Journal of Advances in Science and Technology

Vol. IV, No. VIII, February-2013, ISSN 2230-9659

REVIEW ARTICLE

STUDY OF API FOR WEB BASED OS

www.ignited.in

Journal of Advances in Science and Technology
Vol. IV, No. VIII, February-2013, ISSN 2230-9659

Study of API for Web Based OS

Ruchi Agarwal

Research Scholar, Pacific University, Udaipur, Rajasthan, India

----------♦----------

SYNCHRONIZATION

Synchronization is a way of easing the handling of web based operating systems based mobile phones or other applications like phone book contacts, calendar events, e-mails, and other such personal data. It also introduces a possibility for the users to back up data, by synchronizing information to a remote server. Some algorithms are only capable of fetching information from one or more sources and save it on the local device. Synchronization algorithms do, in addition to fetching information, also have the capabilities of pushing local changes back to the original source and have information on all clients being automatically updated.

There are different solutions to how synchronization should be handled in a WEB OS environment; some systems give the developers no or little additional functionality, while some provide a whole architecture with solutions to problems such as secure account handling, synchronization services and integration into the standard system.

EB OS

In web OS, synchronization is handled by services called Synergy Connectors. An implementation of a Synergy Connector has the capability of synchronizing contacts, calendar events, or messages. It uses an Account for accessing the information on a remote server. The procedure of having a system ac- count service will allow other services to get access to an account, and the information it is protecting, without having to compromise passwords or any other identification credentials. This could be achieved by, for example, providing an access ticket that only gives access to the information that should be accessed by the requesting service. The service may then use the other APIs to manage the respective data types; using the Contacts API for handling contact information, the Calendar API for event scheduling, and the Message API for synchronizing message data. A brief overview of how this works is showed in **Figure**

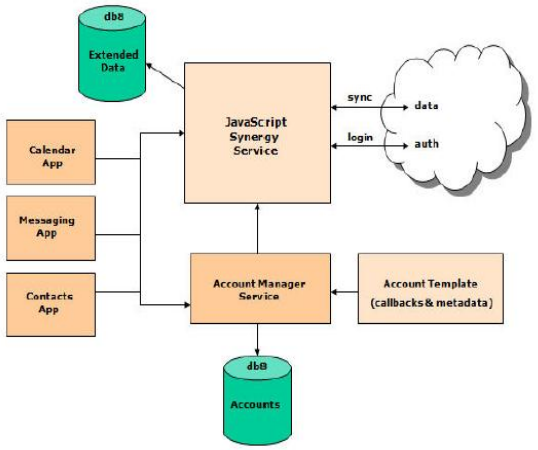

Figure 1.1 : Method for synchronize calendar events, contacts and messages

The service will continue to be able to synchronize changes occurring on the server for as long as it is running. This synchronization from a remote server to the client is usually done in intervals. The intervals are decided by the underlying API that, based on the current processor load and network usage, executes a function registered by the service in order to handle the synchronization procedure.

As of webOS 2.0, HP/Palm is also opening up their Synergy service to allow third-party developers to create connectors for Contacts, Calendars, and Messages. This means that the user will be able to synchronize related information with any external site for which there exists a Synergy connector. [15]

A connector is packaged and distributed as any other application, through the web OS App Catalog. The Connector is built as a JavaScript service that creates an account with the Account Manager and stores the data objects.

In order to be able to package it as a regular application it also has to include an ordinary application; even if the application, in this case, could consist only of an empty file.

The service inside the connector will register the event handlers, also called assistants, which will be responsible for acting when the user commits changes, updates, removes, or adds new information

to a contact. It will also have to implement a trigger callback, which can be called whenever a synchronization event has been triggered, either a manual call by the user or a regular synchronization update. [12]

ANDROID

Android is quite similar to webOS in that both accounts and synchronization services can be created. These services can then synchronize contacts or other information.

A service, which is called a SyncAdapter in Android, performs the synchronization in a background thread while an AccountAuthenticator service handles the account part. [13] [14]

IOS

Synchronization in iOS is built into the operating system and can be used with M4E, MobileMe, Gmail, Yahoo, AOL, and general POP/IMAP, LDAP, CalDAV. There are no special methods designed for third party developer to create new synchronization services, except using the standard APIs for handling and accessing data.

There is only one address book application available, and other applications have to go through it in order to access contact information. Each contact record contains the source from where the information originates. This in- formation is mainly used for displaying the origin for each contact to the user. [15]

ANALYSIS

The web OS platform provides a stable and secure way of handling user sensitive information, such as username and password, for accounts used for synchronizing. Having global secure accounts does also enable multiple services or applications to share the account. The implementation steps necessary to keep this information hidden are however quite advanced, and additional support on the remote server might be necessary for it to be completely secure. There is also no way for the user to know whether the account information is secure or not, since it all depends on the account implementation for each service. Creating a secure way of handling authentication information, that is both easy to implement and easy to use, is no easy task, and not something suitable for this thesis. Further research is needed for a solution to be developed. [12]

Having a service register a callback method that is called whenever the device is ready to synchronize data is an interesting feature, but has to be extended with more user control. It would be suitable for a synchronization API to provide additional callback methods, for example to have services register methods that would be called whenever a calendar event has been added to the local calendar, so that it would be directly pushed out to the remote server.

RESULTS

The proposed synchronization API consists of a few registration functions for registration methods that should be called either when the system is ready for a full scale synchronization, or when an object visible for the service has been changed. The synchronization function would only have to be a trigger function, but the others would have to contain information about the changed data.

An important design question that has to be answered is where a service's functions should be registered. This could be done either each time the service is launched, having each service relaunched on system startup, or registered internally when a service is installed. In webOS, the function registration occurs at installation, and since all services is required to be installed on the device, this is viable even in this solution. This approach would also not require each service to be initialized each time the device is turned on, and instead only initialize and call the functions when they are needed

MESSAGING API

A study done by the Nielsen Company has shown that U.S. teens send an average of over 3 000 texts per month; many teens also say that texting was one of the main reason for them to purchase a mobile phone in the firstplace. [16]

There are several other ways of sending messages, besides SMS, on modern smartphones including MMS, XMPP and other types of instant messaging protocols, most of which originate from desktop computers. These services are similar to each other, and contain the same basic functionality of sending short instant text messages. Some of the services have additional functionality, such as attachments or support for multiple recipients.

This report will not try to address e-mail under this section since it is not designed for instant communication. One common method for allowing third party applications to send instant messages is by letting applications launch the official messaging application; possibly with predefined values, such as message body and recipient. It is also common to support the SMS URI scheme, which allows applications to use links, such as sms:+15105550101?body=hello%20there, to launch the standard messaging application.

WEB OS

The webOS API only provides capabilities to pre-populate fields in the official messaging application. Unlike iOS and Windows Phone 7, webOS uses the

same API to send all supported types of messages, including SMS, MMS and IM messages. [18]

Messaging is handled with the Synergy service, which collects messages from different sources and presents them through a single interface. Developers can develop new Synergy Connectors to communicate with new messaging services, which then allow programs to access the messages with the same API. [15]

WAC

The WAC platform provides a number of ways of handling messages. It supports sending SMS, MMS and e-mail as well as subscribing to incoming messages from these services. The services can however not be extended, as in webOS. WAC also provides extensive methods for searching, filter and listing messages based on different parameters.

The API is using the same functions to send all types of messages but some services only support a subset of the attributes. Subscriptions of incoming messages however use separate functions for each message type.

ANDROID

There are two ways of sending an SMS message using the Android API; either by creating an Intent and open the standard SMS application with some pre-defined values, or by calling the SmsManger API. The API has support for sending text based messages, either as a single message or, if the message is too long, in multiple parts, as well as pure data based messages to a specified application port. [6] [16]

In addition to SMS there also exists a full SIP stack in Android. This makes it easier for developers to create applications that handle VoIP calls, message services or other things that makes use of SIP. This is however not comparable to the simple APIs used for sending SMS. [6].

MAEMO

The Maemo platform, being based on a standard Linux distribution, provides message handling through the Telepathy framework. Telepathy is a framework for managing voice, message, and video communication. It also supports file transfers, managing contacts, and online status (presence). Just as webOS Synergy this allows developers to add support for additional messaging services without requiring application developers to explicitly add support for each service. [13]

Telepathy is built on top of D-Bus (Desktop Bus), which is an inter-process communication framework, and all components run as separate processes [14] [13].

On top there is the Mission Control, which provides the Account Manager and the Channel Dispatcher. The Account Manager handles all accounts the user has set up and can initiate connections. The Channel Dispatcher is responsible for dispatching applications upon either remote requests from the different protocols or local requests from other programs, for example to start a chat with someone. [15]

Each Telepathy Connection Manager handles Connections for one or more protocols. A Connection is the connection to the protocol and it contains contacts, avatars and other things. It can also be used to create new channels for text messaging, calling, file transfers, and so on. Telepathy supports multiple clients, and lets them use the same Connections and Channels. [13]

THE W3C MESSAGING API

The Messaging API from W3C defines methods for creating and sending messages of different types, including SMS, MMS and e-mail. The API is meant to complement the previously defined URI schemes. The API does not handle receiving of messages.

ANALYSIS

Since support for the same feature set that is found in other mobile operating systems is required, the Messaging API from W3C and the use of URI schemes are too limited to be useful. Most of these required features are listed in Figure 3.2. With webOS and Telepathy, developers can extend the messaging platform to support new protocols and services and still have them integrated into the standard APIs. This is an elegant solution and the separation will enable developers to develop services that handle network communications, while other third party developers implement messaging applications that utilizes these services.

Protocols that want to relay their own messages into the system need to implement this in a service. This service needs to be able to create native messages from the protocol's messages, as well as converting native messages to the representation used by the protocol. Services should be careful not to use more of the phones resources (for example battery, CPU, and network usage) then needed. XEP-0286: XMPP on Mobile Devices [16] discusses the XMPP protocol from a battery usage perspective, noting for example 3G radio levels and compression.

RESULT

The resulting API is divided into two layers, as can be seen in Figure 3.2. The lower layer (service level) is meant to be used by services to add support for a messaging protocol. The upper layer (application level) is to be used by regular applications that create a user interface for sending messages through the underlying services.

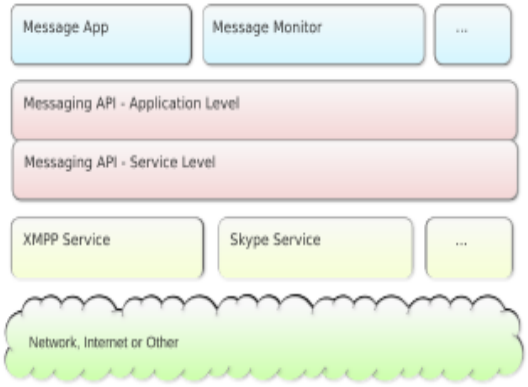

Figure 3.2 : Message model implementation

CALLS API

Making and receiving phone calls is one of the central features in mobile phones, and something that all Web based smartphones have to be able to do. Calling includes incoming and outgoing calls with audio, and video, streams, together with other data and events. Besides traditional telephone calls, calling also includes other types of calls, such as Skype, SIP and other instant messaging services.

The tel URI describes telephone numbers as a string of decimal digits, which uniquely indicates the network termination point. There are also other URI schemes for voice communication protocols, including XMPP and SIP. [12] [8] [13]

The WebOS, iOS and Windows Phone 7 platforms give developers almost the same functionality with regard to calls. On all three platforms there is no direct way for developers to create applications that place calls. Instead there is functionality for opening the default call application with information already typed in. The user must then confirm, often by pressing the call button, for the actual call to be initiated. On these platforms there is also no way to react to incoming or outgoing calls, or retrieve any information about the cellular network, or even get any information about earlier calls. WebOS and iOS handle calls by creating a tel URI and launching it with a special method. In webOS this is done using the Application Manager, either by using the open method with the tel URI or using the launch method with the dialer application's id. [14]

In iOS, developer can use the tel URI as a parameter to the openURL function in the UIApplication class, which will launch the phone application with the given phone number already typed in. [15][16]

In Windows Phone 7, third party applications can, through the PhoneCall- Task class, set the display name and the phone number shown in the standard phone application, but not handle a call directly. [17]

SYSTEM INFORMATION

This section will consider information about the device, operating system and vital hardware parts and how such information should be accessed. Examples of such information could be for example current memory usage, the name and version of the operating system and what capabilities the device has.

Much of this information can already be obtained by using standard methods, either by standardized scripting APIs, as with current time and date, or with other Internet standards, for example using the client's User Agent to transmit operating system name and version. The W3C Device APIs and Policy (DAP) Working Group have been working on an API to gather Systems information and events, as seen in Section 7. This chapter investigates whether the W3C standard is sufficient enough for a web based operating system or if another API will have to add additional functionality. Note that this section does not handle local file storage or anything else that could be abstracted out.

Most operating systems for web OS regard memory information as something that is supposed to be handled exclusively by the operating system itself, and does not like to release that information to any other application.

That is why Android and iOS for example, discourage developer from detecting lack of memory, or calculating the amount of available memory on the device. This is however not something that could be assumed for any operating system, and that is why this kind of abstraction cannot be done in this thesis.

IOS

The iOS API can be used to retrieve information about the local device by using the UIDevice class. According to the API it can be seen as a singleton representation of the device at hand, where applications can get hold of vital information about the device itself. [8]

The UI Device class has divided the properties into five different groups.

Available Features – currently only hold a property to tell whether multitasking is supported on the device or not. The reason for not including additional

features is because of the fact that the developer already knows what brand of device he is working on, and can make assumption based on that fact. Such assumptions will not be possible for a more general API that has to support all possible kind of devices.

Device and Operating System – contains properties for identifying the device, such as a unique id, the model name, and what kind of operating system it is running. Based on this information the application is able to accommodate its services for the current device.

Device Orientation – enables applications to find out the device's physical orientation, in order to tilt the display to accommodate the user interface with the user's view. This category also contains notifications of device orientation, so that applications can be automatically notified when the screen needs to be tilted.

Device Battery State – handles information about the current state and level of the device's battery. The state tells the application whether the device is currently charging, unplugged or fully charged, and the level shows the current charge on the battery in percentage.

Proximity Sensor – indicates whether or not the proximity sensor senses a close object.

This is what Apple has added through their API, but much of the needed information is naturally retrieved by the choice of using objective-C as base.

Monitoring the system and the network connectivity is something that is already built into objective-C and that does not require an additional API from Apple. [8]

ANDROID

The Android API utilizes the Java programming language and, in the same way as iOS, it can get much system information directly from the language itself. [9]

As stated in the beginning of this section, the Android API discourages applications from handling memory information directly, but instead relies on the operating system itself. The application can however retrieve the information, using the ActivityManager, as well as tell the system that the memory level is too low and that the system should consider itself as being in a low memory situation. Note also that, for debugging purposes, a more detailed view could be retrieved directly from the kernel. [9]

WINDOWS PHONE 7

Since Windows Phone 7 has a memory cap of 90 MB on any application that runs on a device with less than 256 MB of total memory, it is necessary to let applications keep track of memory usage. The DeviceExtendedProperties class keeps properties for both the amount that the application is presently using, the maximum amount of memory the application has used during its lifetime, and the total amount of memory on the device. This class can also be used to get device specific properties such as a unique id, the manufacturer and name. [3]

WAC

WAC specifies a Device API by having access methods in the Device Status module fetch information based on pre-specified attributes, listed in the Device Status Vocabulary. The vocabulary groups attributes together in the following groups:

Battery – contains attributes for current battery level, and whether the battery is being charged or not.

CellularHardware – tells whether cellular hardware is available or not.

CellularNetwork – contains current signal strength, roaming capability, and type of operator of the cellular network in use.

Device – contains information about the device, such as model number, version and vendor.

Display – equivalent to the non-standardized Screen Object and contains screen information for the device.

Memory Unit – tells the total memory size, as well as the amount of free built in and/or removable memory on the device.

Operating System – contains information about the operating system: language, version, vendor, etc.

Web Runtime – represent the current web runtime, with information about the available WAC version.

WiFi Hardware – tells whether the device can be used to connect to Wi-Fi networks.

WiFi Network – contains signal strength, network status and SSID of the current Wi-Fi connection.

PHONE GAP

The Phone Gap framework actually includes an API for getting system information, even if the API is very limited. The framework documentation mention properties for getting the device's name, unique id,

platform, operating system version, and version of the Phone Gap API available.

The idea behind the Phone Gap information API is that the developer will only support known devices that are known to have the features required to run a specific application. This makes developing for a pre-specified platform, for example IOS, quite easy, since all IOS devices have very similar hardware features. Problems will however arise when developing a multi-platform application, where availability of some hardware component might not be certain and the device has to manually check for it. That is not possible with the Phone Gap solution. [11]

W3C AND THE SYSTEM INFORMATION API

The System Information API divides all properties and information into a number of different areas, all with restricted access level that each application has to receive permission from the user in order to access, based on the characteristics of the properties. The standard is still only a draft, but in the latest public version, dated February 2nd 2010, the different areas are described .The early work of the System Information API did also include battery status properties, but these have been moved to the Battery Status Event Specification. The specifications do not only include methods for fetching battery information, but also introduce a way of setting an event listener that will receive continues updates of battery level and charging status

ANALYSIS

The handling of system information on current online devices are all limited or simplified by assumptions made by the targeted platform. In the case of Windows Phone 7, there is no way for applications to ask the device if a GPS chip is available or not. The developer can however assume that a GPS is always present on a Windows Phone 7 device, since that is one of the requirements made by Microsoft when selling a license. Some simplifications can be drawn when developing applications for Android and iOS as well. These kinds of solutions are however only viable for platform dependent development. When developing the same applications for multiple platforms, it quickly becomes infeasible to handle pre-defined rules about each platform. [13]

The only solutions studied in this paper that do not rely on presumptions made according to the operating system are WAC and the W3C proposed standard. When comparing the two, they both have quite similar properties and work in the same way. Note that the W3C proposal does not state how to access the device, but only provides information about the device and its parts.

RESULT

The WAC API and the W3C proposed standard complement each other well. An operating system providing support for both of these would cover all necessary properties for application development in modern smartphones. It would provide a fully usable interface for developing web applications that covers everything from system monitoring to accessing information about input and output devices.

MULTITASKING

According to Maximiliano Firtman's definition [16] a device needs, among other things, a multitasking operating system in order to be categorized as a online or web OS. Other definitions might have a different opinion, but it is clear that multitasking is one of the most important features in modern and all operating systems studied in this paper have, if not full then at least some, multitasking capabilities.

There are different kinds of multitasking, and almost all vendors have come up with different solutions to balance the battery life, functionality and userfriendliness in their operating system. Applications can be minimized, and the user can switch between them, or a process might be running completely in the background where they might always be able to get some processor power. Some operating systems have an automated system for shutting down applications when processing power and memory is starting to get low. [14]

IOS

As of IOS version 4.0, having an application running in the background is no longer limited to first party developers. The developers do however need to work through specially designed APIs to achieve this functionality, and even then the background applications have restrictions that limit their access to the phone's resources.

The applications that need support for accessing resources while running in the background have to register to some of the available services to handle the multi-threading for them. [15] If all an application needs is to finish its current task, like downloading a file, the application can register the task with the Task Completion service and the task could finish even if the application is put to the background. When the task has completed, the application will receive a notification to display to the user.

If all an application needs is to finish its current task, e.g. downloading a file, the application can register the task with the Task Completion service and the task could finish even if the application is put to the background.

When the task has completed, the application will receive a notification to display to the user. Another service, called Background audio, allows

applications to continue playing audio when put to the background. This service is commonly used to implement an Internet radio or a music player that needs to be able to play audio even when the user has switched application.

For applications that need to handle incoming network messages, to implement a messaging service, like Skype or GTalk, they may use the service called Messaging service. Here, developers can register a message handler that will receive new incoming messages of a certain type and notify the user accordingly. It is also possible to enable a timed event to occur even in a background application by calling the service Push Notification. The service will alert the application when the timer has run out and execute the timed routine.

The last background service available for applications is the Background Location Service, which enables applications to track the user's movement by enabling the GPS even for programs running in the background. [17]

By having such restricted background access, the operating system does not need to implement a full featured thread scheduler and can save both computation power and battery life, but still give the user the feeling of a multitasking operating system.

As of iOS 4 the operating system also includes an application switcher where users can bring up a list of hibernated applications and let the user browse between them. There is however no native API for letting developers create their own application switcher, manage installed applications or list the services running on the device. [19]

ANDROID

Android has a different approach when it comes to multitasking and switching between applications. The idea is that the user does not have to know whether an application has turned off or is still left in the memory; the operating system will handle that automatically. That is also why the Android operating system does not include an application switcher that lets the user switch between open applications.

Multitasking in Android basically works by having the operating system not turning off applications when the user switches to a new application. Instead, it leaves the hibernated application in the memory, until the memory is needed by another application. By letting the applications stay in memory, the applications will be much faster to start up again. Android gives developers the ability to register its applications as services, giving the application access to actually execute code while running in the background.

Since Android 1.5, services that run in the background are limited to using 5-10 percent of the total CPU. This increases both the availability of the operating system but also the battery life of the device. [14]

To free up memory, as more and more applications are left in memory, Android implements an Out of Memory (OOM) killer. The routine works by having two memory thresholds. When the first threshold is met, the background processes are notified and asked to save their state in the persistent storage. When the applications have saved their state, they return back to the OOM routine which, when the second threshold has been reached, starts to turn off the non-critical applications whose state has been reported as saved. Since all applications save their state, they will be able to return to the same state as before when launched the next time. This means that the only thing differentiating between an application that has been turned of and an application that is still running is the time it takes to launch the application. [11]

API

The Android API for interacting with running applications and services, the ActivityManager API, separates applications and services by defining separate namespaces and separate methods for each of them. The API provides methods for fetching a list of running applications or services or, if the application has got permission from the user, shut down a specified application by terminating its processes. It is also possible to get a separate list for applications that are in an error state, or a list of the most recently started applications that include applications that are no longer running.

The API can be used not only for listing running applications, but also to fetch more specific information about each running process; for example the process' PID, its memory usage and how long it has been running for. An application that is in an error state contains a little bit more information, such as the error message and a stack trace.

WINDOWS PHONE 7

Multitasking and having applications working in the background is, in Windows Phone 7, limited to selected third party and first party applications only. Regular applications are limited to receiving and sending notifications which means that there is, for example, no way for those applications to continue playing music when running in the background. The application can use the received notifications in order to; for example, change the icon tile to display the information dynamically. Microsoft had said that they have plans of extending the multitasking capabilities for third party applications, as well as introducing ways of switching between open applications, but has

yet to define how. That is also why their API cannot be studied in more detail in this thesis.

When a new application is launched, the old application will get a notification asking it to save its current state in the device's memory and hibernate. The procedure is similar to the solutions in iOS and Android where applications can be turned off by the operating system to free up used memory blocks.

Saving the state of each application will enable applications to resume in the same state as when the operating system turned them off. Windows Phone 7 does not include any native APIs for application management; this is only supposed to be handled by the operating system or first party applications.

WEB OS

The webOS approach to multitasking is also a bit different. Here the user has full control over which applications that are running and may turn them off at any time. The user interface consists of cards, which is basically a rectangular screenshot of the application that the user may flick through.

To turn off an application, the user simply grabs the application's card and throws it off the screen. [17]

Open web applications that are running in the background are limited to only accessing operations that are considered to use a moderate amount of memory and CPU to keep the battery from draining too fast. Constant data requests, or accelerometer access, are examples of prohibited usage.

Applications using the PDK are also a bit more restricted when running in the background and may no longer allocate more memory or use the graphic APIs.

Applications that are turned off by the user are typically completely shut down by the operating system, but the user also has the possibility of letting the application stay alive even after it has been terminated. This is done by putting the application icon in the webOS Dashboard, making it a Dashboard Application that, if implemented to support it, may run as a service even after it has been turned off. [17]

If the user has too many application cards open, and the operating system runs out of memory, the user will be prompted with a warning and forced to turn off applications in order to be able to continue using the device. This leaves the user in full control over which applications that should continue to run, and which should be shut down. However, there is no indication of how much memory each application is allocating and the user might close low memory applications unnecessarily, before finding an application that has allocated a larger amount of memory. [14]

MAEMO

Multitasking in the Maemo operating system is more similar to personal computer's operating systems and is one of the few, or possibly the only, operating system for smartphones to implement real memory swapping as well as a desktop like interface for application switching. It also includes full multi-task capabilities for minimized applications, enabling all applications to continue to run even when the user has minimized them or opened some other application. The applications receive notifications, stageActive and stageDeactive, when minimized or activated so they can optimize the CPU and memory usage accordingly.

ANALYSIS

As seen in the previous sections, there are a wide variety of solutions to multitasking on different OS platforms. The problem whether to minimize or close applications could be solved in different ways and is something that should be up to the platform to decide; it could be seen as a question that depends too much on memory capacity of the device and the implementation goals of the vendor for it to be answered in this thesis. Guidelines and general tips on what kinds of simplifications that could be made when exclusively working with web based applications however, could fit the scope, as well as a small proposal on which services should be provided in order for the applications to work in the background and optimize memory usage.

Something common with all solutions, previously mentioned in this chapter, is that an API for getting information about installed and running applications is practical to allow applications to handle listing, switching, launching, closing, installing, and removal of applications. The question in this case however, is whether the operating system should allow such tasks to be carried out by web applications at all, or if this is something that is best handled in the operating system itself. Web pages are normally sandboxed inside the web browser and not aware of any other open applications or tabs. Introducing such a feature would break this fundamental security feature that the Web is built on today [18]

Installing applications, such as extensions and add-ons, is something common in modern desktop browsers and even some mobile browsers, today and the installation schema should be very similar. By comparing the installation procedures of installing applications on smartphones with installing extensions in browsers, it can quickly be concluded that the cases are very similar. This means that installing applications could very well adopt a similar behavior to how extensions are currently handled today.

Removing extensions is another question, since removing an application or extension means exposing installed application data to other applications, which could be a serious problem of integrity. Browsers have dealt with this problem by

not allowing extensions to list or remove other extensions and most OS only allow the built in applications to handle such tasks. [13]

Since web based scripts are running in a sandboxed environment, there is currently no API for handling application switching or list any running applications, with the current web standards, at least not by the definition of an application used in this thesis. It is possible for applications to get information about what is happening, for example if an application has lost focus and been minimized or if it is about to be shut down. But having applications fetching information about other running applications is a question of integrity and in order to find a definite answer, further studies are needed and the question is unfortunately not something that could be answered in this thesis. This means that a safe way of proceeding with this could be possible.

RESULT

Multitasking is required in modern OS, and everybody is doing it to some degree. To maintain full compatibility with current web applications the operating system needs, at least regarding this topic, to function as a standard web browser.

Since compatibility with existing web applications is wanted, all scripts in the application need to continue to run even while the application is not active.

This does not mean that full priority needs to be given to the background applications. A smaller subset of the full performance should be sufficient, just as is done in webOS. [17]

It is not necessary to render anything for the applications that are not visible. If a thumbnail of the application is needed, one can be taken when the application is going out of view. [19]

Web applications that supports being able to be automatically terminated by the operating system, and then resumed in the same state, can be created today by using onLoad and onUnload/onBeforeUnload events. Applications can receive updates for when they are being started or shut down. Since support for pre-existing web applications is wanted, it is hard to implement the solutions that Android, Windows Phone 7, and iOS have implemented.

The solution in webOS however, is reasonable.

Swapping out memory to the hard drive, as has been done in desktop operating systems for many years, is a solution that has proven to work well in Maemo. Moving the memory swapping up the chain, into the web browser, would probably mean that some heuristics could be applied to make it more effective. [11]

REFERENCES

[1] W3C. HTML Living Standard – web storage. http://www.w3.org/TR/ webstorage/ (2011-05-06), 2011. 8

[2] Nikunj Mehta, Jonas Sicking, Eliot Graff, Andrei Popescu, and Jeremy Orlow. Indexed database api. http://www.w3.org/TR/IndexedDB/ (2011-05-18), 2011. 8

[3] Arun Ranganathan and Jonas Sicking. File api. http://www.w3.org/ TR/FileAPI/ (2011-05-18), 2010. 8, 80

[4] Eric Uhrhane. File api. http://www.w3.org/TR/file-writer-api/ (2011-05-18), 2011. 8, 80

[5] Anne van Kesteren. Xmlhttprequest – w3c candidate recommendation 3 august 2010. http://www.w3.org/TR/XMLHttpRequest/ (2011-05-18), 2010. 9

[6] Anne van Kesteren. The websocket api – editor's draft 12 may 2011. http://dev.w3.org/html5/websockets/ (2011-05-18), 2011. 9

[7] Ian Hickson. Html5 web messaging – editor's draft 12 may 2011. http://dev.w3.org/html5/postmsg/ (2011-05-18), 2011. 9

[8] Ian Hickson. Video conferencing and peer-to-peer communication. http://www.whatwg.org/specs/web-apps/current-work/complete/video-conferencing-and-peer-to-peer-communication.html (2011- 05-19), 2011. 9, 78

[9] Stuart Robinson. Multi-core processors to penetrate 45 percent of smartphones by 2015. StrategyAnalytics, Jan 2011. 10

[10] Ian Hickson. Web Workers. Web Workers - W3C Working Draft 10 March 2011, Mar 2011. 10

[11] Google Inc. Google chrome os. http://www.google.com/chromebook (2011-05-12), May 2011. 11

[12] Amazon. Amazon – samsung series 5 3g chromebook. http://www. amazon.com/gp/product/B004Z6NWAU (2011-05-04), 2011. 12, 81

[13] Maximiliano Firtman. Programming the Mobile Web. O'Reilly Media, Inc., first edition, 2010. 12, 64, 66, 68, 74, 75

[14] The PhoneGap project. PhoneGap – supported features. http://www. phonegap.com/features (2011-05-13), 2011. 12

[15] WAC Application Services Ltd. WAC – About WAC. http://www. wacapps.net/web/portal/about (2011-05-23), 2011. 13

[16] WAC Application Services Ltd. WAC – FAQ. http://www.wacapps. net/web/portal/faq (2011-05-23), 2011. 13

[17] WAC Application Services Ltd. WAC – Membership. http://www. wacapps.net/web/portal/membership (2011-05-23), 2011. 13

[18] HP. Application framework and OS. https://wiki.mozilla.org/Labs/ Contacts/ContentAPI (2011-03-20), 2011. 14

[19] IDC. Worldwide smartphone market to grow by nearly 50 percent in 2011. IDC Press Release, Mars 2011. 18

[20] Nathan Olivarez-Giles. Nokia to cut 7,000 jobs, stop developing symbian operating system. Los Aneles Times, April 2011. 18

[21] Google Inc. Using the Contacts API. http://developer.android.com/ resources/articles/contacts.html (2011-05-09), 2010. 20, 21, 81

[22] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627 (Informational), July 2006. 22

[23] HP. Contacts.

https://developer.palm.com/content/api/reference/datatypes/contacts.html (2011-04-15), 2011. 22

[24] HP. People Picker. https://developer.palm.com/content/api/ reference/services/people-picker.html (2011-04-15), 2011. 23

[25] Richard Tibbett. Contacts API. Contacts API - W3C Working Draft 09 December 2010, 2010. 23

[26] Richard Tibbett. Contacts API. Contacts Writer API - W3C Editor's Draft 04 October 2010, 2010. 23

[27] Michael Hanson. Labs/Contacts/ContentAPI.

https://wiki.mozilla. org/Labs/Contacts/ContentAPI (2011-03-20), 2010. 23