spatio-temporal context represents physical space and time. The spatial aspect represents physical space. Location is the most common aspect of a spatial context. Mobile guides such as Cyberguide (Abowd et al., 1997), GUIDE (Cheverst et al., 2000) and the CRUMPET system (Zipf, 2002) belong to a special class of applications that are commonly referred to as location-based services. Based on its relevance, location modelling has emerged as one research branch in location-based services; a comprehensive overview is provided in Jiang and Yao (2006). Many location-based services employ location for the personalisation of geographic maps. Location can be represented either geographically or semantically as described in Beigl (2002). The geographic representation exhibits locations by its position (i.e. coordinates provided from the Global Positioning System (GPS)). On the other hand, the semantic representation describes locations in a more descriptive and humanly understandable way, yet still able to be processed by a computer. The comMotion system described in Marmasse and Schmandt (2000) for example learned meaningful locations semantically by analysing users' GPS logs over time. Besides location, spatial context also models attributes such as the direction of movement, the viewing direction and the speed of movement.

The temporal aspect, of the spatio-temporal sub-context, refers to time and can be represented as an absolute measurement or in a more relative manner (e.g. 'in the evening' or 'before a meeting'). In Hull et al. (1997), time is part of the users' environment. Similarly, Reichenbacher (2007) views it together with location as part of a situation. In Schmidt, Biegl, and Gellersen (1999), temporal context is related to all other context attributes representing the contextual change of those attributes over time. In Bradley and Dunlop (2004), temporal context is described as being "embedded within everything, and is what gives a current situation meaning...".

Overall, although there is some empirical work (Bierig and Goker, 2006) that investigates more closely a context model and its attributes, there is generally a lack of empirical work for investigating the relation between various context attributes.

Device context in webinos

The following device context model describes aspects of a situation seen from the device's point of view. This is proposed as a framework for using device contexts within and across webinos applications. As with the user context, we also believe this to be a fairly comprehensive model that can be used with a wide range of applications involving device contexts. It is anticipated that most, if not all, device context information can be sub-categorised and stored within this generic structure. The device context structure consists of five sub-contexts:



Sensor Context

A sensor context contains context attributes where the values of the context attributes originates from sensors connected to the device. It describes the sensor aspects of the device. It can contain information originating from any sensor on the device. Some examples of sensors may result in the storage of context data are: light, camera, motion, location, touch, temperature, biometric, microphone, keyboard, mouse, and near-field communication. One important aspect for the sensor context it therefore to be able to make sensors update contexts.

Display Context

The display context describes aspects of the display(s) connected to the device. The context information may be (but are not limited to) describing the screen, the type, the identifier, the resolution, the color depth and format, if it is a 3D enabled display, its standby state, the application being displayed, its luominosity and so on. One could also store attributes data about a as well as the shape of the display shape, text and line drawing capabilities.

Sound Context

A sound context is meant for capturing and storing context information about the sound context of the device. This can include information about the available loud speakers, the headset, the system volume, the supported sound qualities such as mono, stereo, surround, and environmental sound. It could be if the sound is muted, what the balance between the loud speakers is set to, the various acoustic levels in the surroundings, whether there is a microphone available or not, the type of microphone connected, the sound card that is installed and so on.

System Context

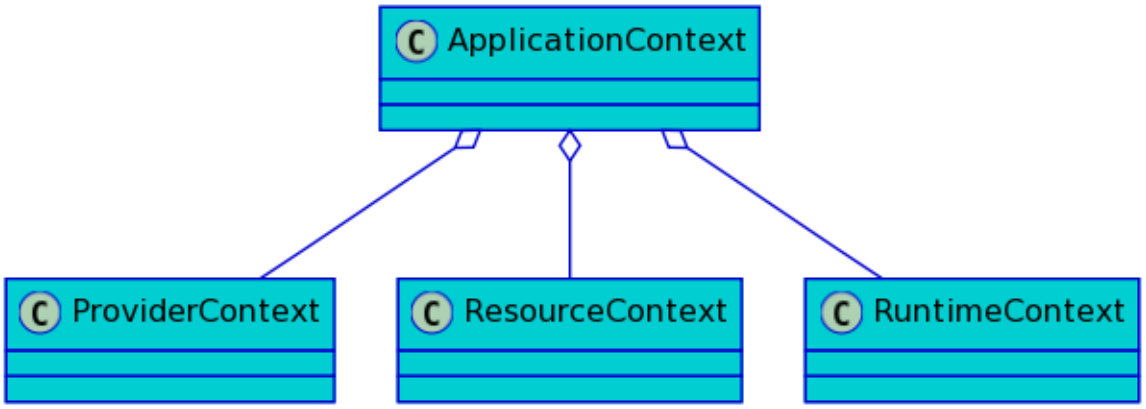
This part of the device context captures the local system capabilities of the device. The context information categorised within this can be (but not limited to) processor, work load, identifier, name, model, operating system, webinos runtime. Power consumption, battery, battery status, hardware devices, multi-media codecs available, mime types supported, the supported HTML version, the supported CSS version, available applications, available displays, available braille devices, printers, USB disks, hard drives, flash drives, database drivers, filesystem, cloud system folders and so on.

Network Context

A network context describes aspects of the various networks that the device either currently is conntected to - or the network bearer capabilities assumed available on the local device. For applications using network context, one could for instance foresee the likelihood of capturing various network topologies, the bandwidth of a network, measures of physical distance between network nodes, measures of latency between network nodes, degrees of trust for network nodes etc. Furthermode, the various attributes could be described for WLAN, Bluetooth, USB, GSM, Ethernet, Firewire, children. Nearby/ connected devices could be added to the network context, such as mobile phones, printers, displays, payment terminals, laptops, car infotainment system and so on. This, a network context is related to Quality of Service (QoS) aspects of the network and node connectivity.

Application context model in webinos

The following application context model describes aspects of a situation seen from the application's point of view. This is proposed as a framework for using application contexts within and across webinos applications. As with the device context, we also believe this to be a fairly comprehensive model that can be used with a wide range of applications involving application contexts. We believe that there is a potentially large amount of context information can be sub-categorised and kept within this structure. The application context consists of the following four sub-contexts:



ProviderContext

The provider category contains information that is related to the provision of applications data and content. It can be information about the provider of the application, such as the name, the authority, the access period, if the application is enabled, and so on. It can also contain information about what type of content/data that the application is capable of offering to other applications including the webinos platform. This could be sound, images, contacts, business cards etc. For instance, if two applications on the same device would like to provide contacts, because the user is creating contacts within each application, there is the issue of negotiating permission with the runtime to be able share this with other applications in a standardised way. Secondly, if an application would like to make some of its data public, one could alternatively specify new attributes and make them public.

ResourceContext

This context describes aspects of the resources available for the application. It can contain information about the originating app store, raw data, layouts, menus, preferences, settings, storages, XML files, and so on. With resources we mean the resouces available for the application, whether it is downloadable or hosted.

RuntimeContext

Context information relating to various runtime components could be included in this category. This could be context data about activities, widgets, activities, services, servers, etc. Furthermore, one could include information about the state of the runtime, the required runtime for an application to work, the required libraries, the minimum libraries that has to be present, the minim html version that is needed and so on.

Context API

The context API constitutes a layer that enables applications to access the underlying volume of contextual data in a uniform way. It provided two APIs that allow applications two different modes of accessing context data:

- Query API, enabling applications to perform queries for particular context data in the storage.
- Change Subscription API, enabling application to subscribe for specific events that bring about a change in context data and be notified in real time when such events happen.

The following code shows the two APIs which are defined, along with the rest of the webinos APIs, in deliverable 3.2:

```
interface ContextManager {
    void executeQuery(in QuerySuccessCallback successCallback,
                      in QueryErrorCallback? errorCallback,
                      in Query query)
         raises(ContextError);

    void subscribeContextEvent(in SubscribeSuccessCallback subscribeSuccessCallback,
                      in SubscribeErrorCallback? subscribeErrorCallback,
                      in OccuringEvent eventHandler,
                      in DOMString eventIdentifier)
         raises(ContextError);
}
```

Context Storage

The Context Storage component deals with storing and returning contextual data. Based on the diversity of devices and operating systems which are to be supported by the webinos platform, pinpointing a specific storage technology and bringing it to all devices/platforms is nearly impossible. This is why, the component operates a connector model in order to make the storage operations independent from the underlying database and operating system. As already mentioned, the design decision is made in support of system portability and flexibility. By providing additional connector implementations, the Context Storage component can connect to various types of database technologies (e.g. relational databases, graph databases, triple stores, etc.). Connectors can be created by the third party webinos developers and should be deployable at runtime.

```
Interface ContextStorage{
    StoreConnector connect(in string connectorID);
    StoreConnector connect(in string connectorID, in object properties);
    object getConnectors();
};

Interface StoreConnector{
    boolean execute(in string statement);
    object executeQuery(in string query);
    int executeUpdate(in string update);
    object getStatus();
    void close();
};
```

Store connectors can be implemented for traditional relational database management systems (RDBMS) such as SQLite or MySQL. On the other hand, this approach also supports but does not enforces the use of storage mechanisms optimized for context-data persistence. Graph databases for example are a type of NoSQL database. The graph database approach differs from RDBMSs as it uses oriented graph structures to store data rather than in tables. All information is represented by means of the graph's nodes, edges, and properties. Nodes are used to represent entities, whilst properties can be added to provide additional information regarding an entity. In turn, the edges define node-to-node and node-to-property connections and thus represent a certain relationship between the two connected items.

The pros and cons of such an approach are:
+ High flexibility by allowing nodes with dynamic properties to be linked arbitrarily to other nodes.
+ High scalability of NOSQL databases.
- Lower efficiency in batch processing compared to RDBMSs.

Name	Platforms	License	URL
Neo4j	Java	Dual: GPLv3 and AGPLv3 / Commercial	http://neo4j.org/
FlockDB	Java	Apache	http://github.com/twitter/flockdb/

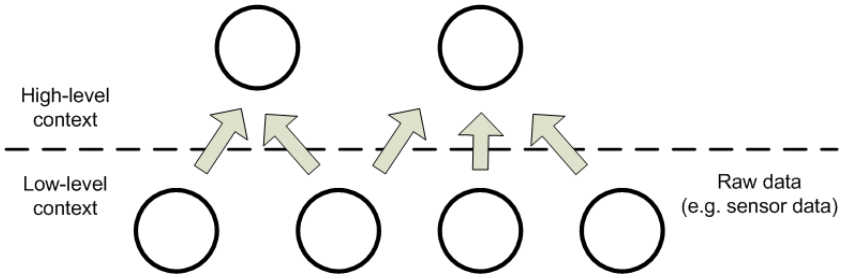
Another example of database systems optimized for context storage are triple stores. As with graph databases, triple stores also rely on graph structures for storing data. In particular, triple stores provide an optimized mechanism for the persistent storage of RDF triples. Compared to graph databases, where focus are mainly the characteristics of the graph (e.g. distances, reachability, etc.), triple stores mainly aim for optimized query processing and knowledge inference. The built-in support for semantics and formal RDF inference rules provides better means to extract new triples. The main language for performing RDF queries is SPARQL (Simple Protocol and RDF Query Language). SPARQL is standardized by the W3C RDF Data Access Working Group (DAWG). It enables flexible queries consisting of triple patterns, conjunction/disjunction patterns, as well as optional patterns.

The pros and cons of a triple store approach are:
+ Both data format and query language are standardized.
+ Unlike RDBMSs, triple stores are optimized for intensive use of query and insertion operations.
- The development of triple stores is still in its inital phase. Lots of triple stores are built on top of traditinoal RDBMSs, possibly introducing performance issues.

Name	Platforms	License	URL
Jena SDB	Java	BSD	http://openjena.org/
AllegroGraph	Windows, Mac OSX, Linux, FreeBSD, Solaris	Commercial	http://www.franz.com/agraph/allegrograph/

Context Reasoning

Context reasoning is needed in order to infer new facts from instance data and class descriptions in the Context Storage. The reasoners are thus the objects that perform the task of deriving additional information. The reasoning process allows the system to derive higher-level context data, by combining lower-level knowledge originating from device sensors, or applications. The reasoning component is pluggable, supporting the addition of specialized reasoning capabilities that are optimized for a specific task and/or environment.



First of all, there is often a need for sensor data fusion. This action aims at integrating different context sources in order to make the available knowledge more reliable. A well known example is the integration of location-aware sources. A user's location can be obtained from various sources: GPS positioning, cell tower triangulation, IP geolocation lookup, etc. All these source have varying accuracies, ranging from a few meters to several kilometers. Especially when users own multiple devices, sensor data fusion can be used to further enhance the precision of contextual data such as location. Context reasoning is a challenging task, as there must be a mechanism in support of mapping lower-level context data to higher-level knowledge. Context reasoners in general rely on two approaches for mapping different levels of context data: the use of ontologies (e.g. OWL), and the use of probabilistic reasoning to produce probabilistic models.

```
Interface Reasoner {
    Reasoner bindOntology(in object axioms);
    void run();
    object getStatus();
    object getInferenceResults();
    void setProperty(in string key, in object value);
};
```

A number of good RDF/OWL reasoners are already available. For example Pellet, an advanced OWL reasoner. The Jena Reasoner on the other hand provides an extensible, which allows a wide range of inference engines/reasoners to be plugged into the Jena platform.

Name	Platforms	License	URL
Jena Inference	Java	BSD	http://openjena.org/
Pellet	Java	AGPLv3	http://www.franz.com/agraph/allegrograph/

Such a reasoner approach could be easily implemented on top of the basic mechanisms of the Context Framework in webinos, i.e. the context acquisition, the access APIs and the context storage.

Dependencies on other components

The webinos Context Framework presents interdependencies with practically every other area in the system. This is primarily due to the nature of context, i.e. it needs to be aware of what is "happening" in the system and acquire/provide access to relevant context data.

Specifically two broad lines of interdependencies can be identified:

- Monitoring the operation of several areas (Discovery, Authentication, Messaging, Analytics, API calls, etc) in order to identify context related events, track their occurrence and acquire the necessary context data for describing the corresponding situation.
- Adopting the implemented Privacy framework in order to ensure a privacy preserving operation of the entire Context framework, meaning allowing the user to enable or disable tracking of context data, enforcing applications to inform the user about their intention to access and use his context data, empowering the user to allow or not applications to access his context data.

Security

In webinos the concept of security is extended in two directions: resources access control and privacy protection.

As reported in the preceding sections one of the most widely used and flexible solutions is represented by XACML (OASISXACML). The XACML specification defines an XML-based language and an architecture for the expression and evaluation of access control policies but doesn't provide neither privacy protection nor the specification and evaluation of credential restrictions in the policies.
With the PrimeLife extension (PRIMELIFE) the privacy requirements can be fulfilled: the user has control of his personal data and can negotiate its disclosure with the access control system.

XACML architecture (Data flow)

XACML (eXtensible Access Control Markup Language) is an OASIS standard (OASISXACML) for access control systems that defines a language for the description of XML access control policies and an architecture to enforce access control decisions.

The XACML architecture depicted in the figure is composed of the following elements:

Access Requestor: the entity which requires the capability.

Policy Enforcement Point (PEP): the entity that performs access control, by making decision requests and enforcing authorization decisions. It also try to execute the Obligations and doesn't grant access if is unable to complete these actions.

Obligations: operations specified in a policy that should be performed by the PEP in conjunction with the enforcement of an authorization decision. These operations must be carried out before or after an access is granted.

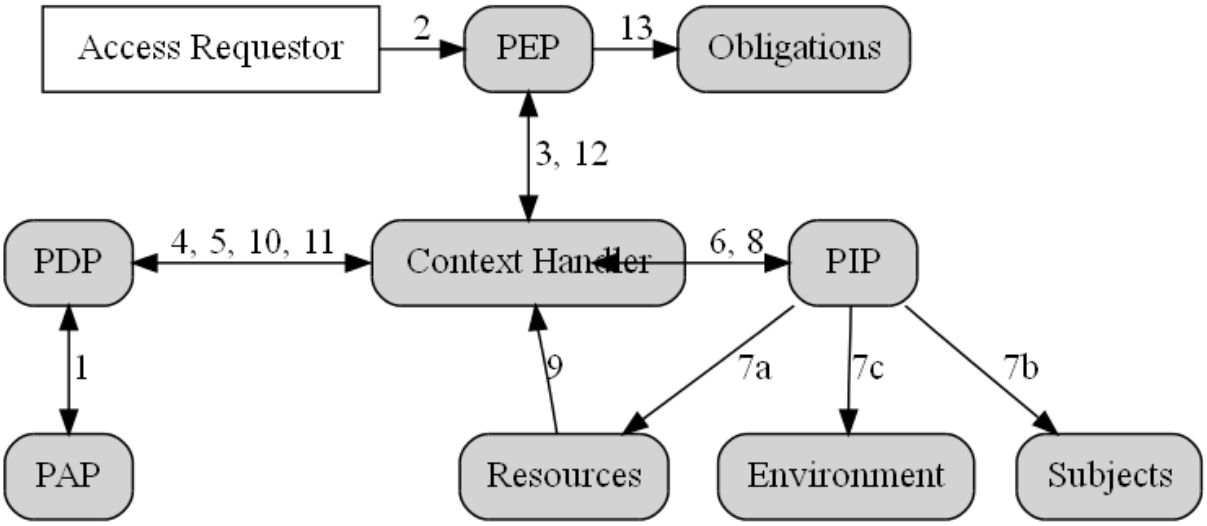
Policy Decision Point (PDP): the main decision point for the access requests. It collects all the necessary information from other actors and concludes an authorization decision.

Context Handler: the entity which sends a policy evaluation request to the PDP and manage context-based information.

Policy Information Point (PIP): the entity that acts as a source of attribute values that are retrieved from several internal or external parties like resources, subjects, environment and so on.

Policy Administration Point (PAP): the repository for the policies, it manages policies and provides them to the Policy Decision Point.

Resources / Subjects / Environment: parties that provide attributes to the PIP.



In a chronological order the data flow sequence is:

1. Policies and policy sets are defined in the PAP and made available to the PDP.
2. The Access Requestor sends a request to the PEP for the access to the specified resource.
3. The PEP translate the request in native format and sends it to the XACML Context Handler. This may include attributes of the subjects, resources, actions, and environment.
4. The Context Handler creates an XACML request context and sends a policy evaluation request to the PDP.
5. The PDP, in order to evaluate the policies, queries the Context Handler for attributes of the subject, resource, action, and environment.
6. The Context Handler obtains the attributes either from the request context created in step 4, or from the PIP.
7. The PIP collects attributes about subject, resource and environment.
8. The PIP returns the requested attributes to the Context Handler.
9. Optionally, the Context Handler includes the resource in the context.
10. The Context Handler returns to the PDP the requested attributes.
11. The PDP sends the response context (including the authorization decision) to the Context Handler.
12. The Context Handler sends to the PEP the response context in native response format.
13. The PEP tries to execute obligations.

XACML extended with PrimeLife for webinos (Data flow)

The architecture chosen to achieve the goals on security and privacy outlined by webinos, is an XACML distributed architecture integrated with the PrimeLife extension.

PrimeLife is a research project on privacy and identity management, funded by the European Commission from 2008 to 2011 (PRIMELIFE). The project proposes an extension to the standard XACML architecture allowing among other things the support of privacy data handling policies on systems with a resource access control based on XACML.

The main motivations behind the choice of the PrimeLife extension, other than the support of data-handling privacy policies are:

- Introduction of negotiation in the evaluation process: negotiation allows an incremental evaluation of access control policies and reduces the disclosure of personal data.
- Support for credential-based restrictions (digital credential and certified attributes)
- Legislation support (EU privacy directives)
- PrimeLife is an open-source European Project

The complete architecture, depicted below, contains new components that are PZH and PZP defined in the high level webinos architecture, Decision Wrapper, Access Manager, DHDF, Data Reader and Request Context derived from the PrimeLife extension and the PDPC (PDPCache) generally included in XACML's PDP component. Follows the description of these components:

Personal Zone Proxy (PZP): defined in the preceding sections, in this context can act in four ways

- enforces authorization decisions at device level
- synchronizes request context data with other devices
- synchronizes PDP policies with other devices
- verifies credentials towards the request context

Personal Zone Hub (PZH): defined in the preceding sections, in this context can act in three ways:

- as a data synchronizer towards the request context (via PZP)
- as a policy synchronizer towards the PDP / PDPC (via PZP)
- as a credential system (responsible for credential verification) towards the request context (via PZP)

Decision Wrapper: responsible for driving the access control policy evaluation and enforcement.

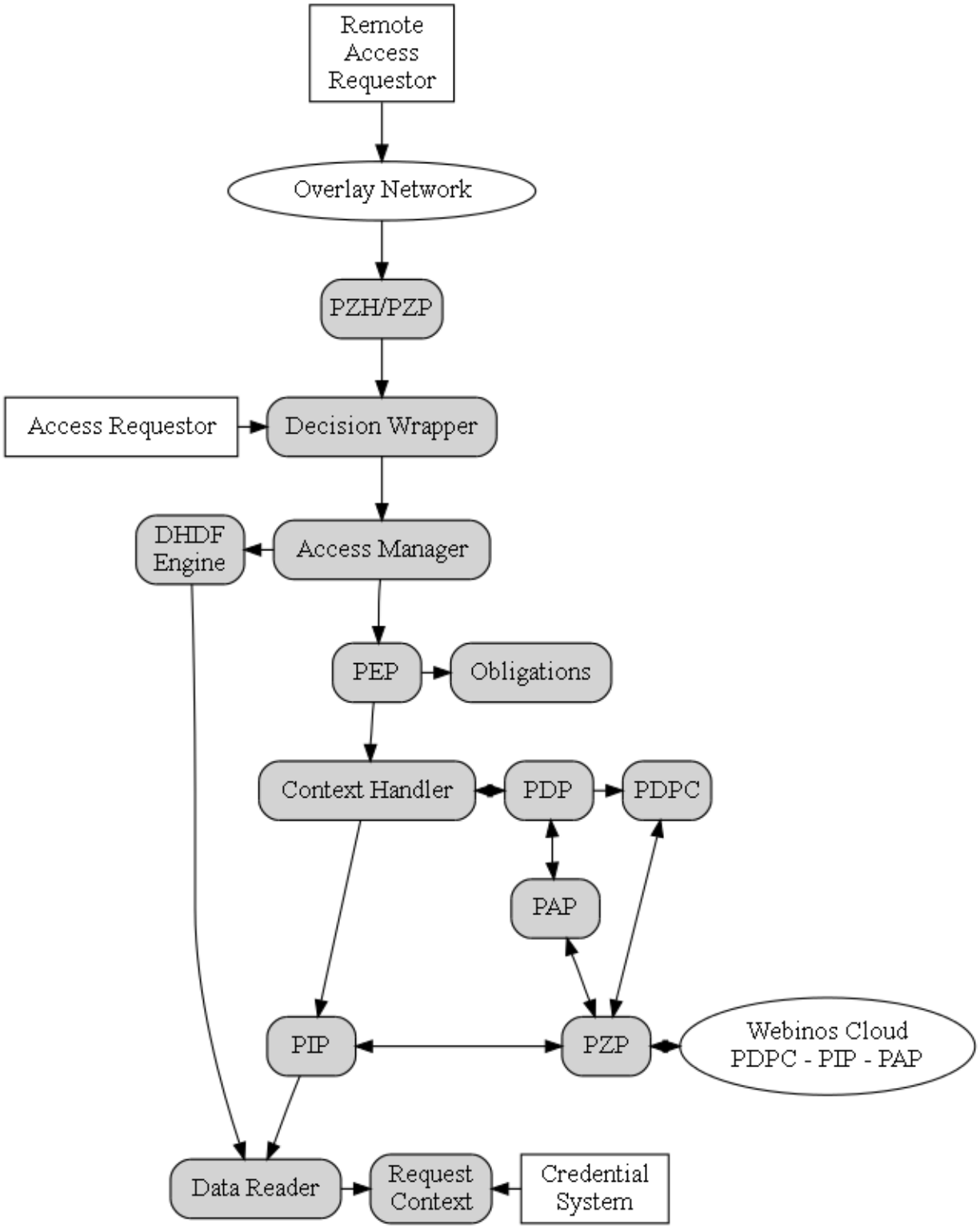
Access Manager: the entity in charge of taking the final decision by combining the XACML access. control and the DHDF data.

DHDF Engine: Data Handling Decision Function engine provides privacy and data handling functionalities.
It does not implement a complete privacy-aware access control system, but rather it is responsible for the management and evaluation of data handling policies only.

Data Reader: responsible for abstracting the communication with the Request Context.

Request Context: responsible for managing all contextual information; it stores all the data and credentials released by a user in a given session.

PDPC: the PDP cache, stores PDP decisions that could be share among personal devices.



The data flow sequence is exposed by the three scenarios presented in the "Formal Specification" paragraph.
Although Obligations are included in the architecture they will be handled in the phase 2.

More aspects on security and privacy will be pointed out in the 3.5 deliverable [D035].

Technical Use Cases

Here are depicted and briefly presented use-cases particularly relevant for policy sub-task.

WOS-UC-TA8-002: Interpreting policies and taking access control decisions

Normal Flow

Pre-Condition
The User has a tablet PC running webinos.
The User has a photo collection on his tablet PC which he would like to share with other people.
The webinos platform has a set of policies installed. These are not contradictory - a decision can always be taken based on their content.

Flow
1. The User has installed a new photo-sharing Application.
2. The photo-sharing Application automatically tries to upload his photos to a shared repository.
3. Accessing photos stored on the webinos Device - his tablet pc - results in an access-control request to the webinos platform.
4. webinos takes the event (Application X accessing File F) and checks against the set of policies (PS) installed.
5. The policies allow this action, and the photo-sharing app resumes uploading photos.

Post-Condition
The correct access control decision is taken with respect to all policies.

Alternate Flow (optional)

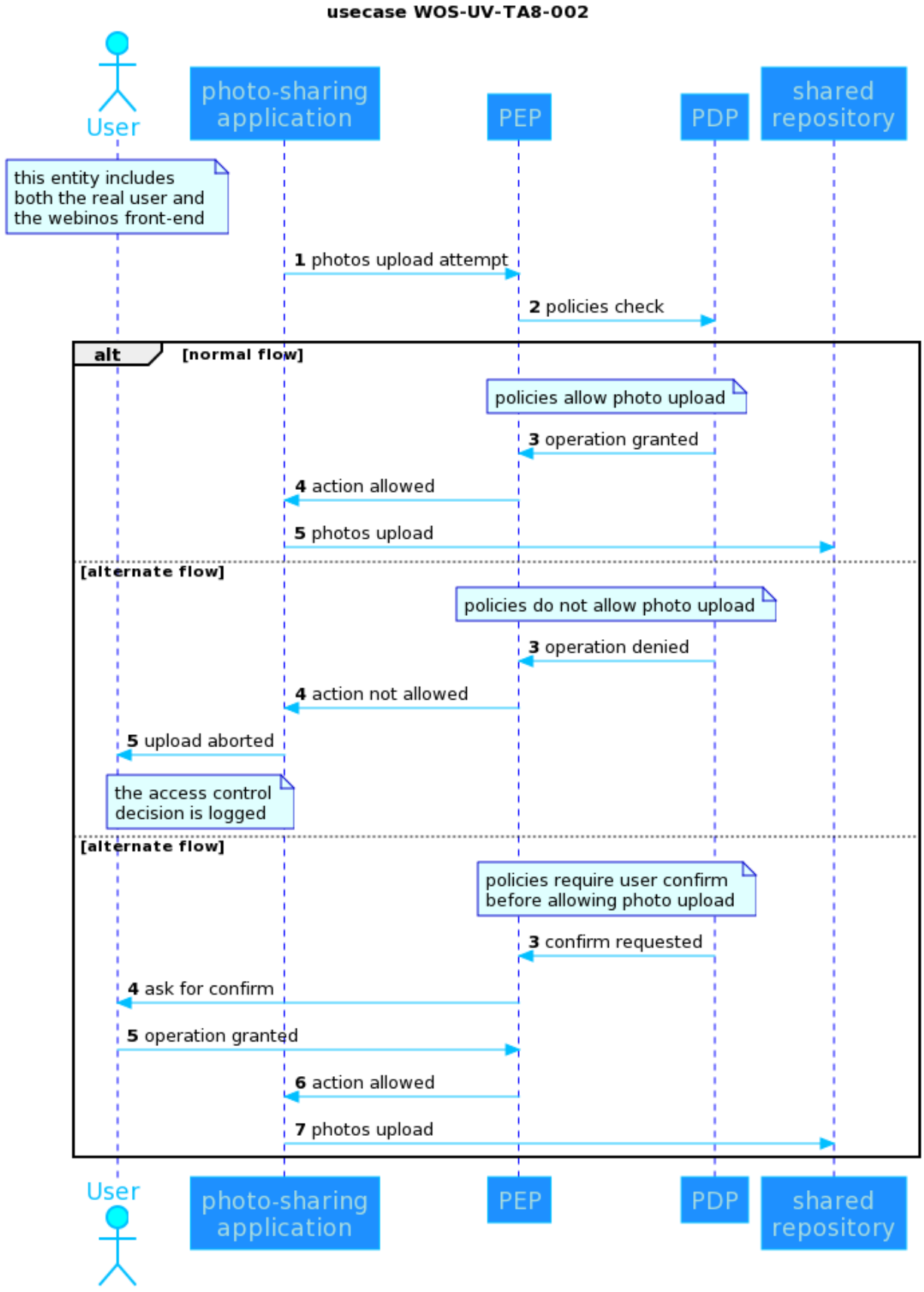
Pre-Condition
Same as normal flow.

Post-Condition
5. The policies do not allow this action and the Application is not able to upload any files.
6. The Application fails gracefully, informing the User of why it is unable to complete the request.
7. The access control decision is logged.

Alternate Flow (optional)
Pre-Condition
Same as normal flow.

Post-Condition
5. The policies require an extra condition before allowing this action: in this case, requesting User input to confirm that this behaviour is permitted.
6. webinos prompts the User to confirm that access to photos is allowed.
7. The User agrees.
8. The photo-sharing Application resumes uploading photos.

Sequence diagram analysis: WOS-UC-TA8-002 - Interpreting policies and taking access control decisions



WOS-UC-TA8-003: Enforcing multiple policies on multiple devices

Normal Flow

Flow

1. The User has a Mobile phone and an in-car entertainment System.
2. The User decides that he does not want to alert local networks to his presence.
3. He uses webinos to specify that his Devices should remain hidden and not discoverable. He does this on his Mobile phone.

Post-Condition

Any webinos Device used by the User subsequently will not be discoverable by a remote network. This includes his car whenever he is using it.

Alternate Flow

Pre-Condition

A User has both personal and work photos on his webinos Device.

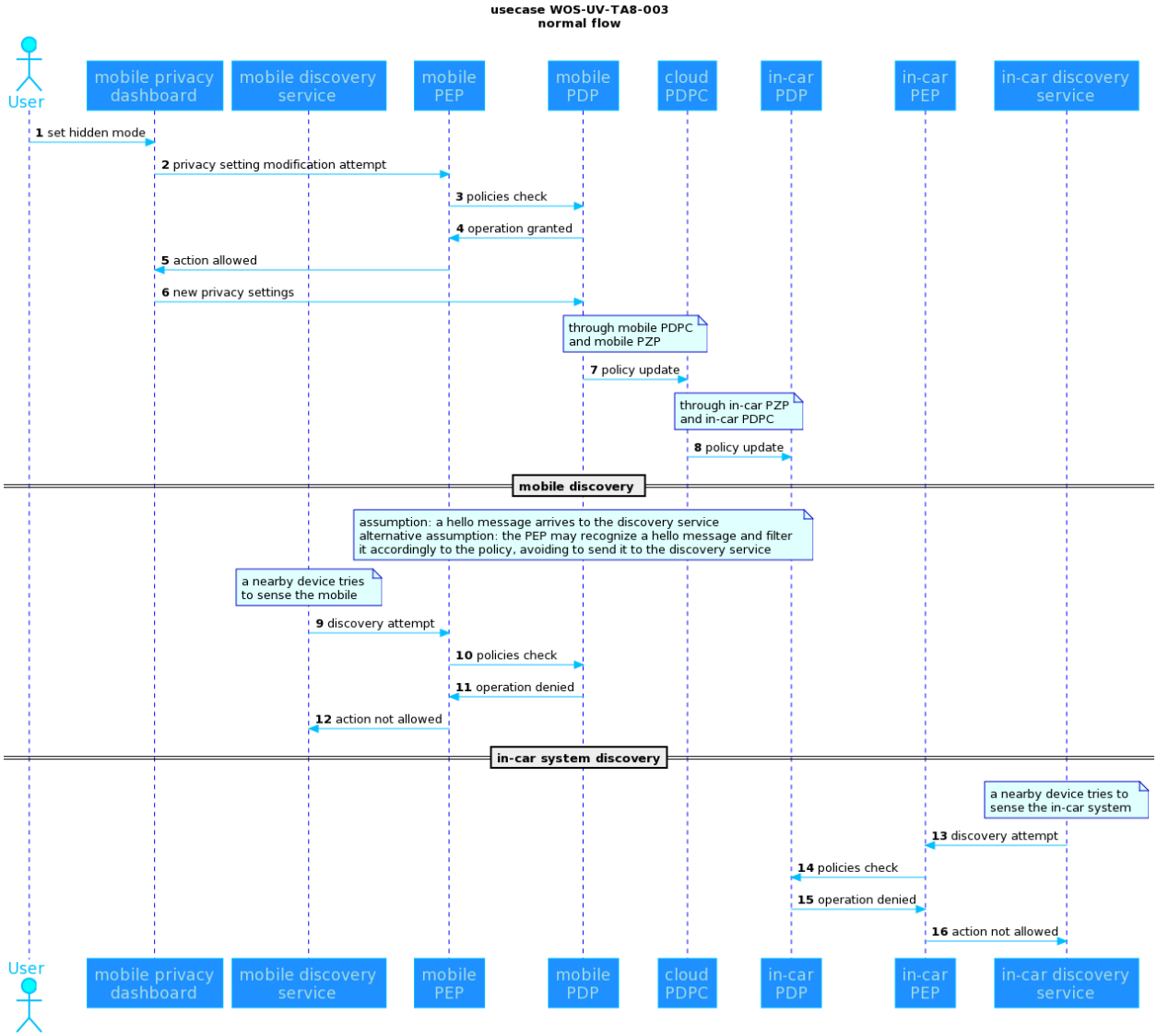
Flow

1. The User uses a Mobile phone provided by his company, a professional photography business.
2. He has configured it to allow him to share personal information with certain social networking Applications.
3. One of the Applications encourages the sharing of personal photos.
4. He does not see a problem with sharing the photos on his Device, and proceeds, allowing the potential upload of all photos on his Device.
5. However, the company policy protects all the photos owned by the company, many of which he has inadvertently given permission to. These are forbidden from being used by non-trusted Applications.
6. The webinos policy engine interprets the User's policy and the company policy - knowing that the company, as the phone's owner, takes precedence - and denies access to the relevant photos.

Post-Condition

The User can only share his own photos with his social network Application, not any of the company's copyrighted photos.

Sequence diagram analysis: WOS-UC-TA8-003 - Enforcing multiple policies on multiple devices (normal flow)



WOS-UC-TA8-007: Policy authoring tools

Normal Flow

Pre-Condition

The Developer's company has ownership of company Devices and is able to set policies which cannot be overridden by the End Users.

Flow

1. The Developer knows that webinos policies are written in a standard language and he downloads a webinos policy editing tools to help him create a suitable policy for his requirements.
2. He uses the tool to select the class of assets which she is concerned about - all data owned by the company.
3. He then sets a mandatory policy that this data is not allowed to be shared with any Application not signed by the company. All access to the data should also be logged.
4. He also selects a default policy for personal data, stating that it should not be shared with Applications by default.
5. The Developer modifies various other settings until he is happy with the policy.
6. The tool provides several examples of how his policy would be applied, and what it allows and denies access to.
7. He then tests his policy against several example Device configurations that the tool supports. This shows him that Devices using his policy would still be able to run, but would not be able to access company data.
8. He saves the policy and signs it to mark it as a trusted company policy.

Post-Condition

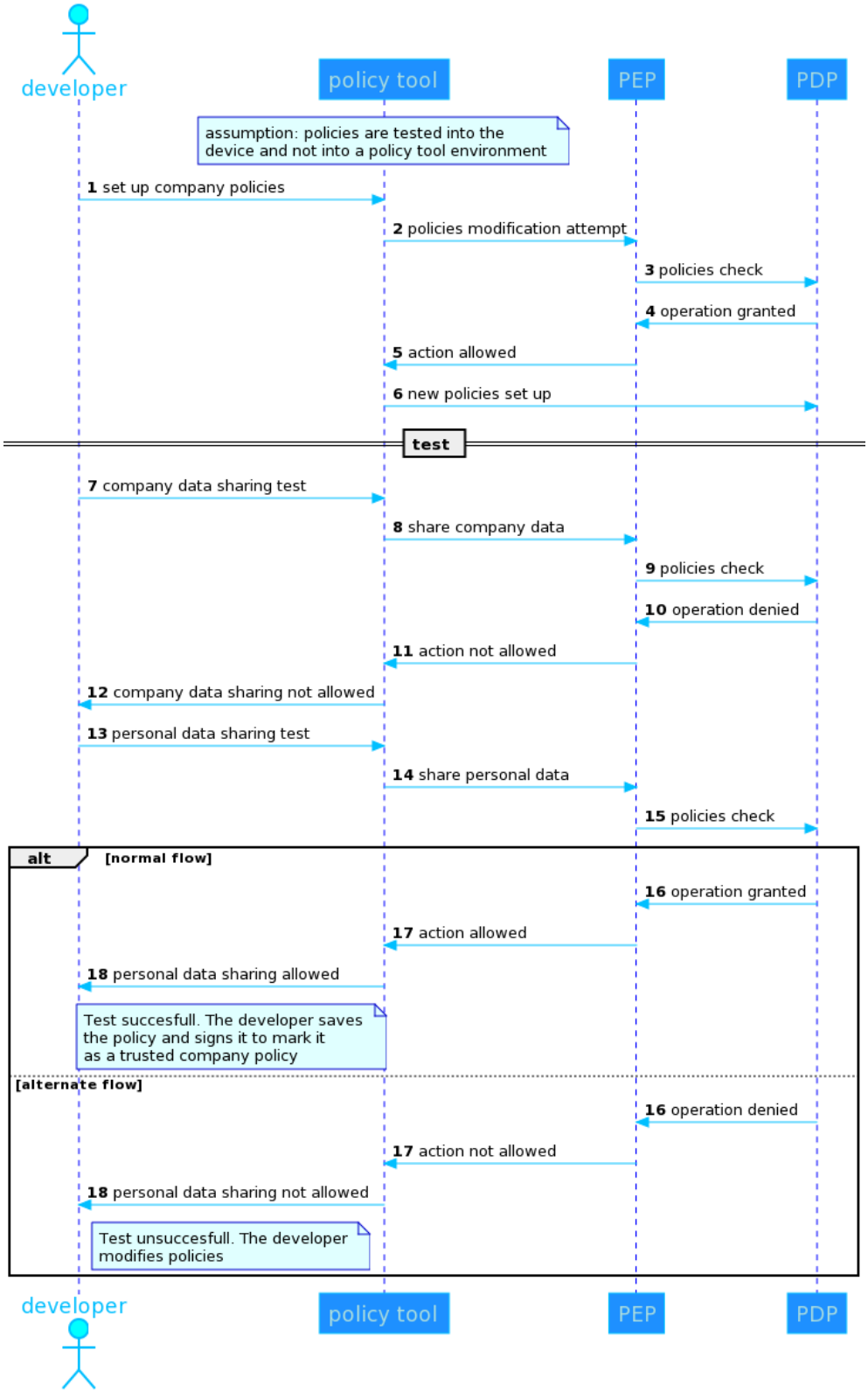
The company policy can be applied to all Devices the company owns and will control webinos Applications in the ways defined by the Developer.

Alternate Flow (optional)

Flow

7. The Developer tests the policy and realizes that it prevents Applications from accessing any data on the Devices. This is shown to him using the policy tool.
8. He realizes his mistake and modifies the policy suitably.

Sequence diagram analysis: WOS-UC-TA8-007 - Policy authoring tools



Here some further use-cases whose aim is to highlight policy interactions.

Cross-Device Policy Synchronisation

Installing an app ('A') on device 'D', granting it permissions, and then wanting to allow it to use device 'E':

- Assume device 'D' and 'E' are used only by the user Justin.
- Justin "installs" 'A' on 'D'.
- At end of install process, after granting the application some permissions, Justin is prompted "This application can be used on multiple devices. Install on [list of user devices and tick box]"
- Justin selects that the application should be installed on device 'E' as well.
- Prompt: "Give application permission to access the same resources? / Customise permissions / Ask me later?"
- Justin clicks "give same resources"
- Webinos runtime creates a message to device 'E' containing instructions for doing this.
  o Name of application, permissions granted on this device
  o Justin's credentials/signature
- Message queued to be sent on the overlay network to the other device, using a secure transport session. This may happen immediately if possible (e.g. if device 'D' is a home server) or may wait for a proximity event, or may be queued on a cloud service waiting for the user to interact with device 'D' in the future.

Using device 'D' to remove permission for app 'A' to access resource 'R' on all devices (Device 'D' and 'E'):

- Peter has been using app 'A' and decides he doesn't trust it after being told by a friend that it sends data to advertisers.
- He is using device 'D', as this is his main device
- He navigates to the policy management area (or maybe he right-clicks on the application icon) and selects "remove".
- He is asked whether he wants it to be removed across all of his devices (a list of his local devices might be selected) and he clicks on all of them.
- Webinos runtime generates a signed update message to each device in his cloud, including device 'E', in a secure manner:
  o "Remove app", app name
  o user credentials, device credentials
- These are sent when the device is able to connect to other devices and when these devices are capable of authenticating themselves
- Depending on the device security policies, the deletion may happen automatically, or the app might be isolated and require local permission to delete

Using device 'D' to authorise app 'A' to access resource 'P' on device 'E' (E and D owned by the same user) and Using device 'D' to authorise app 'A' to access resource 'P' on device 'E' (E and D owned by different users):

- Helen is using app 'A' and wants to use it to access her friend Gloria's photos on device 'D'.
- She selects to discover other devices, and finds Gloria's device 'D'
- She requests access to photos on 'D'.
- Gloria's device prompts Gloria to allow this and presents Alice's device details, as well as the history of communication with this device.
- Gloria approves the communication
- Gloria's runtime adds a temporary permission for the photo app (resource 'P') on Gloria's phone

Updating labels on stored user data - marking some as 'private' on device 'D' and having the same data on device 'E' automatically tagged:

- Gloria decides to make sure her personal data is being protected appropriately
- She looks at her webinos device to see what data about her is known
- Some data is marked as "private" and other data is not.
- She notes that her "place of work" is listed but not marked as private
- She tags this data as "private" and therefore all policies apply to it that applies to other private data.
- She tells webinos to update this setting on all her devices
- Webinos queues a signed policy update request for device 'E':
  o Update policy, data item, new XACML
  o User credentials, device credentials

Registering device 'F' as being a trusted device with respect to device 'D' and device 'E':

- Anna has just bought a new tablet PC ('F'), and would like it to automatically synchronise with other devices
- She uses it to discover her mobile phone ('D'), a webinos device.
- On discovery, she selects the mobile phone symbol and chooses to join the "local webinos device cloud"
- Untrusted message from tablet to phone:
  o "device join", device ID
- This prompts her mobile phone to authenticate her
- She enters her credentials into the phone
- The phone adds this device to a "known" and "personal devices" set
- The tablet PC downloads the relevant set of policies (as well as profile info - out of scope) from the mobile phone and applies them. This includes the identity of other devices in the cloud.

Policy Format

Policies are expressed in the simplified XACML format defined in "BONDI's Architecture & Security Requirements (Appendices) v1.1" (BONDIA&S) following the grammar provided by WAC (WACXMLSP). This format has been used as a basis in different specifications as W3C DAP (DAPWG) and WAC (WACDS).
In the following sections are described only the updates to the BONDI policy format:

- Some new attributes enhance the subject's information to fit webinos needs on distribution of policies between devices and cross device interaction.
- Two new attributes enhance the resource's information to identify services and the extensions that could be declared in an application manifest.

For further details and all points not covered here refer to (BONDIA&S).

IDs in policies

All IDs used in policies to match "subjects" and services (which are "resources") are URIs coming from the data described below.

Whenever an entity (user, device, application or service) is registered for the first time in a PZP/PZH, a record with some basic identity information is stored locally to the PZP/PZH.
These trusting records - mainly used to convert a generic ID in a handy URI (loosely coupling them) and to locate the trusted entity's certificates - should contain the following information:

- ID: according to the entity could be one of the following
    - User ID eg. webinos/facebbok/google ID
    - Device ID eg. MAC address, IMEI, HD serial number
    - Service/application ID eg. URI, URL
- Friendly Name: name that could be prompted in a dialog
- URI: generated on the fly the first time and the used to address the entity.
- Certificates: list of URL/paths to certificates
- Extensions: [For the phase2] for custom metadata and validation rules, e.g. checking IP addresses, MAC addresses, attestation data.

To synchronize these information PZPs/PZH can use JSON array. No format is defined to store locally these data.

JSON definition of a single trusting record:

```
{
  "id": "...",
  "friendly-name": "...",
  "uri": "...",
  "certificates": [
  ...
  ],
  "extensions": {
   ...
  }
}
```

Policy Subject

Given a policy, the policy subject defines the entity to which the policy will be applied. A policy may be applied to many subjects that are called target of the policy.

The subject form used is of the following type:

"User U can access Feature F of Device D through the application A running in a Device X"

"Device D" is also identified as target device whilst the "Device X" is identified as requestor device.

There are four basic information points in this subject:

1. WHO want to access a resource: could be an user (generic or not) or a group
2. WHERE the access requestor is: make possible to differentiate a local access (INTRA-DEVICE) from a remote access (INTRA-PZ or EXTRA-PZ)
3. WHERE the resource is: allow to use the same policy file for many devices
4. WHICH application is used: could be used to differentiate behaviour for application of different authors, distributors etc..

These information may be all present or not and in this case will be used the default values (referring to the INTRA-DEVICE scenario)

Pros:

- The user is represented explicitly.
- Policies can be copied to multiple devices without changes.
- Can represent complex scenarios.

Cons:

- Could be not so intuitive, but the user will manage policies by simple tools
- The policy is user-dependent, but default policies could be defined using some "generic users" (see User Info paragraph (User-Info)).

Subject Attributes

The subject attributes are divided in three groups:

1. Widget/Website Identity
2. Devices Info
3. User Info

The first group comes from the (BONDIA&S) and defines the properties used to identify the applications (widgets/websites). The other two groups regarding to the devices and users information are presented below

Devices Info

The Devices Info attributes express which device will apply the policies (the target) and which will try to access the features of the first one (the requestor).
If the target-id is absent then the target device will be the "current device" and in a similar way if the requestor-id is absent the requestor will be the "current device".
The "current device" is the device that enforces the policy.

Attribute	Type	Value
target-id	URI	ID of the device where the policy will be enforced. An ID could be a generic device URI (see list below) or a PZP generated one
target-domain	URI	Domain of the target device. Refer to the domain URIs listed below
requestor-id	URI	ID of the device requesting resources. An ID could be a generic device URI (see list below) or a PZP generated one
requestor-domain	URI	Domain of the requestor device. Refer to the domain URIs listed below
webinos-enabled	Boolean	True if requestor device is webinos enabled. If this attribute is not specified doesn't matter if the device is webinos enabled or not

Generic device URIs:

- http://www.webinos.org/subject/device-info/id/device (default)
- http://www.webinos.org/subject/device-info/id/pz-device (trusted device)
- http://www.webinos.org/subject/device-info/id/pz-shared-device (trusted device)
- http://www.webinos.org/subject/info/id/trusted (any trusted device)

- http://www.webinos.org/subject/info/id/not-trusted
- http://www.webinos.org/subject/info/id/any

Domain URIs:

- http://www.webinos.org/subject/device-info/domain/automotive
- http://www.webinos.org/subject/device-info/domain/desktop
- http://www.webinos.org/subject/device-info/domain/home-media
- http://www.webinos.org/subject/device-info/domain/mobile

http://www.webinos.org/subject/device-info/domain/ is the default domain URI value and represents all domains.

User Info

The User Info attributes allow to define the user to which a policy is referred. If the user-id is absent then the subject refers to any user..

Attribute	Type	Value
user-id	URI	ID of the user to which the policy will be applied. An ID could be a generic user URI (see list below) or a PZP generated one
user-key-cn	String	The common name of the certificate for the user signature
user-key-fingerprint	String	The fingerprint of the certificate for the user signature
user-key-root-cn	String	The common name of the root certificate for the user signature
user-key-root-fingerprint	String	The fingerprint of the root certificate for the user signature

Generic user URIs:

- http://www.webinos.org/subject/user-info/id/pz-owner (trusted user)
- http://www.webinos.org/subject/user-info/id/pz-member (trusted user)
- http://www.webinos.org/subject/user-info/id/trusted-by-pz-owner
- http://www.webinos.org/subject/user-info/id/trusted-by-pz-member

- http://www.webinos.org/subject/info/id/trusted (user trusted by PZ-owner and any PZ-member)
- http://www.webinos.org/subject/info/id/not-trusted
- http://www.webinos.org/subject/user-info/id/anonymous

Resource Attributes

In order to control the utilization of third party developers extensions, as defined in the Extensions background section of this document, the following attribute is introduced to complete the list defined in (BONDIA&S):

Attribute	Type	Value
service	URI	URI that identifies the service
app-extension	URI	URI that identifies the extension. If for the extension is not defined an URI, this will be composed by the application URI followed by the name of the extension (using percent-encoding for special characters [RFC3986]) eg. the URI for the extension named "Extension 1" from the manifest of "Application A" which URI is "http://www.ApplicationA.com" will be "http://www.ApplicationA.com/Extension%201"

The attributes "service" and "app-extension" could contain one of the following URIs:

- http://www.webinos.org/resource/info/id/trusted
- http://www.webinos.org/resource/info/id/not-trusted

Action features

The representation of all privileged features that are part of WRT and can be enforced by policy manager but are invisible to the developers, is made through the action features listed below that refers to the widget lifecycle and to the WRT and policy control.

- http://www.webinos.org/action/widget-install
- http://www.webinos.org/action/widget-instantiate
- http://www.webinos.org/action/widget-update
- http://www.webinos.org/action/widget-uninstall
- http://www.webinos.org/action/wrt-update
- http://www.webinos.org/action/wrt-monitoring

- http://www.webinos.org/action/data-logging
- http://www.webinos.org/action/network-access
- http://www.webinos.org/action/policy-management

Policy Examples

Below is presented a group of simple policies that cover different scenarios like widget installation, feature / external services access control, resource sharing, policies outsourcing and management. The examples show policies for single and multiple devices.

In the following the "User U" is the user that can physically control the device and is logged in it.

User U wants to install Application A and grant it access to APIs J0-J9

```
<policy combine="first-applicable">
    <target>
        <subject>
            <subject-match attr="id" match="(Application A)"/>
        </subject>
    </target>

    <rule effect="permit">
        <condition combine="or">
                                                <resource-match   attr="api-feature"
match="http://www.webinos.org/action/widget-install"/>

                <resource-match attr="api-feature" match="(API J0)"/>
                <resource-match attr="api-feature" match="(API J1)"/>
                ...
                <resource-match attr="api-feature" match="(API J9)"/>
        </condition>
    </rule>

    <rule effect="deny" />
</policy>
```

User U wants to grant Application A access to APIs J0-J9, but only on device M

```
<policy combine="first-applicable">
    <target>
        <subject>
            <subject-match attr="id" match="(Application A)"/>
            <subject-match attr="target-id" match="(Device M)"/>
        </subject>
    </target>

    <rule effect="permit">
        <condition combine="or">
                <resource-match attr="api-feature" match="(API J0)"/>
                <resource-match attr="api-feature" match="(API J1)"/>
                ...
                <resource-match attr="api-feature" match="(API J9)"/>
        </condition>
```

```
16     </rule>
17
18     <rule effect="deny" />
19 </policy>
```

User U wants to grant Application A access to APIs J0-J9, on devices M and N.

```
 1 <policy combine="first-applicable">
 2     <target>
 3         <subject>
 4             <subject-match attr="id" match="(Application A)"/>
 5             <subject-match attr="target-id" match="{(Device M), (Device N)}"/>
 6         </subject>
 7     </target>
 8
 9     <rule effect="permit">
10         <condition combine="or">
11                 <resource-match attr="api-feature" match="(API J0)"/>
12                 <resource-match attr="api-feature" match="(API J1)"/>
13                 ...
14                 <resource-match attr="api-feature" match="(API J9)"/>
15         </condition>
16     </rule>
17
18     <rule effect="deny" />
19 </policy>
```

User U wants to grant Application A access to all extensions defined in its manifest (only on his cars) and to API J0 (on all devices)

```
 1 <policy combine="first-applicable">
 2     <target>
 3         <subject>
 4             <subject-match attr="id" match="(Application A)"/>
 5         </subject>
 6     </target>
 7
 8     <rule effect="permit">
 9         <condition combine="or">
10             <condition>
11                                                 <subject-match   attr="target-domain"
match="http://www.webinos.org/subject/device-info/domain/automotive"/>
12                 <resource-match attr="app-extension" match="*"/>
13             </condition>
14
15             <resource-match attr="api-feature" match="(API J0)"/>
16         </condition>
17     </rule>
18
19     <rule effect="deny" />
20 </policy>
```

User U wants to share his photos (P0-P99) with User V.

```
 1 <policy combine="first-applicable">
```

```
 2     <target>
 3         <subject>
 4             <subject-match attr="user-id" match="(User V)"/>
 5         </subject>
 6     </target>
 7
 8     <rule effect="permit">
 9         <condition>
10                     <resource-match attr="device-cap" match="(method to access to
photos)"/>
11                  <resource-match attr="param:(name of the param that identifies a
photo)" match="path_to_photos/P(0|[1..9][0..9]?)" func="regexp"/>
12         </condition>
13     </rule>
14
15     <rule effect="deny" />
16 </policy>
```

Grant Application A to use services provided by Application B, hosted somewhere else (not in the current device).

```
 1 <policy combine="first-applicable">
 2     <target>
 3         <subject>
 4             <subject-match attr="id" match="(Application A)"/>
 5         </subject>
 6     </target>
 7
 8     <rule effect="deny">
 9         <condition>
10              <resource-match attr="service" match="(Services exposed by application
B)"/>
11                                                  <subject-match    attr="target-id"
match="http://www.webinos.org/subject/device-info/id/device"/>
12         </condition>
13     </rule>
14
15     <rule effect="permit">
16         <condition>
17              <resource-match attr="service" match="(Services exposed by application
B)"/>
18         </condition>
19     </rule>
20
21     <rule effect="deny" />
22 </policy>
```

Application A wants to use services provided by Application B, also present on the same device M.

```
 1 <policy combine="first-applicable">
 2     <target>
 3         <subject>
 4             <subject-match attr="id" match="(Application A)"/>
 5         </subject>
 6     </target>
```

```
 7
 8     <rule effect="permit">
 9         <condition>
10                     <resource-match attr="device-cap" match="(Services exposed by
application B)"/>
11         </condition>
12     </rule>
13
14     <rule effect="deny" />
15 </policy>
```

User U works for Company MegaCorp who want to have remote access to work data items W0-W99.

```
 1 <policy combine="first-applicable">
 2     <target>
 3         <subject>
 4             <subject-match attr="user-id" match="(MegaCorp)"/>
 5         </subject>
 6     </target>
 7
 8     <rule effect="permit">
 9         <condition>
10              <resource-match attr="device-cap" match="(method to access to work data
items)"/>
11               <resource-match attr="param:(name of the param that identifies a work
data item)" match="W(0|[1..9][0..9]?)" func="regexp"/>
12         </condition>
13     </rule>
14
15     <rule effect="deny" />
16 </policy>
```

User U wants to restrict Application A from using network N.

```
 1 <policy combine="first-applicable">
 2     <target>
 3         <subject>
 4             <subject-match attr="id" match="(Application A)"/>
 5         </subject>
 6     </target>
 7
 8     <rule effect="deny">
 9         <condition>
10                                              <resource-match    attr="api-feature"
match="http://www.webinos.org/action/network-access"/>
11             <environment-match attr="bearer-name" match="(Network N)">
12         </condition>
13     </rule>
14
15     <rule effect="prompt-oneshot" />
16 </policy>
```

User U wants Application A to work using network N when he is at MegaCorp and wants to use network L when he is at home. [For the phase2]

```
 1 <policy combine="first-applicable">
 2     <target>
 3         <subject>
 4             <subject-match attr="id" match="(Application A)"/>
 5         </subject>
 6     </target>
 7
 8     <rule effect="permit">
 9         <condition combine="or">
10             <condition combine="and">
11                                                 <resource-match   attr="api-feature"
match="http://www.webinos.org/action/network-access"/>
12                 <environment-match attr="bearer-name" match="(Network N)">
13                                 <environment-match  attr="[current-checkin-location]"
match="(Location MegaCorp)">
14             </condition>
15             <condition combine="and">
16                                                 <resource-match   attr="api-feature"
match="http://www.webinos.org/action/network-access"/>
17                 <environment-match attr="bearer-name" match="(Network L)">
18                                 <environment-match  attr="[current-checkin-location]"
match="(Location Home)">
19             </condition>
20         </condition>
21     </rule>
22
23     <rule effect="deny" />
24 </policy>
```

User U wants third party S to manage his policy settings on his device M

```
 1 <policy combine="first-applicable">
 2     <target>
 3         <subject>
 4             <subject-match attr="user-id" match="(Third party S)"/>
 5             <subject-match attr="target-id" match="(Device M)"/>
 6         </subject>
 7     </target>
 8
 9     <rule effect="permit">
10         <condition combine="or">
11                                                 <resource-match    attr="api-feature"
match="http://www.webinos.org/action/policy-manage"/>
12         </condition>
13     </rule>
14
15     <rule effect="deny" />
16 </policy>
```

User U wants third party S to manage his policy settings on all his (PZ) devices: M, N and O.

```
 1 <policy combine="first-applicable">
 2     <target>
 3         <subject>
 4             <subject-match attr="user-id" match="(Third party S)"/>
```

```
 5                                                    <subject-match    attr="target-id"
match="http://www.webinos.org/subject/device-info/id/pz-device"/>
 6         </subject>
 7     </target>
 8
 9     <rule effect="permit">
10         <condition combine="or">
11                                                 <resource-match    attr="api-feature"
match="http://www.webinos.org/action/policy-management"/>
12         </condition>
13     </rule>
14
15     <rule effect="deny" />
16 </policy>
```

User U is not allowed to install applications on device M.

```
 1 <policy combine="first-applicable">
 2     <target>
 3         <subject>
 4             <subject-match attr="author-key-fingerprint" match="(Author D1)"/>
 5         </subject>
 6     </target>
 7
 8     <rule effect="permit">
 9         <condition>
10                                                 <resource-match    attr="api-feature"
match="http://www.webinos.org/action/policy-management"/>
11             <resource-match attr="param:id" match="(Policy for application A)"/>
12         </condition>
13     </rule>
14
15     <rule effect="deny"/>
16 </policy>
```

Developer D1 wants to update the required access control policy for application A as part of an upgrade. [For the phase2]

```
 1 <policy combine="first-applicable">
 2     <target>
 3         <subject>
 4             <subject-match attr="author-key-fingerprint" match="(Author D1)"/>
 5         </subject>
 6     </target>
 7
 8     <rule effect="permit">
 9         <condition>
10                                                 <resource-match    attr="api-feature"
match="http://www.webinos.org/action/policy-management"/>
11                 <resource-match attr="param:id" match="application-id:(Application
A)"/>
12         </condition>
13     </rule>
14
15     <rule effect="deny"/>
```

```
16 </policy>
```

Application A1 wants to access (historic) context information provided by service/sensor S1 on device M. [For the phase2]

```
 1 <!-- SIMILAR TO: User U wants to grant Application A access to APIs J0-J9, but only
on device M -->
 2 <policy combine="first-applicable">
 3     <target>
 4         <subject>
 5             <subject-match attr="id" match="(Application A1)"/>
 6             <subject-match attr="target-id" match="(Device M)"/>
 7         </subject>
 8     </target>
 9
10     <rule effect="permit">
11         <condition>
12               <resource-match attr="api-feature" match="(API that allows to access
(historic) context information provided by services/sensors)"/>
13               <resource-match attr="param:(param that identifies services/sensors)"
match="(Service/Sensor S1)"/>
14         </condition>
15     </rule>
16
17     <rule effect="deny" />
18 </policy>
```

Privilege Apps and Services (Access Control)

The scope and support of providing Privileged Apps and Services into webinos applications for security:

- Preventing unauthorized access, Authenticating users, Enforcing acceptable user policies
- Authorizing all user access based on the user's role
- Identifying and monitoring shared user or device access
- Controlling access, Monitoring and reporting on all network, OS access
- Only Digitally signed applications for the extensions should be allowed to be executed by the webinos runtime.
- Grant access to exchange of Private to Public keys, Certificate, access tokens, nodes, contextual data between the PZP and PZH.
- Identify the treats and list the actions in the access control list. In case of treats block the application or the source to access the resource or the content and notify the User.
- Access control policies that restrict the extensions that try to access the OS services.

The following are the summarised and the identified Privileged requirements for webinos applications

- Interpreting policies and taking access control decisions
- Enforcing multiple policies on multiple devices
- Policy Authorizing tools
- Cross- Device Policy Synchronisation
- Policies for the Stored data
- Dynamically Sharing Content with other Users in a Controlled Manner
- Checking access to APIs – Refers to Content Adaption
- Create contexts from a pre-defined template

Using Privileged Apps and Services in webinos for security purpose

There are two ways fur using Privileged Apps and Services in webinos for security purposes.

1.) Enforce access control policies at the Runtime Environment.
2.) An Application which uses system commands and classes which manages the OS services, access rights, registries, and roles based on the users and so on.

1.) Enforce access control policies at the Web Runtime Environment.

Privileged Apps Policy Examples:
The detailed description with the examples is shown how to enforce access control policies at the Web Runtime Environment which can be found in the Technical Uses Cases and Policy Examples section in the link below:
http://dev.webinos.org/redmine/projects/wp3-1/wiki/Spec_-_Security

Support of Privileged Apps in webinos

Authentication and Authorization:

The Privileged Apps in webinos supports from the first level of User Identity. Authentication is the first step to confirm the user identity and then to grant other security properties, like authorization, permissions and access control to what resources that the user can access all the other webinos devices in the personal zone. The Discovery Mechanism identifies the other webinos devices in the personal zone and authenticates the user. The access control logs the Authentication to the personal zone (user authentication with the personal zone hub). A Notification pops up in the User's profile that the device is enabled to access the other webinos devices in the personal zone. The Privileged Apps and Services monitors and keeps track of the personal zone identity and personal zone proxies. Once the public private key mechanism is generated, retrieving data should be all running in privilege app space updating the user credential information such as password, certificates, applications, status. Enable access to recorded decisions when the user isn't available in real time, e.g. OAuth where a user grants a temporary right to access personal data one server to another remote server.

There is a common authorization model defined for all the trust domains and common language for expressing security policies. The Privileged apps would support authorizations at all levels of granularity. Access control domain that receives such a request will lookup authorization policies that are relevant to the authorization request. Storing Authorization and Certificates in a safe and protected place for the digitally signed applications, so that tampering with them is prevented. Once the User Identity is checked the Privileged apps would look and identify applications which have been granted particular privileges under this particular user. View a list of all their webinos applications and show the authority that certified the application for the users. Privileged apps would have restrictions of the access control policies on applications from potentially malicious applications that cause damages and support device integrity to restrict malwares. Policies shall be enforced to strongly protect from misusing device capabilities while running. Support customized encryption of any data stream. Ensuring that only trusted components are downloaded and that the applications are guaranteed some level of execution and ensure that an application does not access device features, extensions and content other than those associated to it. It would delegate decisions to a trusted third party when appropriate like what restricted capabilities that an application wants to use and to be able to do so in advance of installing or running the application.

Discovery:

When Discovery retrieves the address information of a device or services either through local or remote networking access, the access control should check whether the address, devices or services are valid or not; if it is valid then should allow access if not then it should restrict access. Similarly, if a service driver is required to be installed for the device, privilege application should support the driver. This would require application to be in privilege mode to be able to perform this action. The Device visibility control, device in multicast mode could be passive or active listener. (Privilege app should control the device visibility options). Access control to access different file system area and obtain user credentials information. Capable of setting dynamic access control policies for device data when initiating an association to another webinos device.

Context:

User A doesn't want to allow User B to access his (location, photo, photo-album, a playlist) and other stuffs across other devices as well. So there should be policies defined (example: Resources/File in Deliverable 3.1 Security Specifications: http://dev.webinos.org/redmine/projects/wp3-1/wiki/Spec_-_Security). When a context manager (entity) asks for a context data it shall grant and retrieve the data and the Policies should be able to make a decision (PDP) and enforce them. Privileged apps to look what purpose limitations and what data handling obligations will be agreed by the requesting party upon obtaining the user consent. Review and manage which applications users have granted permissions to, and in what context.

Extensions:

- Only digitally signed applications for the extensions shall be allowed to be executed by the webinos runtime.
- A list of extensions to be logged by the Privileged Apps and Services.
- Privileged Apps and Services shall keep track of the Plug-ins, applications, untrustworthy external code, remote services if accessing the OS services, or disable them at the first instance in case of serious threats, or notify the user. By defining access control policies we can restrict the extensions that try to access the OS level

2.) An Application which uses system commands and classes which manages the OS services, access rights, registries, roles based on the users.

Since, in webinos the Privileged APIs are not provided to the Application developers, an Application developer has to use the Normal APIs for developing the applications which does not access critical data. But, JavaScript APIs for browsers, Extensions, Android classes can be used to access the critical data. Sothese applications have to be signed with a certificate and stored in the certificate stores in the registry of the webinos device.

(Some references of Privileged APIs are based on ideas from this source: http://msdn.microsoft.com/en-us/library/aa455835.aspx)

Application Specification in webinos, whether it is Privileged or Not?

Application execution is based on permissions. Privileged and normal permissions distinguish what applications can do when they run. Applications running at the privileged level have the highest permissions. They can call any API, write to all areas of the registry (including protected areas), and have full access to system files. Applications running privileged can also install certificates on the device. Privileged applications can switch to run kernel mode. Few applications need to run as privileged. In fact, allowing them to run privileged allows them to change the operating system environment, and can threaten the integrity of the device.

Applications running normal cannot access protected registry keys and system APIs. Untrusted and unprivileged applications are not normal applications. Most applications run normal. They cannot call trusted APIs, write to protected areas of the registry, write to system files or install certificates to protected stores. They can install a certificate to the Certificate store. Applications do not run if blocked because they are not allowed to execute. An application could be blocked because it is not signed by an appropriate certificate, because the user blocks it after being prompted, and so forth.

Certificate Signing for webinos devices

OEMs, mobile operators, and application developers use certificates to sign applications and files that run on a mobile device. Certificates are contained in certificate stores in the registry of the device. Privileged Execution Trust Authorities Contains trusted certificates. Applications signed with a certificate from this store will run with privileged trust level (Trusted). Unprivileged Execution Trust Authorities contains normal Certificates. Revocation seems to be a useful concept for webinos that allows a Mobile Operator or device owner to block a specific application or group of applications. The device prevents the execution of the matching binaries. A code signing certificate or a Certificate Authority (CA) certificate can be blocked by revoking the corresponding certificate. This way, a specific application developer or a whole class of applications can be blocked. An individual executable can be blocked by revoking the hash of the binary.

Certificate Signature for webinos Application

If an application needs to be run as Privileged on a webinos device, then the application needs to be signed with a Privilege Certificate. By signing with a privileged certificate the application can call any API, and there are essentially no security restrictions on what the application can do and access. When a privileged application is released it should be ensured that the application is signed with a certificate that is in the privileged store of the webinos device. On the webinos devices the Application Developer should ask the OEM or mobile operator to sign the application.
For a normal application there is no need to sign by the OEM or the mobile operator and the applications are allowed to run since there is no need for a privileged certificate that has to be signed.

XACML Policy Examples in Privilege Apps and Services

XACML Policy examples addressed in Privilege Apps and Services are:

• Access to resources/Installation controlled by the manufacturer (that provides the policy) and then by the user
• Control an application access to APIs (of an extension or not)
• Policies for an Application Installer, A process viewer and Policy management application
• Merge two Processes
• HasPrivilegesOfRole Policies and Requests
• Policies based on Subject and Resource attributes

Please find the examples under this link:
http://dev.webinos.org/redmine/projects/wp3-1/wiki/Privileged_Applications

Policies based on Subjects and Resources:

If an authorization decision is required to base on some characteristic of the subject other than its identity. Then, the most common application is the subject's role RBAC (Role Based Access Control). XACML provides facilities to support this approach. The corresponding policy must also contain a reference to the subject identified in the information resource itself.

Attributes of subjects may be identified by the <SubjectAttributeDesignator> element. This element contains a URN that identifies the attribute. The <AttributeSelector> element may contain an XPath expression over the request context to identify a particular subject attribute value by its location in the context.

Conceptual components

- Security Policy, Privilege Monitor and Privilege manager:
  Security Policy: A collection of access control constraints, abstractly representable as a policy-set, as defined in the security policy model in this specification, that describes the circumstances under which Web Applications are permitted to access Features and underlying Device Capabilities.

  Privilege Monitor and Privilege Manager will log privileged application activity that would fail under a standard user account. For an application to log activity you must enable Privilege Monitoring in the Application Privileges or Shell Integration sections of the policy, when you insert an application group.

- Access to resources:
  Access to Resources based on the security policies and access controls these resources may be a package, attribute, data related to physical entities like (CD player, Camera, Engine), Configuration specification, containing the various files and digital signatures.

- The GUI interfaces for the Privilege Apps and Services are: Application Installation, Package and Launcher:
  The Application Installation verifies the digital signature of the application packages and allows or disallows the installation of an application based on the packages selected. Launcher verifies that an application loads the required files. The Packages consists of data pertaining to the resources, extensions, Manifests and so on.

- Certificate Store:
  A certificate store will often have numerous certificates, possibly issued from a number of a different certification authorities. It consists of key pairs that encrypt and decrypt the symmetric key used for encrypting and decrypting data by Encrypting File System, digital certificates and so on.

Protocols

Device Management:

webinos shall provide Device management for updating of the software that resides on the system or of the system configuration itself.

Supported Device Management scenarios include:

- Installing application software, a complete image, or critical patches and fixes
- Uninstalling or reinstalling application software
- Activating or deactivating installed applications
- Adjusting the user's configuration

The protocols for the following device management capabilities are:

- Configuration management: These Protocols focuses on establishing and maintaining consistency of a device's performance and its functional and physical attributes with its requirements, design, and operational information throughout the process.

- Software and hardware management: Its purpose is to provide a standard interface to configure and operate. It looks for versioning, building, packaging, configuring and installing.

- Notification (alert): Represents how a persistent notification is to be presented to the user using the NotificationManager.

- Logging: Log if there are any errors, log records to a variety of destinations such as log files or the console.

- Remote Monitoring and Remote Diagnostics: Monitor and anaylse the protocols set on the device and check for network connections remotely. Identify the faults and errors in the network and diagnose.

Privacy Certificate Authority Protocols

Token Formats:

Security token is a representation of a collection of any claims, so what to be defined here are the definitions of token formats for this Privacy-CA protocol. The two token formats are as follows:
1)Identity Certificate request Token: This token shall be placed in Identity Credential Request element when Requesting a identity certificate.
2) Identity certificate response Token: This token shall be an encrypted and encoded identity certificate. It is returned from the Privacy CA (A trusted third party that issues platform identities) to the platform.

Remote Data Binding Protocol:

The Remote Data Binding Protocol ensures that the tickets are encrypted by a key that is known only to the ticket issuer, thus event the device owner cannot access the tickets data.

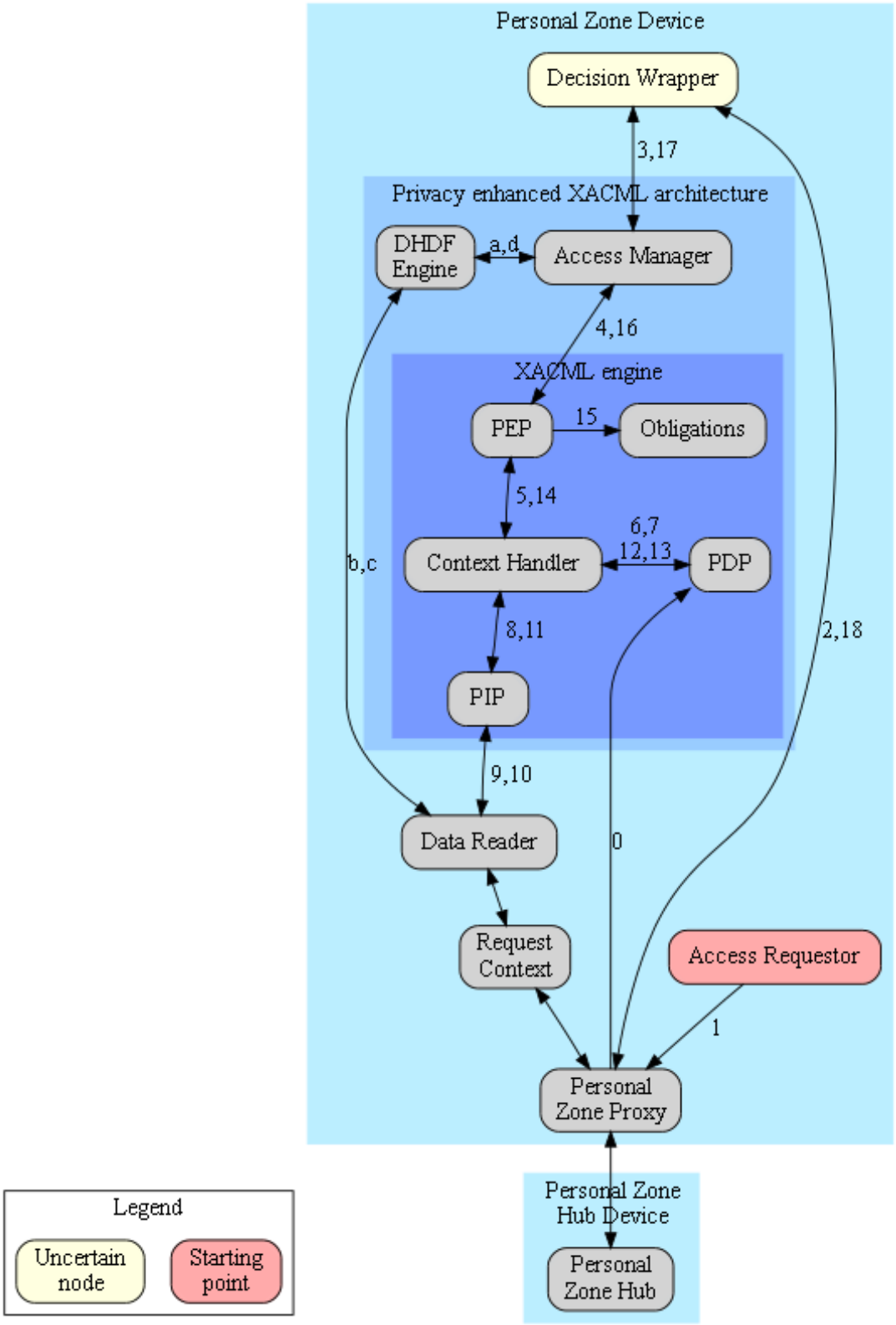
For further information on the Protocol specification see in the link: http://xml.coverpages.org/TMP-ProtocolV10.pdf

Formal Security Specification

Below are depicted three use cases to show the workflow for:

1. requests come from an applications inside the device
2. requests come from outside the Device
3. requests come from outside the Device, and no Personal Hub is reachable (or it is not contacted for resource usage optimization)

Distributed architecture, Case 1: requests come from an application inside the device



Flow description

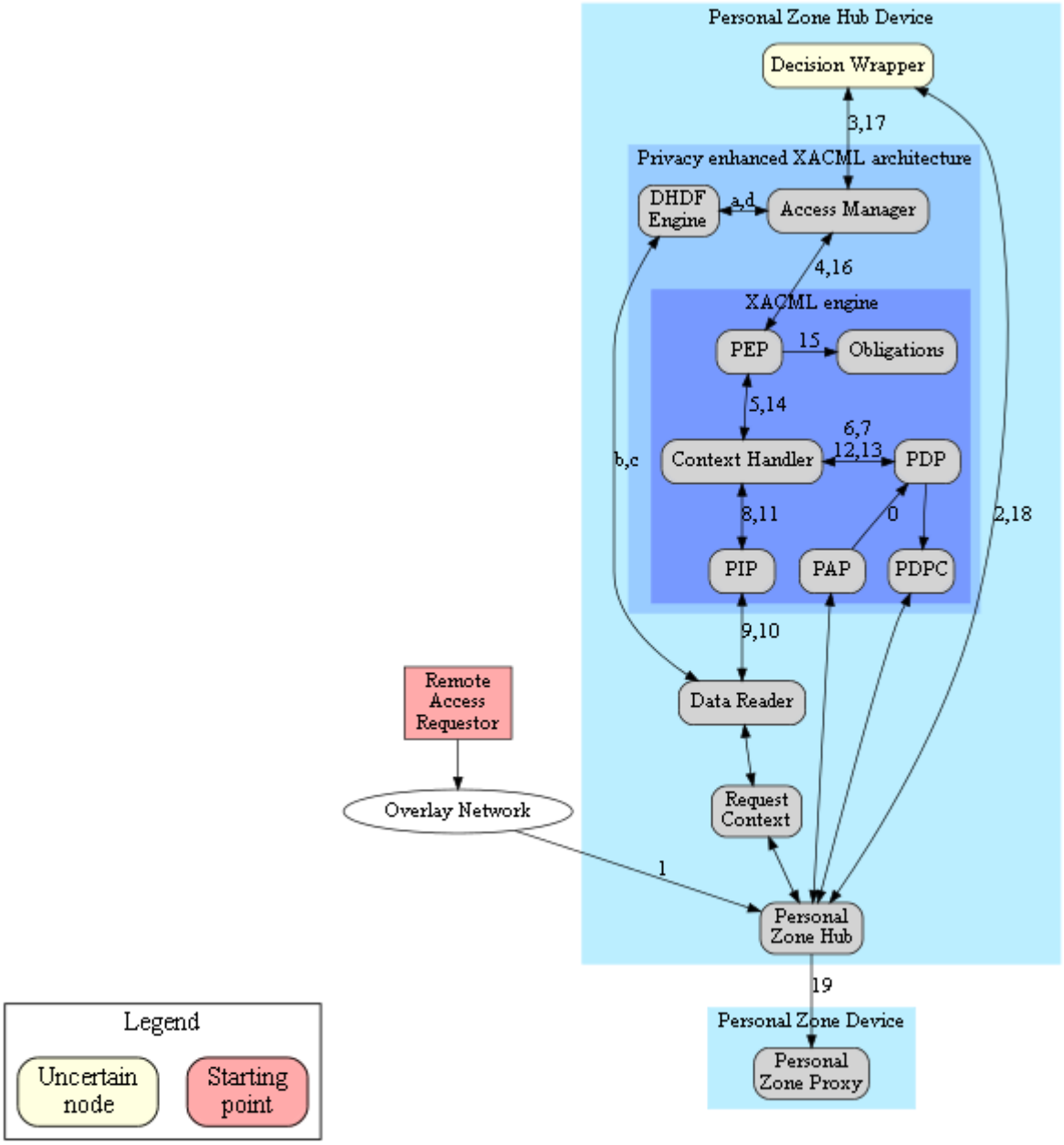
0. PZP update policies and make them available to the PDP
1. The access requestor sends an access request to the PZP
2. The PZP sends the request for access to the decision wrapper
3. The decision wrapper forwards the request towards the access manager, together with the possible relevant
data handling policies attached to the requested resources

XACML engine branch {
4. The access manager forwards the request to the PEP
5. The PEP forwards the request to the context handler
6. The context handler constructs an XACML request and sends it to the PDP
7. The PDP requests any additional attributes from the context handler.
8. The context handler requests the attributes from a PIP
9. The PIP asks the data reader for requested attributes
10. The PIP obtains the requested attributes
11. The PIP returns the requested attributes to the context handler
12. The context handler sends the requested attributes to the PDP which evaluates the policy
13. The PDP returns the authorization decision to the context handler
14. The context handler translates the response to the native response format of the PEP and returns the response to the PEP
15. The PEP fulfils the obligations
16. The PEP returns the access decision to the access manager
}

Privacy enhanced branch {
a. The access manager forwards the request and the Data Handling Policies to the DHDF engine
b. The DHDF engine asks the data reader for evaluation information
c. The DHDF engine obtains information needed for evaluation
d. The DFDH engine evaluated DH policies and returns the decision to the access manager
}

17. The access manager combines the XACML access control decision and the DHDF data handling decision,
then it sends the result to the decision wrapper.
18. The decision wrapper forwards the result to the PZP which enforces it.

Distributed architecture, Case 2: requests come from outside the Device



Flow description

0. PAP update policies and make them available to the PDP
1. The remote access requestor sends an access request to the PZH across the overlay network
2. The PZH sends the request for access to the decision wrapper
3. The decision wrapper forwards the request towards the access manager, together with the possible relevant
data handling policies attached to the requested resources

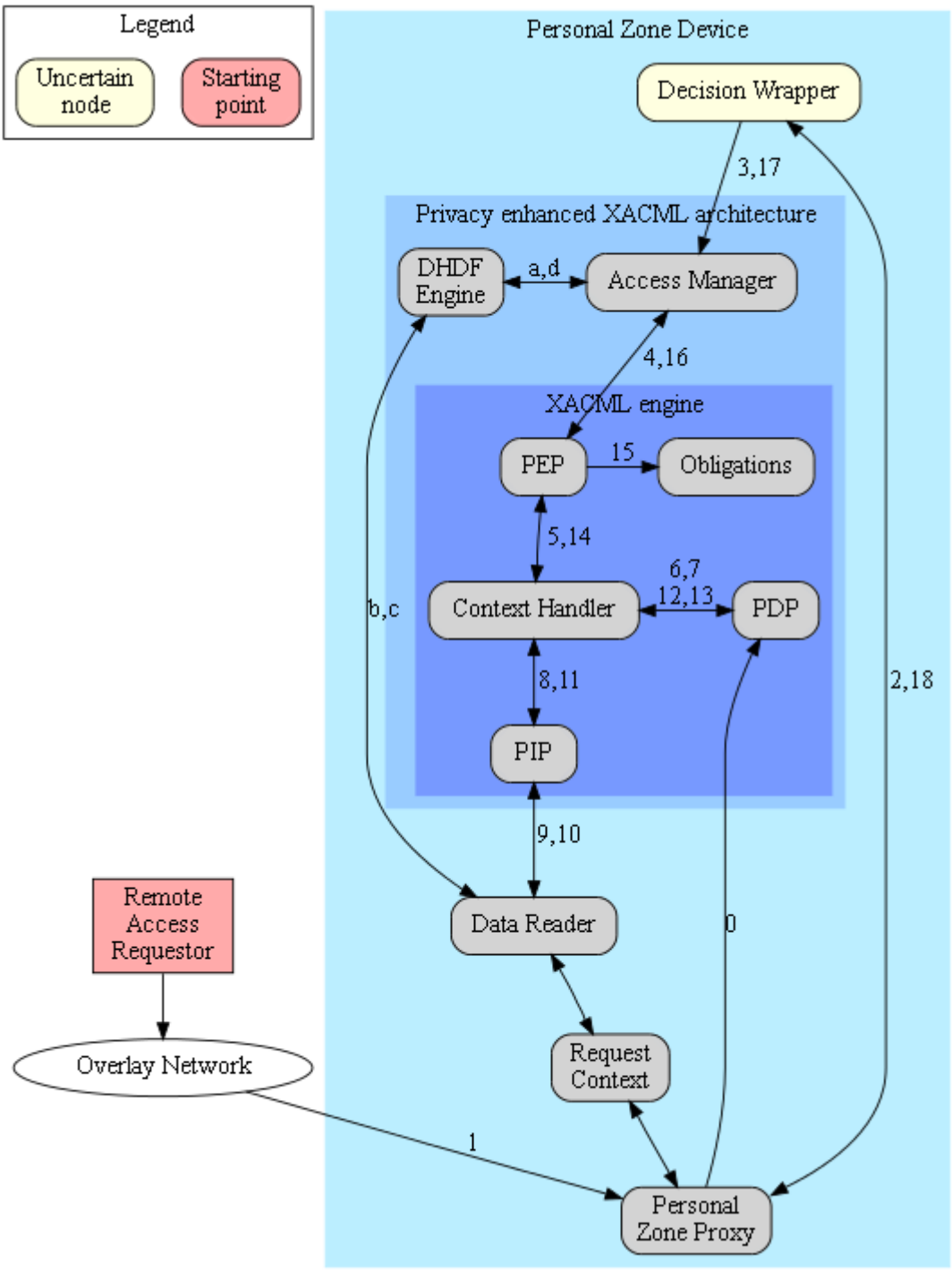
XACML engine branch {
4. The access manager forwards the request to the PEP

5. The PEP forwards the request to the context handler
6. The context handler constructs an XACML request and sends it to the PDP
7. The PDP requests any additional attributes from the context handler.
8. The context handler requests the attributes from a PIP
9. The PIP asks the data reader for requested attributes
10. The PIP obtains the requested attributes
11. The PIP returns the requested attributes to the context handler
12. The context handler sends the requested attributes to the PDP which evaluates the policy
13. The PDP returns the authorization decision to the context handler
14. The context handler translates the response to the native response format of the PEP and returns the response to the PEP
15. The PEP fulfils the obligations
16. The PEP returns the access decision to the access manager
}

Privacy enhanced branch {
a. The access manager forwards the request and the Data Handling Policies to the DHDF engine
b. The DHDF engine asks the data reader for evaluation information
c. The DHDF engine obtains information needed for evaluation
d. The DFDH engine evaluated DH policies and returns the decision to the access manager
}

17. The access manager combines the XACML access control decision and the DHDF data handling decision,
then it sends the result to the decision wrapper.
18. The decision wrapper forwards the result to the PZH
19. The PZH forwards the result to the PZP which enforces it.

Distributed architecture, Case 3: requests come from outside the Device, and no Personal Hub is reachable (or it is not contacted for resource usage optimization)



Flow description

0. PZP update policies and make them available to the PDP
1. The remote access requestor sends an access request to the PZP across the overlay network
2. The PZP sends the request for access to the decision wrapper
3. The decision wrapper forwards the request towards the access manager, together with the possible

relevant
data handling policies attached to the requested resources

XACML engine branch {
4. The access manager forwards the request to the PEP
5. The PEP forwards the request to the context handler
6. The context handler constructs an XACML request and sends it to the PDP
7. The PDP requests any additional attributes from the context handler.
8. The context handler requests the attributes from a PIP
9. The PIP asks the data reader for requested attributes
10. The PIP obtains the requested attributes
11. The PIP returns the requested attributes to the context handler
12. The context handler sends the requested attributes to the PDP which evaluates the policy
13. The PDP returns the authorization decision to the context handler
14. The context handler translates the response to the native response format of the PEP and returns the response to the PEP
15. The PEP fulfils the obligations
16. The PEP returns the access decision to the access manager
}

Privacy enhanced branch {
a. The access manager forwards the request and the Data Handling Policies to the DHDF engine
b. The DHDF engine asks the data reader for evaluation information
c. The DHDF engine obtains information needed for evaluation
d. The DFDH engine evaluated DH policies and returns the decision to the access manager
}

17. The access manager combines the XACML access control decision and the DHDF data handling decision,
then it sends the result to the decision wrapper.
18. The decision wrapper forwards the result to the PZP which enforces it.

Analytics

The objective of webinos analytics is to help application developers provide and learn more about the application and its users. It is designed for application developers that would like to outsource the analytics task to gain more focus and time to develop the application. Webinos analytics is the process of aggregating reports from application sessions that users have with the applications. Gathering metrics is the process of gathering and making information about the individual application sessions available for the analytics. Conducting metrics is therefore the enabler for providing analytics. Webinos applications must be equipped with webinos metrics components as means to enable the aggregation of webinos analytics. Analytics is the process of aggregating data from the captured metrics data, and performing the analytics on top of the metrics can be seen as a specialised application process.

How does the webinos metrics and analytics work for the application developer and the users? There are many ways one can perform analytics and subsequently many architectures that could be implemented for providing analytics. For webinos applications, the metric data can either be conducted within the local application on the client side, or alternatively remotely in conjunction to the Web site that the applications on the devices communicate with. Webinos offers two different ways to collect the metrics data: within the application on the user device, or remotely on the Web site that the application communicates with.

Conducting webinos metrics is quite straight forward. The overall user scenario to keep in mind is:

1. The user interacts with the application on her device, such as requesting for some content on the Web
2. The application uses some webinos metrics software - either on-device, or on the Web
3. The webinos metrics software collects metrics data about how the application is used
4. The gathered metrics data is securely sent/uploaded to the analytics service
5. The analytics service gathers and analyses the metrics data for the application developer
6. The application developer logs on to the analytics service to see and learn about the application and its users

How to integrate webinos analytics with the application

Webinos applications can run across multiple devices and Web servers. Much of the communication in a Web application is via the Hypertext Transfer Protocol (HTTP). The HTTP-protocol is regarded as stateless, because there is no notion of state preserved by the protocol itself. Webinos applications are however not stateless, because the user has to interact with and go through various stages to interact with the application. Any notion of state has therefore to be explicitly gathered and stored somewhere by the Web application. This can either happen on the device or on the Web site in a webinos application.

The application on the user device sends HTTP-requests to Web sites for content, and as part of these Web request, it has to provide sufficient context information for the Web site to compose and provide the correct response.

The chosen implementation strategy for the webinos analytics is to provide a suite of metrics components that together can cater for the needs of a wide range of Web application architectures in need for outsourcing to webinos analytics. For example, it could be an offline application running locally on the device, that connects to the Web perhaps once per month, or it could be a Web application where several thousand HTML requests and responses travel between the client and various Web servers. By addressing both of these, we believe we will able to address most needs for metrics and analytics in Web applications, because one can choose to gather the metrics data in the application instance running on the users' device, and one could also want to perform some analytics work as light weight Web workers on the Web server side. The work balance for metrics gathering can therefore be shifted towards the online Web servers or more towards the client application on the device depending on the analytics requirements of the application.

An important aspect for webinos analytics components is for users and application developers to be able to trust them. Providing for analytics across applications therefore means catering for privacy. It is in general possible to monitor individual interaction with application, but also to aggregate statistics for a set of users. Most analytics approaches today provide aggregated statistics for all users using the Web application/ Web servers. However, if one places some metrics on the local device, then one can achieve very accurate and also individual statistics. This could be used for an application to provide smarter, and more personalised information experiences. Note however that individual statistics about a user always is personal and sensitive data above all. Storing and using such data requires permission/ consent from the user.

The scope of the webinos metrics components is to be a solution for application developers to be able to plug in and start using webinos analytics.

As with Web applications, webinos applications can be built in the following ways:

1. Client-side application
2. Client-side and server-side application
3. Server-side application

The communication between the client and the server will normally be using the hypertext transfer protocol.

Unlike static websites, webinos applications offer a richer experience which can be more integrated with mobile devices, tablets, car infotainment, and laptops. The webinos applications will be able to access location services, motion, camera, local sensors, networks, and so on. The webinos applications will have different session models and, as mentioned will also enable offline use. This means that the scope of collecting metrics in the webinos platform is two-fold:

- To enable the gathering of metrics data through the use of the context API
- To enable the gathering of metrics data pertinent to HTTP-requests and responses; including context information around these

Conceptual Architecture

Reading this will help webinos application developers understand the conceptual level of integrating the application with webinos metrics components. An application developer integrates in the same way on any webinos enabled device. This makes it easy to develop and it obtains efficient development, allowing application developers to focus on what the application should do instead of what marketing managers would like to know.

It takes only some minutes to read through this guide to the conceptual architecture, and afterwards you will be able to spend only a few hours to plan what you would like to conduct metrics on, and then you integrate this with your application. It is possible to only read the metrics API specifications to be able to integrate. However, we recommend for every application developer to spend some time to read through this conceptual document in order to know the possibilities before starting to integrate with webinos metrics.

As mentioned, this documentation is independent of the target device that your application will be/ or is running on.

Overall system concepts

Here is some of the core overall system concepts and vocabulary that application developers will need to be aware of when we discuss webinos analytics:

Webinos application sessions

A webinos application session starts when an application instance starts, and ends when the application instance stops. Each session is uniquely identifiable for your application. Throughout the session there will be certain points and events that you will be interested in gathering metrics about and analysing.

Webinos metrics data and metric points

The metrics are chunks of data that you are interested in gathering. This can be about the user context, the device context, the application context, specific events that occur, or contextual information about HTTP-requests and their responses, in your application where you would like to conduct data and metrics. Thus, metrics is context information that captures important aspects of the user, application, and the device. The metric points are the places in your application that you would like the metrics to be conducted.

Webinos metrics events

A metrics event is created by you at the every chosen metrics point in the application. An interesting event occurs in the application and you as an application developer decide that it is important to log this event. A metrics event contains metrics data. Here are some examples of metrics data that application developers might want to log:

- The user opens a particular view or page
- The user changes her profile
- The user is using different devices
- The user earns more loyalty points
- The user checks out and pays for products
- The user arrives at a new destination
- The user changes the privacy policy
- The application presents an advertisement
- The user sends a new HTTP request
- The application receives a HTTP response
- The application crashes
- The device goes off-line
- The device goes online
- The battery status of the device is low
- The application is upgraded
- The user has changed to another device
- The user specifies some interests
- The user queries for some information to a search facility
- The application starts
- The application stops
- The user chats with another user
- The application sends data to another application
- The cookie(s) in the Web application are changing

Webinos metrics workers

Metrics can be conducted either locally on the user device or remotely on web servers. In both cases there are metrics worker threads/ processes that perform the metrics separately in the background. This enables both heavy- and light-weight metrics tasks to be executed without affecting the user interface behaviour or any other communication activities such as web requests and responses. The overall idea is to conduct webinos metrics as a background task preferably on the user device, and

alternatively on the web server side. In webinos, we therefore propose two metrics types of metrics workers:

- **Application metrics workers** - integrated with the local application on the device
- **Server metrics workers** - integrated at the front-tier of remote web servers

A Web site can be standalone of distributed across several Web servers. In the latter case, information from the metrics events occurring in the client application will be needed to glue together Web logs on the server side. It is difficult to uniquely identify application sessions on the website side. Logging the webinos metrics event on the client side is therefore the recommended route, however, sometimes this will lead to too much metrics and the marketing manager might instead want you to monitor what is going on the Web server side and not bother too much about what happens on the client application.

An e-commerce Web site might consist of a product offering server, an advertisement server, and a separate payment service provider. In this case it will be difficult to know what is going on in the application session, because the dynamically generated Web pages from the domain name result in a response that is build from subsequent requests to the other Web servers in the background.

Webinos server side deals with this problem by offering webinos metrics workers for the Web server. These components are geared towards Web sites that consist of several Web servers, and when there is a need for metrics on the server-side to be able to act in the metrics data in the situation between a Web request and its response from the Web site.

Application metrics logs and server metrics logs¶

Webinos metrics workers collect metrics data and add them to the metrics logs. A webinos metrics log is either stored locally on a device or remotely on for instance a web server. Metrics logs located on the user device are called application metrics logs. Metrics logs created by a worker thread/ process on a remote Web server are referred to as server metrics logs. Both types of metrics logs are created by the webinos metrics workers which are either running locally on the user device, or remotely hooked up and integrated in the front-tier of the Web site / Web farm.

Technical Use Cases

Use case I: Integration with webinos local application metrics library

1. Register your webinos application client
2. Create a private webinos analytics key unique for the application upon the completion of the registration process
3. Download and install the webinos analytics library from webinos for client-side applications
4. Use the policy manager to register your intent for using metrics
5. Use the context manager to obtain access to context information about the user, the device, or the application context. You can create and/ or use context information that is private for your application.
6. Add calls to startSession(), pauseSession(), resumeSession(), uploadSession(), and closeSession().

7. Create metrics events to record specific events, and add the relevant context information to these events as arguments to the metrics event.
8. Test and evaluate the integration by running the application on a client
9. Log in to your webinos analytics page to see how the metrics data is presented.

Use case II: Integration with the webinos Web worker metrics library

1. Register your application Web site
2. Create a private webinos analytics key unique for the Web site upon the completion of the registration process
3. Download and install the server metrics library from webinos for webinos-enabled Web sites
4. Use the policy manager to register your Web server's intent for using the server metrics library
5. Use the context manager to obtain access to context information about the user, the device, or the application context whenever Web requests occur.
6. Create metrics events to record specific Web requests and responses, and add the relevant context information to these events as arguments to the metrics event.
7. Test and evaluate the integration by running the Web site
8. Log in to your webinos analytics page to see how the metrics data is presented.

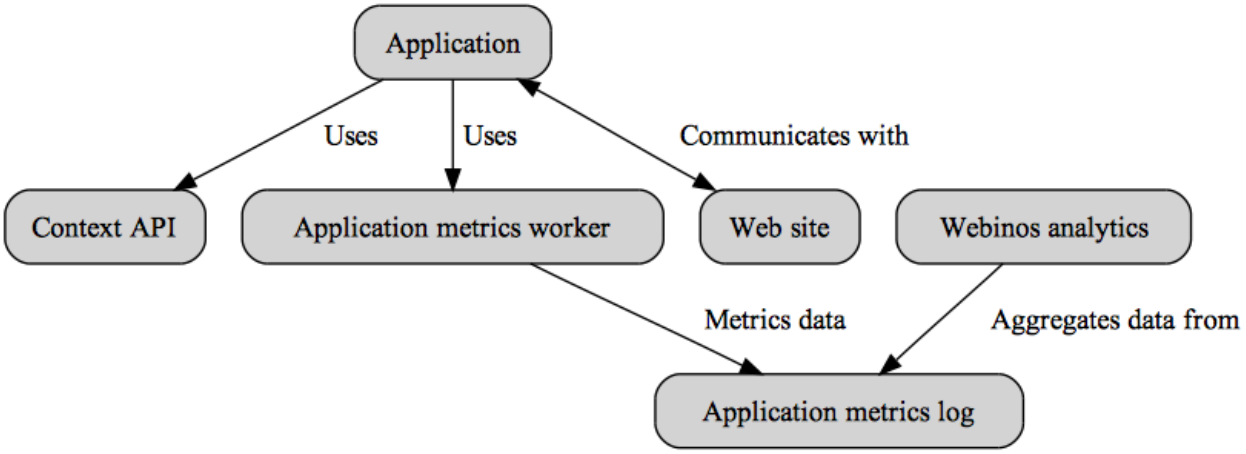
Formal Specification

Reference architecture for application metrics workers

The following diagram shows the reference architecture for when an application developer chooses to implement Use Case I above. The advantage of following this approach is that one can very accurately monitor and capture the application session for each individual application on the device. It ensures a high quality of the gathered metrics data, because the application on the user device is in full control over the interaction states that the user is going through. The general principle is that each application can monitor the local metrics data and events as and when these happen on the device. Moving the webinos metrics intelligence towards the application client can also significantly reduce the computational resources needed for the webinos analytics service to effectively aggregate analytics data from the metrics data of the individual application sessions.

The application registers its intent to use the application metrics library. It also does the same for the context API. If the webinos runtime allows this, the application will be writing metrics data to the local application metrics log when it encounters the metrics points in the application. The application developer is able to consume the context information provided by any context provider on the device - provided that the policy manager permits. Such context information originating from other context providers in the system is captured and added to the application metrics log. Examples of such context information are location, nearby devices, wireless networks, and so on. Furthermore, context information about the application, such as when it started, paused, resumed, stopped, and ended can be logged. Finally, metrics data about web requests can be added to the log by the application metrics worker - again only if the policy manager permits. Thus, any context information sourced by a context

provider on the end-user device, including also context information about web requests and responses, can instantly be logged by the application metrics workers.



The webinos analytics receives the application metrics log from a background task that is running on the user device. The application session is easy to maintain locally, because there is only one session. The application maintains its own state, and the application developer can choose to gather and add whatever context information believed to be needed for the analytics. Capturing context information about web requests is also straight forward. The application developer makes use of the context API to capture and manage the web context attributes whenever the application is requesting for information from remote web servers. The aggregation of analytics from the logged metrics data becomes an easy task, because all of the captured metrics data is uniquely related to the application sessions at hand, and well defined context providers on the user device - such as for example various sensors.

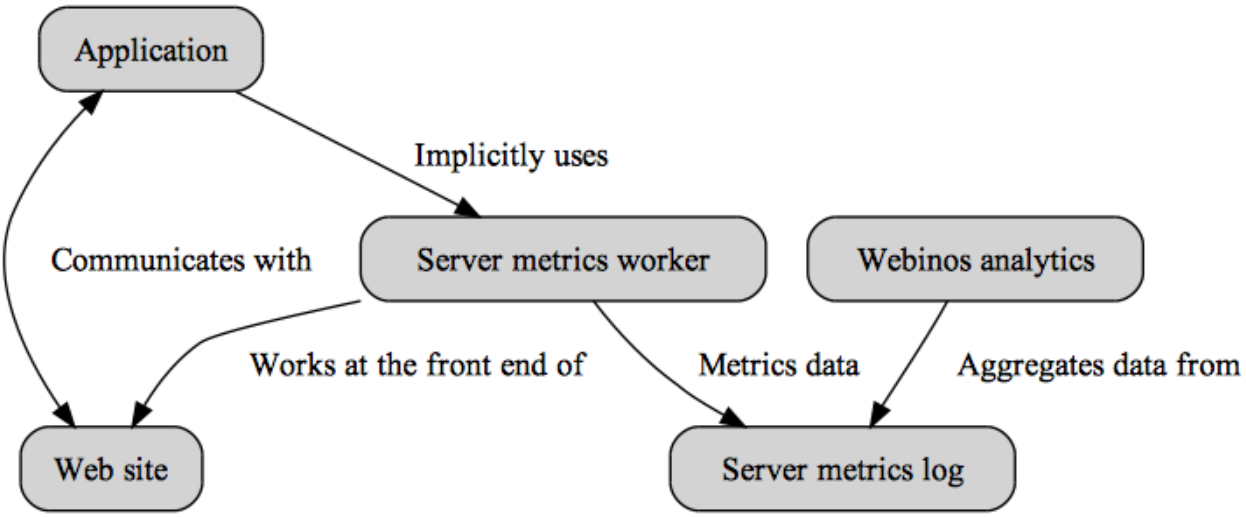
Note that webinos-enabled applications using the webinos metrics library need to flag the intent to use the API. This means that it is possible for the webinos runtime, and the individual users, to switch off all or some of the gathered metrics data at any time. We believe that this will create more trust and transparency for the users compared to the existing approach of, for instance, Google analytics, which is: "don't tell the user about what is being metered, and don't let them switch it off either".

Thus, both from the local metrics and privacy point of views, webinos analytics makes progress in favour of users compared to existing analytics solutions for the web. Moreover, from the application developer's point of view, it is possible to gather richer metrics data, because the application metrics worker is working locally within the context of each application session.

Reference architecture for server metrics workers

This next diagram exhibits the reference architecture for when an application developer chooses to implement Use Case II above. The advantage of this approach is that the metering capability is added to the front-end of the Web site. This is a less intrusive approach, because the metrics worker is integrated on the Web server instead of within the client application. This approach is also capable of capturing metrics data about individual application sessions. The difficulty, however, is in being able to keep track of the application session and state when interacting with the Web site. In general, this is regarded as more difficult to do on the Web server side, because the amount of context information that can be captured other than those directly related to the Web requests. The usefulness of responses is reduced

due to the stateless nature of the HTTP protocol, and also because there can be many sessions from different applications instances to serve at the same time. However, it ensures that the client application code does not have to be altered to achieve the metrics data, because the server metrics worker is responsible for capturing states that the user is going through. The general principle is that each Web site gathers the metrics data of individual application sessions as and when these events happen. Moving the metrics intelligence towards the server side however does increase the computational resources needed for the webinos analytics service to aggregate quality data from the metrics. The metrics points would not always correlate with the points of Web requests, and similarly it sometimes would also be artificial to send Web requests to a Web server just because a metrics point in the client application was reached. Such situations create overhead for the webinos analytics in terms of aggregating analytics data.



The server metrics workers use the context information available on the Web server in conjunction with Web requests and responses to capture and elicit metrics data. A Web request starts inside the client application on the device when the user clicks a Web link, or formulates the request via some other means. The request lasts until the local application has retrieved, parsed, and rendered the response to the user. During the response period, the Web site performs a variety of subsequent activities often starting with sending cookies to the local application at first, before proceeding to respond with subsequent content that in return can lead to subsequent requests for other content such as more HTML text, scripts, multimedia, and images. In a webinos-enabled application that works by performing many Web requests and responses during the application session, it can be natural to conduct metrics on the contextual information pertaining to web requests and responses.

Web servers in general have a set of environment variables that together define the context of the Web server for each connection, including Web requests and responses. Some of these variables are Web server specific, whereas the majority of them are Web request specific. One can obtain a list of these variables on the Web server. Server metrics workers have access to these environment variables and use a cookie framework to keep track of individual application sessions. Although, it is in principle possible to extend the set of context information available on the Web server side, we believe this approach to be more limited compared to gathering metrics data from within the actual application

instance residing on the user device. For more on available environment variables (i.e. contexts information), seehttp://en.wikipedia.org/wiki/Common_Gateway_Interface#History

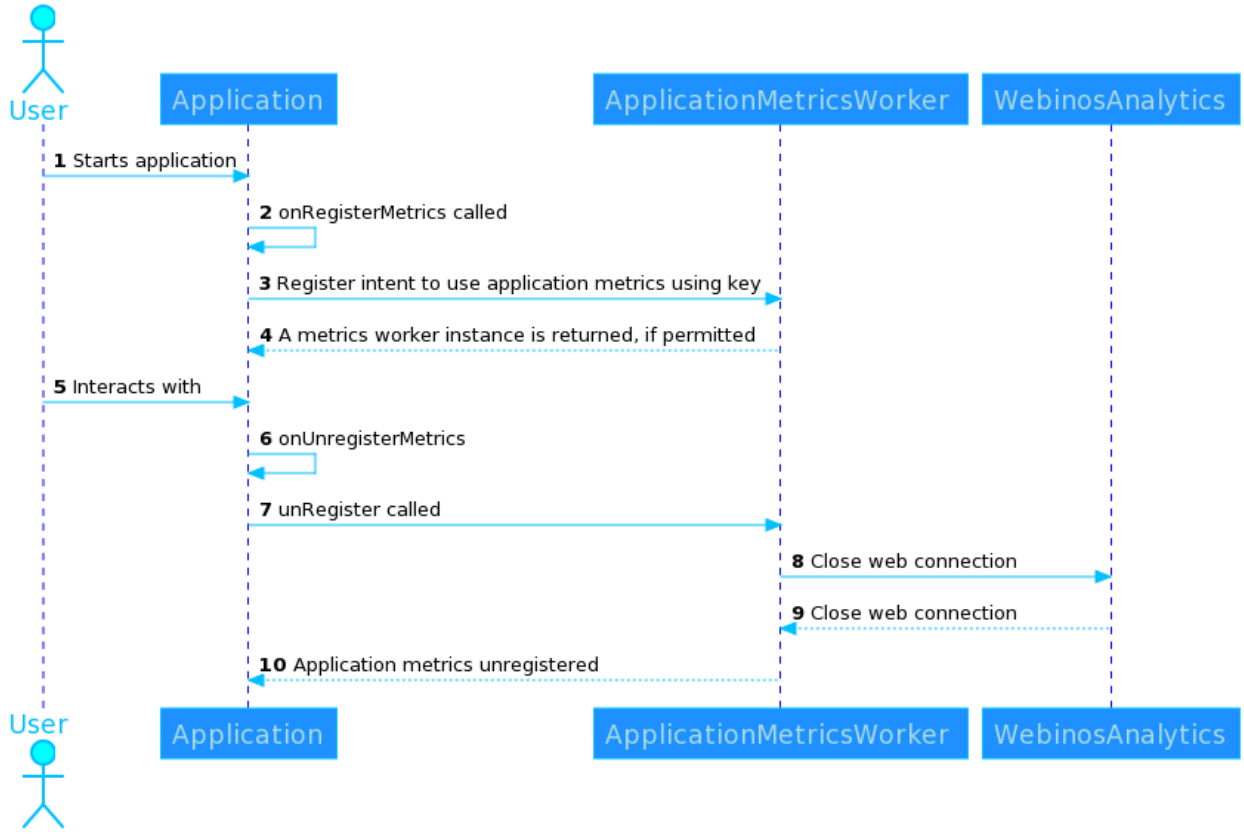
Note that from the end user's point of view, this use case is less intrusive, gathers less context information, but is more difficult to control because the server metrics worker is not deployed on the user device.

UML sequence diagrams for application metrics workers

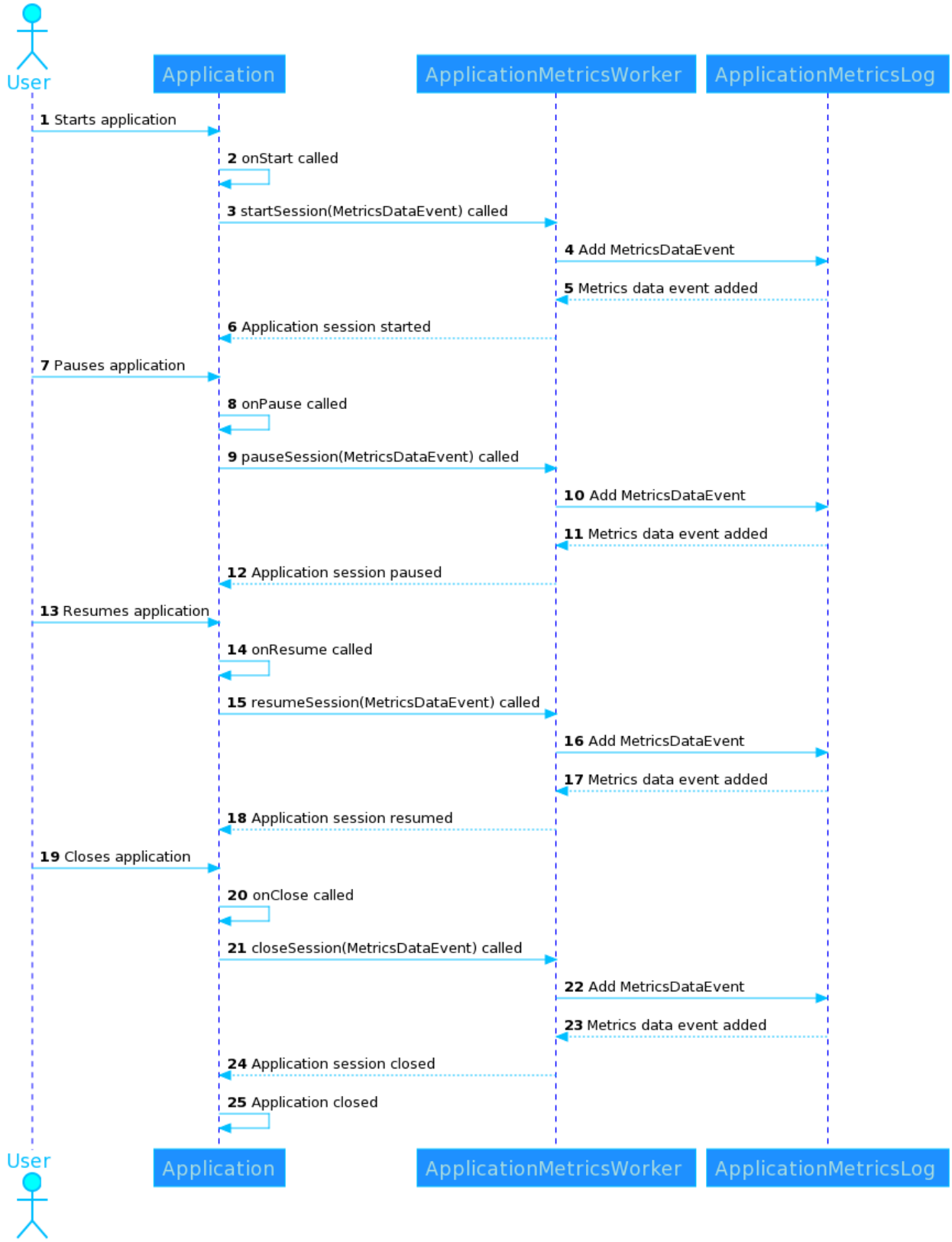
This section presents sequence diagrams that involve application metrics workers on the user device. These are:

- Register the intent to use the application metrics worker
- Gathering metrics data about the application session lifecycle
- Register the intent to consume specific context information
- Using context information from specific context providers as metrics data
- Adding web requests and responses as metrics data
- Encountering metrics points within the application
- Uploading the metrics data to webinos analytics

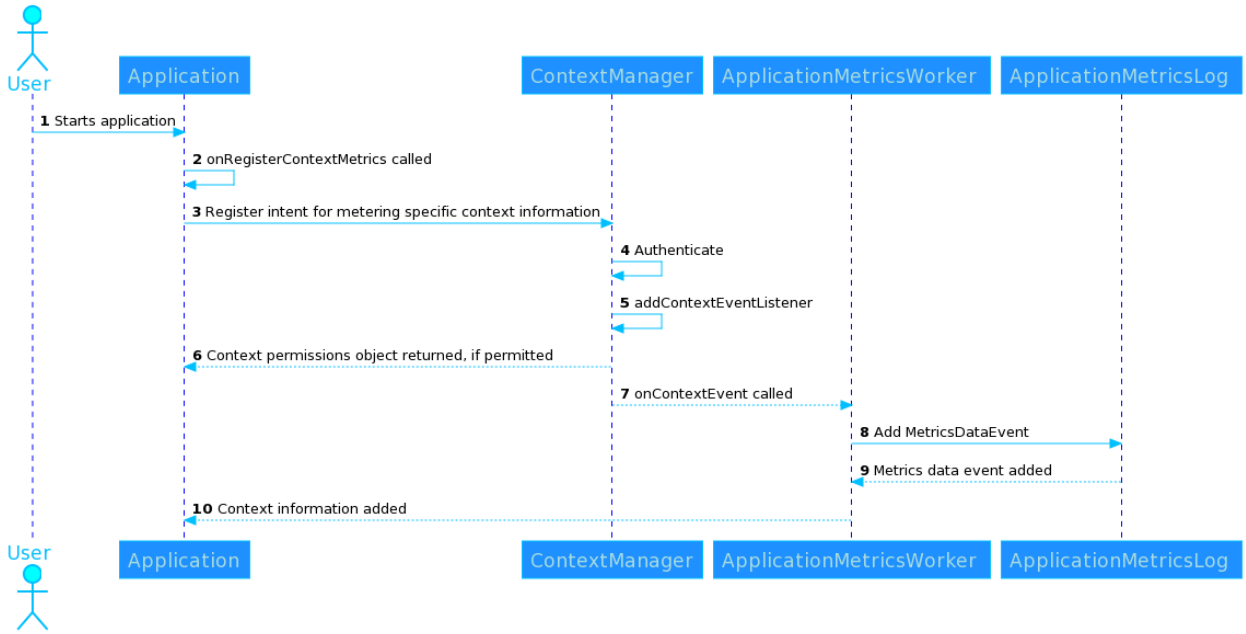
Register the intent to use the application metrics worker



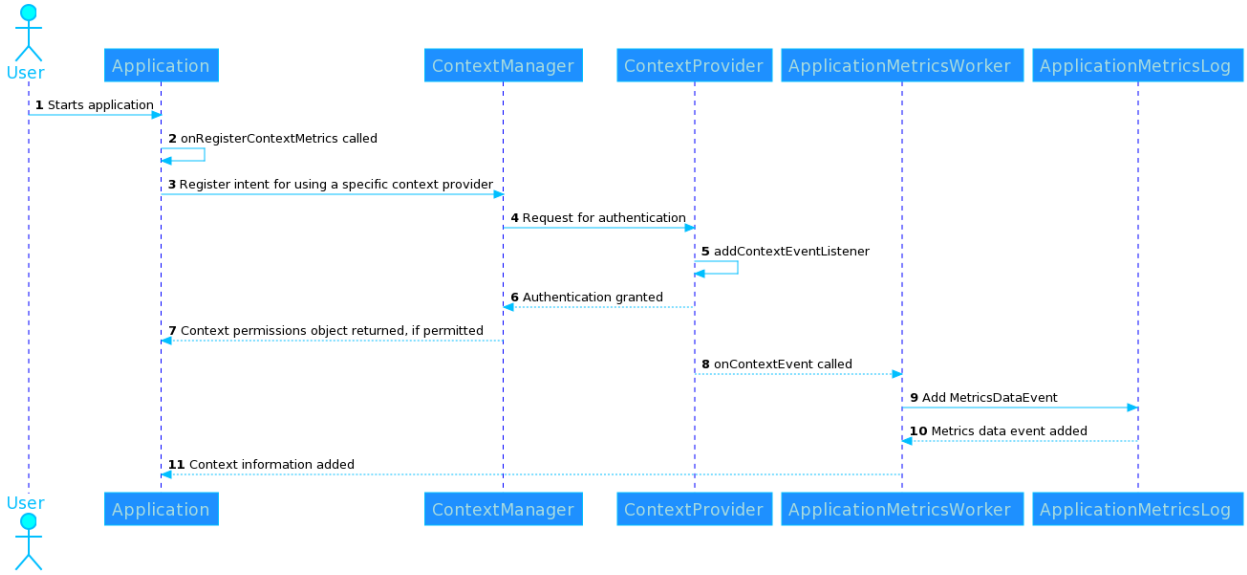
Gathering metrics data about the application session lifecycle



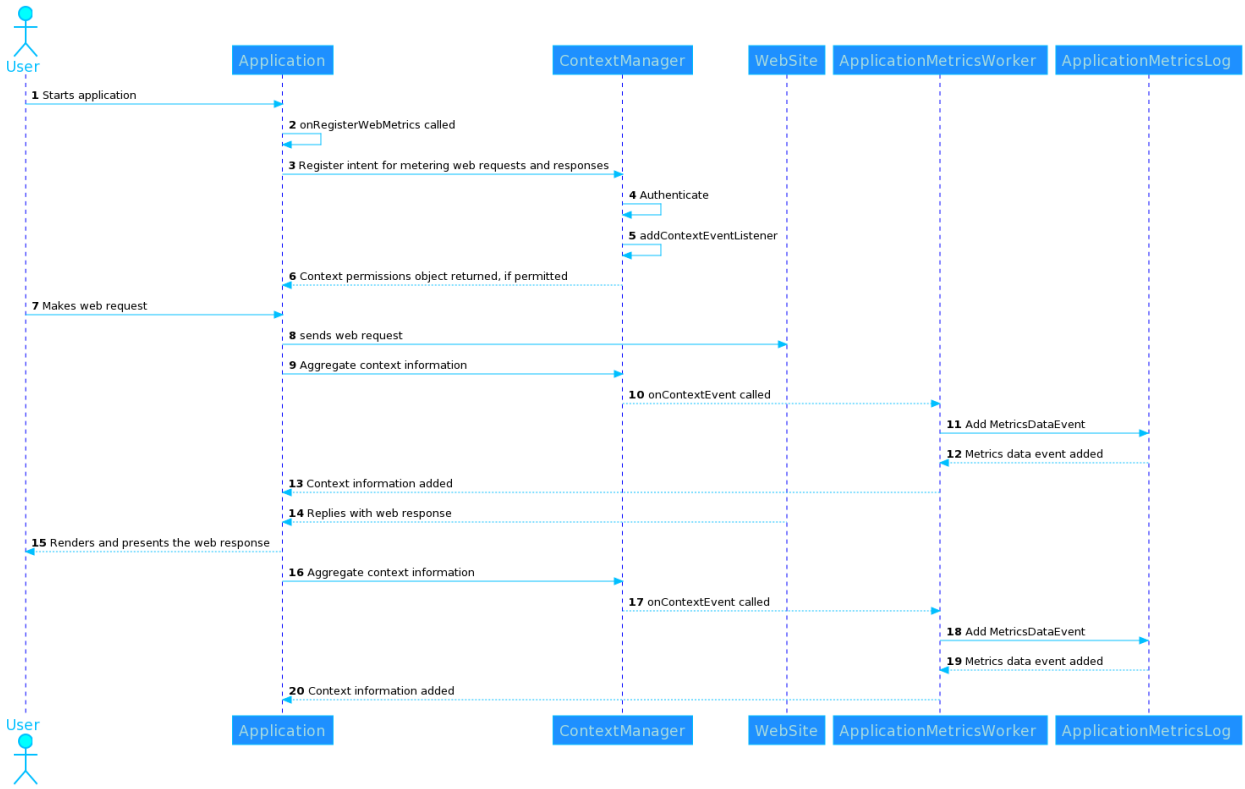
Register the intent to consume specific context information



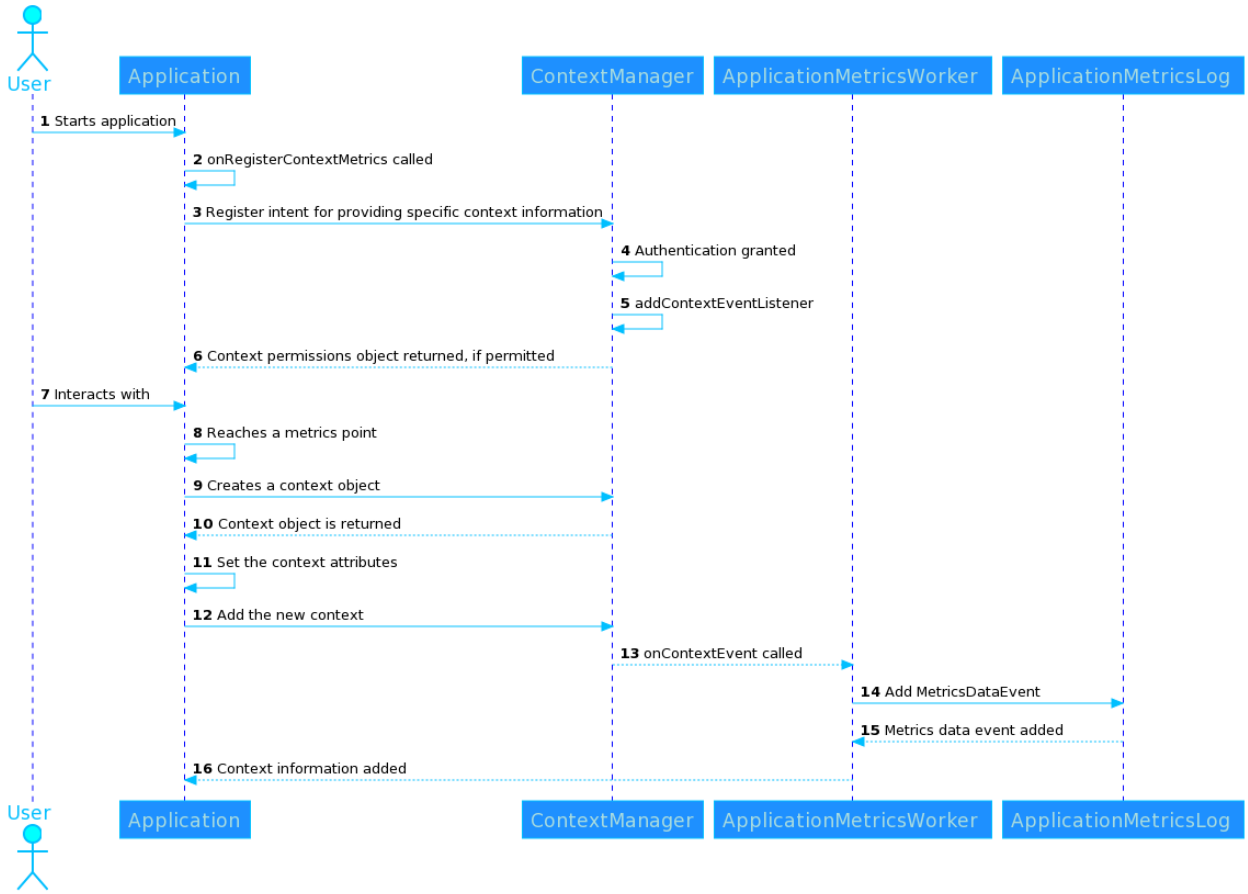
Using context information from specific context providers as metrics data



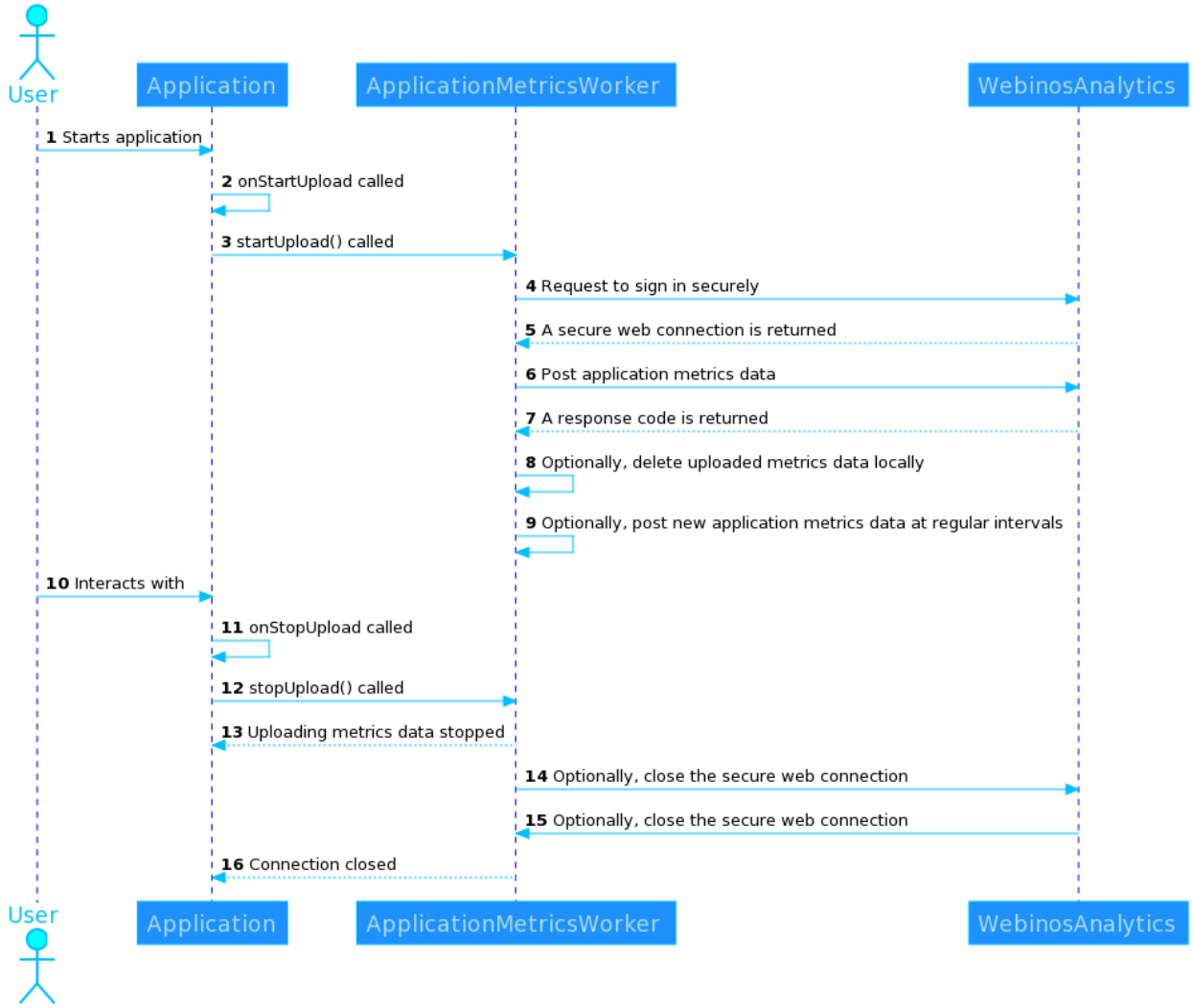
Adding web requests and responses as metrics data



Encountering metrics points within the application



Uploading the application metrics to webinos analytics



Functional and non functional requirements

The following table shows the requirements related to the webinos metrics/ analytics specification. They are from the requirements report (Webinos-D22) where they were ranked as first priority:

Requirement no.	Description
ID-USR-DT-02	The webinos system must minimise exposure of personal individual identifiers or canonical identifiers of webinos entities.
ID-USR-OXFORD-34	It shall be possible to include device identity information in application session data.
PS-DEV-ambiesense-08	The webinos runtime environment shall support customised encryption of any data stream (independent of its data type or format).

PS-DEV-Oxford-89	A method must be provided to enable webinos applications to explain how collected sensitive data will be managed (e.g. company name, purpose description).
PS-DWP-ISMB-202	The webinos runtime must ensure that an application does not access device features, extensions and content other than those associated to it.
PS-DWP-VisionMobile-12	webinos shall use user privacy preferences when granting/denying access to user private information.
LC-DEV-POLITO-001	The removal of an application on a device must remove any personal data linked and specific to that application installed on the device.
CAP-DEV-ambiesense-052	An application shall be able to run multiple threads within an application, so that the user interface does not temporarily stop/freeze for the user when the application is busy with downloading, uploading, or otherwise processing data.
CAP-DEV-FHG-200	The webinos runtime shall be able to run applications in the background.
CAP-DEV-SEMC-010	webinos shall provide means for applications to access user's profile data.
CAP-DEV-SEMC-011	webinos shall provide means for applications to access user's preference data.
CAP-DEV-SEMC-014	webinos shall provide means for applications to read and write to local file storage.
CAP-DWP-VisionMobile-31	webinos runtime shall support a standardised way of collecting application analytics.
CAP-DWP-VisionMobile-32	webinos runtime shall support a standardised way of collecting application performance statistics and failures.
CAP-DWP-VisionMobile-33	webinos shall expose application analytics data to the application developer.
CAP-DWP-VisionMobile-34	webinos shall be able to expose the application performance data to the application developer.
TMS-DWP-NTUA-	The system must be capable to identify, store, retrieve, and communicate contextual

001	information for an application executing on user's devices.
TMS-DWP-NTUA-003	A webinos runtime must be able to identify and store contextual information for the present situation within which an application is executed.

Dependencies on other components

Application metrics workers are dependent on the following:

- Context API
- Secure storage
- Secure Socket Layer communication

Server metrics workers are dependent on the following:

- The web server platform of choice
- The Common Gateway Interface of the web server
- Secure storage

Synchronisation

This section is work in progress and a complete specification will have to wait until we have progressed further with implementation work.

Introduction

Personal Zones are an abstraction layer encompassing your personal devices and services. This layer is exposed as a local API in each device as well as the personal zone hub that runs in the cloud. For this abstraction layer to function, the shared context needs to be synchronized across the zone. The context covers authentication data, user preferences, device capabilities and environmental conditions, and provides a means for applications to dynamically adapt to changes in the context. The Personal Zone needs to function even when devices are offline, or are temporarily unable to access the Internet.

Synchronisation data storage

This involves a timestamped history of recent changes held in memory or persisted as JSON.

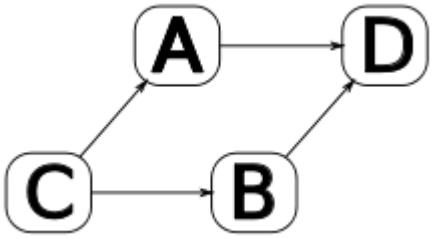
Synchronisation Triggers

Synchronization is triggered when a device first registers with the zone, when it wakes up after being powered down, or it is switched back to online mode, or when changes are signaled across the zone. Note that changes may be buffered to improve network efficiency and to prolong battery life.

Synchronisation algorithm

Synchronization can take place pairwise between devices, or between a device and the personal zone hub. If the data model consists of independent data items, one algorithm is to pick the latest version of a given item. This involves an assessment of the clock skew between devices, and assumes each webinos enabled device has a clock. If the data items are not independent, the merged data needs to satisfy the data integrity constraints. A transactional model of updates can help, by providing the time of the latest transaction. This assumes each device keeps a time stamped history of recent transactions. The algorithm should further be designed to recover from error conditions where a given device violates the integrity constraints.

The three way merge algorithm starts with the current state of the two devices (A and B) and then determines the most recent common ancestor (C), before identifying what further changes are needed to produce a merged version (D). This will involve a means to resolve clashes, which may involve giving precedence to one of the two devices.



Courtesy of David Pattison

Identification of the base version can be facilitated by a mechanism for keeping a history of recent changes, and labeling versions of the data set to be synchronized. This is easiest in a hub and spoke model, but can be generalized for the peer to peer case when the hub is unavailable.

Synchronisation protocol

This assumes that the data set to be synchronized can be described by a hierarchy of data items. The protocol involves searching for a common base version and then exchanging updates since that version. The updates can be described in terms of add, delete, update, and move operations. Updates are serialized as JSON, along with the version number and timestamp. This is work in progress, and we expect to provide further details as the work proceeds. The general feasibility of the approach has been demonstrated by distributed revision control systems such as mercurial, see:

- Mercurial: The Definitive Guide, Bryan O'Sullivan, 1 January 2007, http://hgbook.red-bean.com/

6. Conclusions

This document, together with webinos deliverable 3.2 "Webinos phase I device, network, and server-side API specifications" and deliverable 3.5 "Webinos phase I security framework" defines the specification used for the first phase of the webinos project.

For the use within the webinos project and for implementation purposes, the Wiki version of this specification is considered to be definitive. For publishing and filing purposes and for offline reading, 'snapshot' documents of this Wiki will be created and made available.

The initial webinos platform will be built in WP4 of the webinos project based on this specification. This platform will than be used to demonstrate that it provides the functionality required by existing Web applications, adds new concepts and provides new features for future applications and will act as a testing field for innovative concepts, for example in the area of context/analytics/metrics handling.

Based on the experiences with this initial version of the webinos platform, the three specification documents will be updated to reflect the experiences with the platform implementation and the application development on that platform, as well as possible additions required due to technological advances of the underlying hardware.

The updated version of this specification (webinos phase II specification, deliverables 3.3, 3.4 and 3.6) will be released in mid-2012.

7. Glossary

The following definitions are used throughout the discussion and specification of webinos. The following list of terms is not exhaustive. Other terms may be used and will be defined and added throughout the project work.

Some definitions will originate in certain domains and may overlap. So, a "Feature [WAC]" might be different from an automotive "Feature".

Definitions of Stakeholders

User (USR)

... of Webinos Apps

- Discovers availability of application
- Select Application for use and installation (including preconditional commercial steps)
- Configures application for his personal preferences
- Uses application to perform intended actions
- Share knowledge about application availability with other users (e.g. invite/encourage to install)

Developer of Webinos Apps (DEV)

- Create software packages
- Develop application programs using various programming languages and toolkits (DIE, SDK, etc…)

Application Service Provider of Installable Webinos Apps (ASP)

- Offers applications for consumption (by means of installation) on devices under a business model of their choice (e.g. retail, direct, free, etc.)
- Provides application lifecycle management
- Manages application developer relations

Application Service Provider of hosted Webinos Apps (ASP-HA)

- runs Webinos Applications „in the Cloud“ on behalf of a Webinos Application User
- offer hosted instances of Webinos platform components

Device Manufacturer (DMA)

- offers execution environment for the Webinos runtime
- supports Webinos discovery functionality for devices and applications
- demands Webinos to add tangible user benefit by means of making Webinos applications available on their devices

Network provider (NET)

... of Internet Connectivity (not local / short-range connectivity)

- provides and manages access to the internet for users as well as for application service providers

3rd Party Service Provider (3RD)

- Supplies functionality outside of the Webinos platform and functions set to Webinos App Developers, that they can use in their Apps

Developer of Webinos Platform (DWP)

- creates functional specification and reference implementation of the Webinos Platform
- provides developer community support where applications developers are engaged
  Simpler: That's us - the Webinos project team

General Definitions

Access Control

Protection of resources against unauthorized access; a process by which use of resources is regulated according to a security policy and is permitted by only authorized system entities according to that policy. [RFC 2828]

Access Control List (ACL)

A mechanism that implements access control for a system resource by enumerating the identities of the system entities that are permitted to access the resources. [IEC 62351-2, ed. 1.0 (2008-08)]

Access Management

The management of policies, rules, processes and information to control access to certain resources. In particular, the collection of systems and/or services associated with specific on-line resources and/or services that together derive the decision (privileges) about whether to allow a given entity to gain access to those resources or make use of those services [STORK Glossary and Acronyms]

Account

A set of data, tools and functionalities, that can be accessed after the successful accomplishment of the processes of Authentication and Authorization. It allows for accountability.

Accountability

The property of a system (including all of its system resources) that ensures that the actions of a system entity may be traced uniquely to that entity, which can be held responsible for its actions. [RFC 2828]

Actuator

A mechanical device for moving or controlling a mechanism or system. It is operated by a source of energy, usually in the form of an electric current, hydraulic fluid pressure or pneumatic pressure, and converts that energy into some kind of motion. [Wikipedia]. Webinos must provide methods that allow Web Applications to control Actuators.

Advertising

the process of public promotion (of some product or service) [http://wordnetweb.princeton.edu/perl/webwn?s=advertising]

Anonymity

"Anonymity requires that other users or subjects are unable to determine the identity of a user bound to a subject or operation" (BS ISO/IEC 15408-2:2008)

Anonymous

The condition of having a name that is unknown or concealed. [RFC 2828]

Application installation

Installation (or setup) of a program (including drivers, plugins, etc.) is the act of putting the program onto a computer system so that it can be executed. Because the requisite process varies for each program and each computer, many programs (including operating systems) come with a general-purpose or dedicated installer – a specialized program which automates most of the work required for their installation.

Application lifecycle

A continuous process of managing the life of an application through governance, development and maintenance.

Application package

see Installable Web Application

Application Programming Interface (API)

An interface, i.e. a point of interaction between components, implemented by a software program that enables it to interact with other software. It facilitates interaction between different software programs similar to the way the user interface facilitates interaction between humans and computers.
[Source: Wikipedia]

Applications

"A complete, self-contained program that performs a specific function directly for the user. This is in contrast to system software such as the operating system kernel, server processes, libraries which exists to support application programs and utility programs." -- http://foldoc.org/application+program

Applications may exist on a device or be hosted online.

AppStore
An overall set of client and/or server functions that may enable various capabilities of the user related to the application lifecycle, e.g. widget discovery, selection, preview, payment, authorization checks, update, etc. [Source: WAC]

AppStore Client
A device client that supports client-side functions of an AppStore. [Source: WAC]

AppStore Server
A network server that supports server-side functions of an AppStore. [Source: WAC]

Attack
An assault on system security that derives from an intelligent threat, i.e., an intelligent act that is a deliberate attempt (especially in the sense of a method or technique) to evade security services and violate the security policy of a system. [RFC 2828]

Attacker
The originator of an attack.

Attestation
The process of vouching for the accuracy of information. External entities can attest to shielded locations, protected capabilities, and Roots of Trust. A platform can attest to its description of platform characteristics that affect the integrity (trustworthiness) of a platform. Both forms of attestation require reliable evidence of the attesting entity. [http://www.trustedcomputinggroup.org/developers/glossary/]

Attestation of the Platform
An operation that provides proof of a set of the platform's integrity measurements [http://www.trustedcomputinggroup.org/developers/glossary/]

Authentication
"Authentication is the process of confirming the correctness of the claimed identity." -- http://www.sans.org/security-resources/glossary-of-terms/

Authentication Mechanism
The mechanism used to corroborate claimed information with a specified, or understood, level of confidence. Mechanisms include the use of username/password, X.509 client certificates and biometrics, etc. [STORK Glossary and Acronyms]

Authentication Server

A server that provides authentication services to users or other systems. For example, the user passes its identity and password (or certificate, smartcard, biometric data) to the authentication server; the latter verifies this data and grants the authentication proof (e.g., a credential) to the user.

Authenticity

A property that ensures that the identity of a subject or resource is the one claimed. Authenticity applies to entities such as users, processes, systems and information [IEC 62210, ed. 1.0 (2003-05)]

Authorization

A right or a permission that is granted to a system entity to access a system resource. [RFC 2828]

Authorization Server

A server that consults the security policy, extracts the relevant security rules, evaluates these rules with the current access parameters, eventually, invokes the conflict resolution process, and generates the corresponding credentials that permit the access to resources.

Availability

"The property of being accessible and usable upon demand by an authorized entity" (ISO/IEC 13335-1:2004 taken from BS ISO/IEC 27001:2005)

Certification Authority (CA)

- A trusted third party 'clearing house' that issues digital signatures and digital certificates. [ISO/IEC 27002:2005]
- An infrastructure that issues certificates signed by or chained to a root certificate owned by the organization operating the CA infrastructure. CA-operating organizations typically own multiple root certificates, and apply various certification practices, e.g. level of business or domain validation required for issuing certificates signed by a particular root certificate. [Source: WAC]

Certificate Store

A certificate store will often have numerous certificates stored, possibly issued from a number of a different certification authorities. It consists of key pairs that encrypt and decrypt the symmetric key used for encrypting and decrypting data by Encrypting File System, digital certificates and so on.

Child application

A child application is a self contained application package that is included within another application package (parent application). The are used as means for distributed application development and deployment where a parent application can export child applications to other devices in order to provide a service that can be distributed over several devices to the user.

Claim

A statement made by one entity about itself or another entity that a relying party considers to be “in doubt” until it passes “Claims Approval”. [STORK Glossary and Acronyms]

Cloud

"Cloud computing is Web-based processing, whereby shared resources, software, and information are provided to computers and other devices (such as smartphones) on demand over the Internet." (http://en.wikipedia.org/wiki/Cloud_computing)

"The Cloud" can therefore refer to a range of services provided by remote infrastructures. A single instance of a Cloud is Amazon EC2, for example.

Cloud service

Cloud application services or "Software as a Service (SaaS)" deliver software as a service over the Internet, eliminating the need to install and run the application on the customer's own computers and simplifying maintenance and support. People tend to use the terms ‘SaaS’ and ‘cloud service’ interchangeably, when in fact they are two different things.

Confidentiality

"The property that information is not made available or disclosed to unauthorized individuals, entities, or processes" (ISO/IEC 13335-1:2004 taken from BS ISO/IEC 27001:2005)

Content adaptation

Is the action of transforming content to adapt to device capabilities. Content adaptation is usually related to mobile devices that require special handling because of their limited computational power, small screen size and constrained keyboard functionality.

Context

Schilit et al. (1994) define context as where you are, who you are with, and what resources are nearby. This might suggest that context is more focused on the user’s surrounding as opposed to his/her inner states. A more specific definition is provided by Dey et al(2001). They defined it as any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.. Some examples of context types according to the actor/entity that is involved are: User context (contexts describing a user’s situation), software context (context describing a software’s situation).
Context is more or less the set of facts or circumstances that surround a situation or event.

Context Acquisition

the process of acquiring context data that describe aspects of a situation. This is usually done through the identification of a corresponding occurring event.

Context awareness

A system is context aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task. Three important context-awareness behaviours that an application might exhibit can be identified:

• The presentation of information and services to a user
• The automatic execution of a service
• The tagging of context to information for later retrieval.

Ref: Dey & Abowd, David R. Morse & Stephten Armstrong & Anind K. Dey

Context Model

The conceptualization of how context data (within webinos) relate to determining factors such as situations, events and entities

Context Objects

A dedicated objects that holds context data about a corresponding webinos entity

Context of Use

Context of use refers to the "users, tasks, equipment, and the environments where a system is used" - ISO/IEC 13407

Context Sources

Context information can be found in a variety of sources. These can be as follows:

• Features in content items or the items themselves, these can include metadata about the content item
• Index terms
• Structure of a subject area as represented by a classification scheme, thesaurus or ontology
• User searches, user logs (containing queries, click behavior, timings and so on)
• Comments on content items (from individual users, groups, experts etc)
• Sensor information about the environment (location, temperature, time, etc)
• Actuator information such as a remote control mechanism or a switch that records its state when someone has pressed it
• User activities and user generated content over social media platforms

Context structure

Context structure refers to the schema and the organization of the factors characterizing a specific event. This event consists of information decomposed to the primitive substances, that combined together give a consistent body of the structure. In the information technology dialect the event may refer to a specific task, situation or a goal of the system. The corresponding information describing the system is the input mostly coming from various sensors, ontologies, function of task and related data.

The structure is designed to enable effective matching and retrieval of contexts. The user context structure consists of five subcontexts:

• Environment context (it captures the entities that surround the user. These entities may be things,

services, temperature, light, humidity, noise and persons)

• Personal context. It consists of two sub contexts: the physiological context (pulse, blood pressure, weight, hair colour etc) and mental context (mood, interests, anger, stress etc)

• Task context. This context describes what people are doing. The task context can be described with explicit goals, tasks, actions, activities, ore events.

• Social context

• Spatio-temporal context. This context aspect describes aspects of the user context relating to the time and spatial location (time, location, direction, speed, shape of buildings etc)

Credential

Data that is transferred or presented to establish either a claimed identity or the authorizations of a system entity. [RFC 2828]

Cross-Device Application

A webinos application capable of running on multiple devices, such as a set-top box and smartphone, not necessarily simultaneously.

Delivery Notification Event

An event that indicates the outcome of the delivery of another event, according to the "Delivery notifications" part of the "Messaging" section in the Webinos System Specifications.

Device

A piece of hardware such as a mobile telephone, laptop, PC, home media centre, in-car system, netbook, tablet, game console, or router. The term “terminal” may be used analogue.

Device Cloud

In the context of webinos, a set of devices grouped by some criteria (proximity, user, complementing features). Synonymous with webinos cloud.

Digital Certificate

A structure that associates an identity with an entity such as a user, a product or an Application Instance where the certificate has an associated asymmetric key pair which can be used to authenticate that the entity does, indeed, possess the Private Key (IEC 62541-2, ed. 1.0 (2010-02))

Digital Identity

A set of claims made by one digital subject about itself or another digital subject." Where a digital subject is "a person or thing represented or existing in the digital realm which is being described or dealt with. -- The Laws of Identity

Digital Signature

The result of a cryptographic transformation of data which, when properly implemented, provides the services of:

- origin authentication
- data integrity
- signer non-repudiation. [FIPS 140-2]

Direct Anonymous Attestation (DAA)

A protocol for vouching for an AIK using zero-knowledge-proof technology. [http://www.trustedcomputinggroup.org/developers/glossary/]

Domain Name System Security Extensions (DNSSEC)

is a suite of Internet Engineering Task Force (IETF) specifications for securing certain kinds of information provided by the Domain Name System (DNS) [Wikipedia]

Distributed Hash Table (DHT)

A distributed data structure used to perform a look up service similar to a hash table, where a key is mapped to a value. The key is first mapped to a number in a range. Network nodes are responsible for maintaining the mapping from numbers to values for different parts of the range. Distributed Hash Tables form an attractive solution to mapping user identities to the URLs for personal zone hubs, where the search is distributed across the hubs in a federated process.

Electronic Identity

see Digital identity

Entity

Any concrete or abstract object of interest, including relations among things (IEC 61360-1, ed. 3.0 (2009-07))

Event

A set of structured data that is meant to be exchanged among two or more addressable entities (e.g., applications, services).

Event dispatcher

Logical unit within the WRT in charge of routing events among local and/or remote addressable entities.

Extension

A collection of data, APIs and/or platform-specific code, that is not part of webinos but is meant to be used in conjunction with it and is consistent with webinos APIs (use case: WOS-UC-TA3-004: Embedding Proprietary Extensions).

The terms browser addon and browser extension are generally used for third party modules that are run in a separate security context from Web pages, and have elevated privileges. Such extensions are typically implemented as a mix of JavaScript, markup, style sheets and other resources, and have access to browser specific APIs. They are designed to work with a specific browser, e.g. Firefox, but are typically independent of the underlying operating system.

Feature

An option or functional capability available to the user.

Geo-location

Geo-Location is an identification of the real-world geographic location of an object, such as a cell phone or an Internet-connected computer terminal. Geo-location may refer to the practice of assessing the location, or to the actual assessed location.

Group signature

A method for allowing a member of a group to anonymously sign a message on behalf of the group.

Idemix

An anonymous credential system developed by IBM Research and based upon zero knowledge proofs. Users are issued strong credentials. Idemix can then be used to prove that a credential has certain properties without disclosing anything else about the owner of that credential. This provides effective authentication and good privacy based upon strong credentials.

Identification

An act or process that presents an identifier to a system so that the system can recognize a system entity and distinguish it from other entities. [IEC 62351-2, ed. 1.0 (2008-08)]

Identity

Informally, identity "is whom someone or what something is, for example, the name by which something is known." -- SANS

A digital identity is "a set of claims made by one digital subject about itself or another digital subject." Where a digital subject is "a person or thing represented or existing in the digital realm which is being described or dealt with". -- The Laws of Identity

Identity Certificate Response Token

Token be an encrypted and encoded identity certificate. It is returned from the Privacy Certificate Authority (A trusted third party that issues platform identities) to the platform.

Identity Certificate Request Token

Token placed in Identity Credential Request element when Requesting a identity certificate.

Identity token

An identity token identifies an entity. For example, it can be a cryptographic key or a certificate. In most cases, additional knowledge is needed to be able to validate the token, and as such to authenticate the entity which identifies by this token. In case of the key, the validating entity needs to know the key. In case of the certificate, the validating entity needs to be able to validate the entire certificate chain.

Infotainment

It is a “information-based media content or programming that also includes entertainment content in an effort to enhance popularity with audiences and consumers.”

Installable Web Application

A Web Application that is packaged in a way to allow a single download and installation on a user's device. The installed package could contain the complete application, i.e. the html, css and JavaScript files as well the "manifest" file or the installed package could contain the "manifest" file only. The "manisfest" file contains metadata describing the app. "Installed Web Applications" can for example benefit from digital signing of package and that API access control decisions could be done at installation time. Furthermore there might be marketing/deployment/charging advantages with "Installed Web Applications. Examples of Installable Web Applications are W3C Widgets and Chrome Installable Web Apps (http://code.google.com/chrome/apps/docs/index.html).

Integrity

"The property of safeguarding the accuracy and completeness of assets" (ISO/IEC 13335-1:2004 taken from BS ISO/IEC 27001:2005)

JavaScript API

An API for Web Applications defined using an Interface Definition Language (IDL). JavaScript APIs could for example be provided as a means for a Web Application to gain access to Device Capabilities. The definition of the API itself concerns the interfaces, methods, properties and other attributes that make up the API.

Local Storage

A memory for recording (storing) information (data) entirely located on a specific device

Launcher

The part of the webinos runtime responsible for loading and executing applications which may not be running or be in an inactive move.

Manifest file

see Installable Web Apps

Metadata

"Structured data about data. Increasingly this term refers to any data used to aid the identification, description and location of networked electronic resources." -- SWDB

Mutual authentication

A mechanism whereby two parties can verify each other's identity, e.g. a website can authenticate a user, and the user can authenticate the website. This provides mitigation against spoofed sites, spoofed DNS and spoofed IP addresses.

Namespace

XML namespaces are used for providing uniquely named elements and attributes in an XML document. An XML instance may contain element or attribute names from more than one XML vocabulary, e.g., the W3C Widget vocabulary and the added webinos specific attributes.

Node

In XMPP Service Discovery mechanism, a node is a specific service/device which is communicable or about which information can be obtained. There are two types of nodes in XMPP: Info Nodes which are directly addressable and Items Nodes which are not addressable directly. Info Nodes are connected over IP and could be directly communicated using its full JID. Item nodes information is retrieved using Info Nodes. Details about the communication part to item nodes is not covered in XMPP standard, only information about fetching information about item nodes is specified.

Optional extension

An extension that, if not available, still allows the application to run, yet with less and/or degraded functionality.

Optional feature

A feature that, if not available, still allows the application to run, yet with less and/or degraded functionality.

Parent application

A parent application is an application that contains child applications within its application package. See: child application.

Personal Service

This is a service that runs as part of the user's personal zone, and may be exposed to others via the personal zone hub, subject to the user's preferences. The user can install new personal services (provided by 3rd parties) in addition to the ones that are provided directly by the webinos platform.

Personal Zone (PZ)

This provides the means for users to manage their personal devices and services. The zone is exposed on each device as a set of local APIs, with a shared model of context that is synchronized across all devices in the zone together with the personal zone hub. The personal zone supports a single sign on mechanism so that users authenticate to a device, the device to the zone, and the zone to applications/services. The zone further supports an overlay networking model that hides the details of addressing and interconnect technologies. This is based upon 3rd party components that enable the zone to scale in the face on a continuing evolution of networking technologies. Note that some devices (e.g. a TV or networked printer) may be shared by several people, and forms part of a shared zone.

Personal Zone Hub (PZH)

The Personal Zone Hub runs on a Web server with a public URL, and provides the means for people to access your devices and services subject to your preferences. It supports setting up logical peer to peer connections across Firewall/NAT boundaries.

Personal Zone Proxy (PZP)

A webinos-enabled device supports the zone apis (see personal zone) that enable it to function as part of the user's personal zone. Other non-webinos devices may be connected through webinos-enabled devices, which act as personal zone proxies.

Personally identifiable information

As used in information security, refers to information that can be used to uniquely identify, contact, or locate a single person or can be used with other sources to uniquely identify a single individual.

Platform

An operating system and software environment running on top of a device, managing the device hardware and providing common services for efficient execution of various application software.

Examples: The most common operating systems used in mobile smart phones include Symbian OS, Android, iOS, RIM Black Berry OS, or Windows Mobile. Popular modern operating systems for personal computers include Microsoft Windows, Mac OS X, and GNU/Linux. MeeGo is a Linux-based open source mobile operating system project, primarily targeted at mobile devices and information appliances in the consumer electronics market.

Plugin

see Extension

Policy

Abstractly, a policy is a rule that defines a choice in the behaviour of a system . Security policies in Webinos are the permissions defined by users of the webinos runtime in order to allow or deny access

to a resource or capability by an application or component. For example, a policy may define that Application A may not use the GPS location capabilities on a device.

A "Privacy Policy" or "Privacy Manifest" is a statement given by an application defining how the application will use information about (or belonging to) the user. For example, a privacy policy might assert that user contact details are not passed on to any third parties.

Policy Administration Point (PAP)

Point which manages policies. -- Wikipedia XACML

Policy Decision Point (PDP)

Point which evaluates and issues authorization decisions. -- Wikipedia XACML

Policy Enforcement Point (PEP)

Point which intercepts user's access request to a resource and enforces PDP's decision. -- Wikipedia XACML

Policy Information Point (PIP)

Point which can provide external information to a PDP, such as LDAP attribute information. -- Wikipedia XACML

Policy Manager

A component on the webinos runtime capable of editing and viewing security policies based on interaction with the user.

Policy Synchronization

The process of synchronising policies between webinos devices, to maintain a common set of security policies. This involves making sure every platform has the same set of XACML policy files dictating how applications and users are allowed to use the runtime. Synchronisation primarily occurs between a device's personal zone proxy and the personal zone hub.

Privacy

The right of individuals to control or influence what information related to them may be collected and stored and by whom and to whom that information may be disclosed.[ISO 7498-2: Security Architecture]

Privilege

Right or permission expressly granted to a single or specified group of user(s) or device(s) to perform specified actions, in specified roles and associated to established identity [IEC/PAS 62443-3, ed. 1.0 (2008-01)]

Privilege Management

The management of applications on the runtime, stating which ones are given "privileged" status and therefore access to non-public webinos APIs.

Reliability

The ability of a system to perform a required function under stated conditions for a specified period of time. [RFC 2828]

Remote Data Binding Protocol

The Remote Data Binding Protocol ensures that the tickets are encrypted by a key that is known only to the ticket issuer.

Required feature

A feature that is absolutely needed for an application to run.

Revocation

Allows a Mobile Operator or device owner to block a specific application or group of applications.

Risk

The level of impact on agency operations (including mission, functions, image, or reputation), agency assets, or individuals resulting from the operation of an information system, given the potential impact of a threat and the likelihood of that threat occurring. [NIST SP 800-30]

Risk Analysis

(see also Risk Assessment) Systematic use of available information to identify hazards and to estimate the risk [ISO/IEC Guide 51, Definition 3.10]

Risk Assessment

The process of identifying risks to agency operations (including mission, functions, image, or reputation), agency assets, or individuals by determining the probability of occurrence, the resulting impact, and additional security controls that would mitigate this impact. Part of risk management, synonymous with risk analysis, and incorporates threat and vulnerability analyses. [NIST SP 800-53]

Role Based Access Control (RBAC)

A form of identity-based access control where the system entities that are identified and controlled are functional positions in an organization or process. [IEC 62351-2, ed. 1.0 (2008-08)]

RPC Request event

An event that contains a JSON-RPC 2.0 request object or request batch, according to the "Delivery notifications" part of the "Messaging" section in the Webinos System Specifications.

RPC Response event

An event that contains a JSON-RPC 2.0 response object or response batch, according to the "Delivery notifications" part of the "Messaging" section in the Webinos System Specifications.

Security

All aspects related to defining, achieving, and maintaining confidentiality, integrity, availability, non-repudiation, accountability, authenticity, and reliability. [ISO/IEC 13335-1]

Security policy

- A set of rules and practices that specify or regulate how a system or organization provides security services to protect sensitive and critical system resources. [RFC 2828].
- A collection of access control constraints, abstractly representable as a policy-set and supported by the operating system or execution environment, that describes the circumstances under which Web Applications are permitted to access Features and underlying Device Capabilities.

Security Policy Enforcement

The ability to understand and apply security policies

Security Mechanisms

A process (or a device incorporating such a process) that can be used in a system to implement a security service that is provided by or within the system. Some examples of security mechanisms are authentication exchange, checksum, digital signature, encryption, and traffic padding.[RFC 2828]

Security Model

- A schematic description of a set of entities and relationships by which a specified set of security services are provided by or within a system.[RFC 2828]
- Generally, a security model is a "formal system" used to specify and reason on the security policy (i.e., it is used as a basis for formal specification proofs). It is thus intended to abstract the security policy and handle its complexity. It represents the secure states of a system as well as the way in which the system may evolve. It also allows for verifying the consistency of the security policy, and for detecting and resolving possible policy conflicts.

Security Service

A processing or communication service that is provided by a system to give a specific kind of protection to system resources.[RFC 2828]

Set-top box

Is a device that connects to a television and an external source of signal, turning the signal into content which is then displayed on the television screen or other display device.

Sensor

A device that measures a physical quantity and converts it into a signal which can be read by an observer or by an instrument. [Wikipedia]. Webinos must provide methods for Web Applications to get access to sensor data.

Sensor network

Sensor network consists of spatially distributed autonomous sensors to cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants.

Service

An addressable logical functional entity that is exposed on a device or a server. The service availability varies over time depending on the device status (on/off), service status (stopped/running/installed/uninstalled) and connection status (connected/disconnected). Examples of services are: an API to get the temperature from a sensor, a Web API provided by a Web Server or a remote control API provided by a TV.

Service Discovery

The procedure that provides the means to detect devices and services provided in a Webinos network or devices and services connected via short range bearers. The procedure includes the means:

- for a service to advertise its availability and service capabilities.
- for an application to search for available services.

Short Range Bearer:

Technology used to established a direct peer to peer connection for exchanging data between one or many devices over a short distance such as Bluetooth, ANT+, NFC, Zigbee, Wireless USB, UWB or WiFi Direct.

Single Sign-Off

Single sign-off is the property whereby a single action of signing out terminates access to multiple software systems. [wikipedia - http://en.wikipedia.org/wiki/Single_sign-on]

Single Sign-On

It is a property of access control of multiple, related, but independent software systems. With this property a user logs in once and gains access to all systems without being prompted to log in again at each of them [wikipedia]

Smartphone

A smartphone is a mobile phone that offers more advanced computing ability and connectivity than a contemporary basic feature phone. Smartphones and feature phones may be thought of as handheld

computers integrated within a mobile telephone, but while most feature phones are able to run applications based on platforms such as Java ME, a smartphone allows the user to install and run more advanced applications based on a specific platform. Smartphones run complete operating system software providing a platform for application developers.
[Source: Wikipedia.]

Social community site

A Web site that focuses on building and reflecting of social networks or social relations among people, e.g., who share interests and/or activities. A social network service essentially consists of a representation of each user (often a profile), his/her social links, and a variety of additional services.

Social Context

The term has sociological origins and refers to both people and organizations that endue a person. Social Context is the aggregation of social circles where somebody lives, interacts with others and develops his activities, beliefs and social intelligence. A person's social context includes both the personal social context (his friends and the communities he belongs to) and his community social context (their role and identity in different communities).
In other words, social context describes the social aspects of the current user context. It can contain information about friends, neighbours, co-workers, relatives, and their presence. One important aspect in a social context is the role that the user plays in the context. A role can for instance be described with a name, the user's status in this role.
From a technological aspect, it is a mathematical graph-instance of the real social context of a user. After the development of the social Web, the term has gained great exposure, and refers to data that state – explicitly or implicitly – the connection among users. These connections may have the following elements:
• The friends and family (relationships) — the augmented or live address book
• Social information, such as events, groups, calendars, locations
• Social Objects that connect users, such as photos, videos, comments, likes
• Social activities — find friends/family, invite, share content, tag users, rate
This information can be used to construct a set of egocentric social networks in which a user is at the center of a set of relationships with others, who may also have ties to one another.
Ref: Leveraging Social Context for Searching Social Media, Marc Smith, Vladimir Barash, Lise Getoor, Hady W. Lauw

Social Graph

The collections of social connections that someone maintains in an online social platform (or in any platform that supports the creation of explicit or implicit connections among users)

Social media

Social media are media for social interaction, using highly accessible and scalable publishing techniques. Social media uses web-based technologies to turn communication into interactive dialogues.

Social Proximity

Social Proximity is the phenomenon of overlapping for different people's social circles. That helps them to reinforce the deep bonds of trust that facilitate exchange of tacit knowledge, and thus to develop more powerful connection among them. Social proximity is actually a very schematic and physical indicator. It means a person is in touch with others.

Sub-node

A node can have sub-hierarchy of nodes underneath it. A node under a node is referred as sub-node.

Threat

Any circumstance or event with the potential to adversely impact agency operations (including mission, functions, image, or reputation), agency assets, or individuals through an information system via unauthorized access, destruction, disclosure, modification of information, and/or denial of service. [NIST SP 800-53]

Transcoding

The direct digital-to-digital conversion of one encoding to another.

Trusted Application

An application which has been assigned a specific level of trust, which can be based upon a variety of application attributes and installation context attributes, as defined by a security policy. [Source: WAC]

Trusted Entity

Entity which is assumed to appropriately enforce security policies. Because of this assumption, the entity may cause other security policies to be obviated.
For example: A trusted authorisation entity declares a user to be authorised for control thereby challenges authentication procedures, that would normally be applied, are not invoked. [IEC 62210, ed. 1.0 (2003-05)]

Unlinkability

"Unlinkability requires that users and/or subjects are unable to determine whether the same user caused certain specific operations." (BS ISO/IEC 15408-2:2008)

Unrecognised

An application is considered as "unrecognised" if it does not carry a signature belonging to a defined and recognised identity.

Untrusted Application

An application is considered untrusted if it has not been assigned any specific level of trust, and is thus assigned to a default level of trust. [Source: WAC]

User profile

It is a collection of personal data associated to a specific user. A profile refers therefore to the explicit digital representation of a person's identity. A user profile can also be considered as the computer representation of a user model.

Vulnerability

A flaw or weakness in a system's design, implementation, or operation and management that could be exploited to violate the system's integrity or security policy. [RFC 2828]

WAC

The Wholesale Applications Community (WAC) is an open global alliance formed from the world's leading telecoms operators. It has been established to increase the overall market for mobile applications. It strives to unite a fragmented applications marketplace by providing common developer tools, a revenue share model across the entire ecosystem and common network and terminal APIs. For more information, see http://www.wholesaleappcommunity.com.

For further information related to the latest WAC Waikiki Beta Releases, see also http://members.wholesaleappcommunity.com/corespec/spec.html#toc-definitions.

Web Analytics

The Official http://www.webanalyticsassociation.org/?page=aboutus Definition of Web Analytics: "Web Analytics is the measurement, collection, analysis and reporting of internet data for purposes of understanding and optimizing Web usage".

Web Application

The term used generically to refer to an application authored in Web formats, including HTML, JavaScript, CSS and various media formats. [Source: WAC]

Web Browser

A terminal application which provides a Web Runtime Environment supporting Websites. [Source: WAC]

Webinos

Webinos is the system we are developing to deliver the "one application for all devices" vision. It includes:

- A client-side software implementation providing functionality or APIs for displaying a user interface and using device features
- A standard describing this software implementation
- Developer tools (documentation, software) for writing applications that use the standard
- A shared set of protocols and standards for sharing applications, data and metadata between devices, as well as for providing a seamless user experience

- Privacy and security controls to enhance the user experience and protect against threats to user, applications and the system itself.

Webinos API

An API that exposes access to some internal functionality of a piece of software for use by Webinos developers, to get access to specific information, to trigger special behavior, or to perform some other action. The Webinos APIs are typically client-side script APIs, for use in Webinos browsers and similar Webinos user agents (as opposed to server-side APIs, for example).
[Inspired by W3C Web APPs WG's definition of API]

Webinos application

An application written using webinos technologies that will run on a device, across a range of devices reflecting the domains mobile, stationary devices, automotive or home media and/or server. The application will be able to securely and consistently access device specific features, communicate over the cloud and adjust to the device and context specific situation.

Webinos Component

A term to indicate any component augmented with webinos-based intelligence that interact with or manage a Webinos Network (i.e. Webinos Agent, Webinos Device, Webinos Runtime)

Webinos Consortia

The webinos project has been defined by a strong consortium of 22 founding partners from nine countries. These companies are academic and industrial with a cross-domain focus; they include mobile phone manufacturers, telecommunication operators, automotive company, research and analytics firm, a communication company and a range of well-known research facilities. It is the intention to engage further companies in the definition and adoption of webinos – as such the current webinos project members reach out to external companies via a range of communication tools (website, conferences, etc.)

Webinos project

The webinos project refers to the commitment of the 22 webinos founding members to collaborate, define and deliver webinos within the three year time period following the official kick-off in September 2010.

Web technology

Commonly associated with Web applications that facilitate interactive systemic biases, interoperability, user-centered design, and developing the World Wide Web.

Website

A remotely hosted collection of resources authored in Web formats (including HTML, JavaScript, CSS and various media formats) and served by a Web server so as to be viewable in a Browser. [Source: WAC]

Widget

An interactive application for displaying and/or updating local data or data on the Web, packaged in a way to allow a single download and installation on a user's machine or mobile device. [Source: WAC]

Web Runtime Environment (WRT)

A WRT is considered as terminal software which supports the execution of Web applications.

Zero Knowledge Proof

An interactive method for one party to prove to another that a give statement is true without revealing anything else. This can be used for privacy friendly authentication where one party proves to another certain properties, such as membership of a named group, or an age range, without disclosing their identity or actual age. See Idemix for a library implementing zero knowledge proofs.

Acronyms

Acronym	Definition
ACL	Access Control List
API	Application Programming Interface
CA	Certification Authority
DAC	Device API and Policy (W3C)
DLNA	Digital Living Network Alliance
DNS	Domain Name Service
IETF	Internet Engineering Task Force
IDL	Interface Definiton Language
JID	Jabber ID
LTE	Long Term Evolution
MNO	mobile Network Operator
mDNS	multicast DNS
NAS	Network Attached Storage
NPAPI	Netscape Plugin API
OEM	Original Equipment Manufacturer
PAP	Policy Administration Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point

PII	Personally Identifiable Information
PIP	Policy Information Point
POS	Point of Sale
PWN	Personal Webinos Network
WAA	Web Analytics Association
WAC	The Wholesale Applications Community
WRT	Web Runtime Environment
RBAC	Role-Based Access Control
SSO	Single Sign-On
TLS	Transport Layer Security
TTS	Text to speech
UPnP	Universal Plug and Play
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
VoD	Video-on-demand
XMPP	Extensible Messaging and Presence Protocol

8. Resources

Web Applications

[CRX] http://code.google.com/chrome/apps/docs/developers_guide.html
[OpenWebApps] https://developer.mozilla.org/en/OpenWebApps
[AppCache] http://dev.w3.org/html5/spec/offline.html
[W3CWidgets] http://www.w3.org/2008/webapps/wiki/WidgetSpecs
[W3CWidgetFamily] http://www.w3.org/2008/webapps/wiki/WidgetSpecs
[OperaWidgets] http://dev.opera.com/sdk/
[Widgets] http://www.w3.org/TR/widgets/
[WidgetAPI] http://www.w3.org/TR/2011/WD-widgets-apis-20110607/
[WidgetUpdate] http://www.w3.org/TR/2010/WD-widgets-updates-20100928/
[WidgetURI] http://www.w3.org/TR/2009/WD-widgets-uri-20091008/
[WidgetDigSig] http://www.w3.org/TR/2011/WD-widgets-digsig-20110607/
[WARP] http://www.w3.org/TR/2010/CR-widgets-access-20100420/
[WAC] http://specs.wacapps.net/2.0/jun2011/
[WACWebTech] http://specs.wacapps.net/2.0/jun2011/core/web-standards.html

Extensions and Plug-ins

[Chrome-NPAPI] http://code.google.com/chrome/extensions/npapi.html
[HP-PDK] https://developer.palm.com/content/api/dev-guide/pdk/overview.html
[FF-Addons] https://addons.mozilla.org/en-US/firefox/browse/type:7
[FireBreath] http://www.firebreath.org/display/documentation/FireBreath+Home
[Luce] http://www.rawmaterialsoftware.com/juce.php
[npruntime] https://developer.mozilla.org/en/Gecko_Plugin_API_Reference/Scripting_plugins
[npapi-browser-side-api]
https://developer.mozilla.org/en/Gecko_Plugin_API_Reference%3aBrowser_Side_Plug-in_API
[npapi-plugin-side-api] https://developer.mozilla.org/en/Gecko_Plugin_API_Reference/Plug-in_Side_Plug-in_API
[NaCl] http://code.google.com/p/nativeclient/
[PPAPI] http://code.google.com/p/ppapi/
[moz-pepper] https://wiki.mozilla.org/NPAPI:Pepper
[node.js] http://nodejs.org/docs/v0.4.8/api/addons.html
[js-ctypes] https://developer.mozilla.org/en/JavaScript_code_modules/ctypes.jsm
[using-js-ctypes] https://developer.mozilla.org/en/js-ctypes/Using_js-ctypes
[Npruntime] Mozilla Developer Network: Scripting plugins
https://developer.mozilla.org/en/Gecko_Plugin_API_Reference/Scripting_plugins
[NPAPI-Browser-Side-API] Mozilla Developer Network: Browser Side Plug-in API
https://developer.mozilla.org/en/Gecko_Plugin_API_Reference%3aBrowser_Side_Plug-in_API

[NPAPI-Plugin-Side-API] Mozilla Developer Network: Plug-in Side Plug-in API https://developer.mozilla.org/en/Gecko_Plugin_API_Reference/Plug-in_Side_Plug-in_API

Authentication

[OpenID] Open standard for authenticating users, Home page, http://openid.net/
[OAuth] Open protocol to allow secure API authorization, Home page, http://oauth.net/
[Liberty] Liberty Alliance - consortium for developing a distributed identity management system, Home page, http://www.projectliberty.org/
[Kantara] Initiative to help ensure secure, identity-based, online interactions, Home page, http://kantarainitiative.org/
[SAML] Security Assertion Markup Language standard suite v2.0, Home page, http://www.oasis-open.org/standards#samlv2.0
[Shibboleth] Standards based, open source software package for Web single sign-on, Home page, http://shibboleth.internet2.edu/
[Kerberos] The Kerberos Network Authentication Protocol, home page, http://web.mit.edu/kerberos/
[XMPP] Extensible Messaging and Presence Protocol, Request for Comment 3920, http://tools.ietf.org/html/rfc3920
[SASL] Simple Authentication and Security Layer, Request for Comment 4422, http://tools.ietf.org/html/rfc4422
[IdMetasystem] Information Card wikipage, http://en.wikipedia.org/wiki/Information_Card
[FirefoxAM] Firefox Account Manager, Mozilla Hacks article "Account Manager coming to Firefox", http://hacks.mozilla.org/2010/04/account-manager-coming-to-firefox/
[WebID] WebID - Web Identification and Discovery, W3C Editor's Draft, http://www.w3.org/2005/Incubator/webid/spec/.
[Webinos-D3.2] Webinos Deliverable: D3.2 API Specifications 30 June 2011
[Webinos-D3.5] Webinos Deliverable: D3.5 Security Architecture

Discovery

[COHEN11] J. Cohen, S. Aggarwal, Y. Y. Goland General Event Notification Architecture Internet Draft, 2000.

Security

[ZHOU11] Yajin Zhou, Xinwen Zhang, Xuxian Jiang and Vincent W. Freeh Taming Information-Stealing Smartphone Applications to appear in the proceedings of TRUST 2011: The 4th International Conference on Trust and Trustworthy Computing, June 2011, Springer Berlin.
[DEME11] Matt Demers CyanogenMod Adds Support For Revoking App Permissions
[BICH11] Patrik Bichsel, Dave Raggett and Rigo Wenning Web authentication is deeply flawed, and it is time to fix it presented at the W3C Workshop on Identity in the Browser, May 2011.

[OWASP10] Jeff Williams and Dave Wichers OWASP Top 10 Application Security Risks - 2010. Risk A1: Injection
[D027-Ethan] Ethan Attacker Persona , From D2.7: User expectations on Security and Privacy - Webinos Project Deliverable , February 2011.
[D027-Frankie] Frankie Attacker Persona , From D2.7: User expectations on Security and Privacy - Webinos Project Deliverable , February 2011.
[D027-Jimmy] Jimmy Assumption Persona , From D2.7: User expectations on Security and Privacy - Webinos Project Deliverable , February 2011.
[D027-Helen] Helen Assumption Persona , From D2.7: User expectations on Security and Privacy - Webinos Project Deliverable , February 2011.
[D035] D3.5: Security Architecture - Webinos Project Deliverable , June 2011.
[WAC] WAC 2.0 Core Specification - Widget Security and Privacy , January 2011.
[BONDI] BONDI Architecture and Security Requirements , July 2009.
[XACML] eXtensible Access Control Markup Language (XACML) Version 2.0 , February 2005.
[ACSOAS] Access Control Service Oriented Architecture Security
[BONDIA&S] BONDI's Architecture and Security Appendices, January 2010
[D022] D2.2 Requirements & developer experience analysis, February 2011
[D035] D3.5: Security Architecture - Webinos Project Deliverable, June 2011.
[DAPWG] Device APIs and Policy Working Group
[OASISXACML] OASIS eXtensible Access Control Markup Language (XACML) TC
[PRIMELIFE] PrimeLife
[RFC3986] Uniform Resource Identifier (URI): Generic Syntax, January 2005
[SUNXACML] Sun's XACML Implementation
[WACCS] WAC Core Specification: Widget Security and Privacy, June 2011
[WACDS] WAC Device Specifications, June 2011
[WACXMLSP] WAC: XML definition of Security Policy

Glossary References

[WAA] http://www.webanalyticsassociation.org/?page=aboutus
[WAC] WAC Waikiki Beta Release Core Specification - Definitions
[BS ISO/IEC 15408-2:2008] Evaluation criteria for IT security (British Standards Online)
[BS ISO/IEC 27001:2005] Information security management systems (British Standards Online)
[BS EN ISO 13407:1999] Human-centred design processes for interactive systems (British Standards Online)
[SWDB] Statewide Database - the Redistricting Database for the State of California
[SANS] SANS Glossary of Security Terms
[FOLDOC] The Free Online Dictionary of Computing (Editor - Denis Howe)
[The Laws of Identity] The Laws of Identity by Kim Cameron, 5/12/2005
[PH10TERMINOLOGY] Andreas Pfitzmann and Marit Hansen. A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management v0.34, 10 Aug 2010. http://dud.inf.tu-dresden.de/literatur/

[RFC 2828] Internet Security Glossary, May 2000, http://www.ietf.org/rfc/rfc2828.txt
[NIST SP 800-53NIST] Special Pubblication 800-53 - Recommended Security Controls for Federal Information Systems and Organizations - Rev. 3, Aug 2009, http://csrc.nist.gov/publications/PubsSPs.html
[STORK]Glossary and Acronyms FP7 Project STORK (Secure Identity Across Borders Linked), Glossary and Acronyms, 10 Jul. 2009, https://www.eid-stork.eu/index.php?option=com_processes&Itemid=&act=streamDocument&did=615