Tool Support for Constructing Mobile Mashups

Lasse Holmstedt[1], Tommi Mikkonen[2], and Mikko Terho[3]

[1] Nokia Qt Development Frameworks
Invalidenstrasse 117, Berlin, Germany
lasse.holmstedt@nokia.com
[2] Tampere University of Technology
Korkeakoulunkatu 1, Tampere, Finland
tommi.mikkonen@tut.fi
[3] Nokia Devices, Visiokatu 5, Tampere, Finland
mikko.j.terho@nokia.com

Abstract. The ability to instantly publish software worldwide, and the ability to dynamically combine data, code and other content from numerous web sites all over the world has opened up entirely new possibilities for software development. In web terminology, a web site that combines ("mashes up") content from more than one source into an integrated experience is referred to as a *mashup*. At present, the development of mashups usually relies on the tools for composing server-side software, and off-the-shelf browser is commonly assumed as the runtime environment. However, when considering client-side mashups that are well-suited for mobile devices due to local processing and associated interactivity, numerous complications exist. One of these problems is available tool support, which is commonly targeted to desktops and browsers. In this paper, we introduce a tool for developing client-side mashup applications. In spirit, the tool is similar to tools available for mainstream mashup development, but all the actual processing is done on the client side using a special purpose runtime environment.

Keywords: Web applications, mashup development.

1 Introduction

In the past few years, the Web has become a popular deployment environment for new software systems and applications such as word processors, spreadsheets, calendars and games. In the new era of web-based software, applications live on the Web as services. They consist of data, code and other resources that can be located anywhere in the world.

The ability to instantly publish software worldwide, and the ability to dynamically combine data, code and other content from numerous web sites all over the world will open up entirely new possibilities for software development. In web terminology, a web site that combines ("mashes up") content from more than one source is commonly referred to as a *mashup*. Mashups are content aggregates that leverage the power of the Web to support instant, worldwide sharing of content.

Today, mashups are run inside a web browser. However, because the web browser was originally designed to be a document viewing tool – not an environment for highly interactive applications – there are challenges when running web applications and mashups that behave in a highly interactive fashion, especially in mobile devices. Support for user interface widgets can also be limited. Furthermore, poor performance of the web browser can be a major issue especially when running mashups in mobile devices. On the other hand, success stories of application stores by Apple and others have shown that users wish to use content and applications that have tight integration with the mobile platform. Consequently, it seems more fruitful to run mobile mashups predominantly on the client side, as already proposed in [1, 2], to combine the best possible performance and benefits of the services residing in the web.

In this paper, we introduce a tool created for easing the development of mobile mashups that are predominantly run on the client-side. The tool has been constructed using Qt, an industry-scale cross-platform environment, which has been rapidly extending into mobile devices. In terms of background, the work is based on our previous research results [1, 2], and on extending tools that the Qt environment provides for composing compelling applications.

The paper is structured as follows. Section 2 discusses mashup development in general. Section 3 introduces Qt and associated tools that were used in our implementation. Section 4 provides the tools for composing client-side mashups, and Section 5 introduces some sample mashups composed using these tools. Section 6 provides a discussion on our experiences, and Section 7 finally concludes the paper with some final remarks.

2 Mashup Development

Mashups – systems that amalgamate existing content from the web – can be composed manually using the classic DHTML technologies. However, since the actual representation of data, behavior and content can vary dramatically between different web sites, manual mashup construction can be extremely tedious, fragile and error-prone.

A number of tools are available for mashup development. To begin with, mashups can be developed using general-purpose web application development platforms. Unfortunately the capabilities of such general-purpose web programming environments are still somewhat limited in features, especially when considering flexible extraction and combination of data from different web sites, which is important for mashup development. Most general-purpose web content development tools bear the same (or similar) shortcomings. Finally, there are also tools that have been intended for mashup development, many of which are more or less experiments. In such tools, there are some common emerging themes and trends [2]:

Using the web browser not only to run applications/mashups but also to develop them. For instance, and Yahoo Pipes (http://pipes.yahoo.com/pipes/) and Google Mashup Editor (http://code.google.com/gme/) use the web browser to host the development environment and to provide seamless transition between the development and use of the mashups.

Using visual programming techniques to facilitate end-user development. Visual “tile scripting” and “program by wire” environments are provided by Yahoo Pipes, for example.

Using the web server to host and share the created mashups. Many mashup development tools store the created mashups on a web server that is hosted by the service provider.

Direct hook-ups to various existing web services. Since the Web itself does not provide enough semantic information or well-defined interfaces to access information in web sites in a generalized fashion, most of the mashup development tools include custom-built hook-ups to existing web services such as Twitter (http://twitter.com/), Digg (http://digg.com/), Facebook (http://www.facebook.com/), Flickr (http://www.flickr.com/), Yahoo Traffic (http://developer.yahoo.com/traffic/), Google Maps (http://maps.google.com), and various RSS news feeds.

Despite the ever-increasing role of the web browser as target platform for mashups, the availability of a web browser is not an essential requirement for mashup development. On the contrary, there are technologies that utilize custom-built, special-purpose web runtimes that can bypass security limitations associated with the browser and can offer better performance, e.g., by performing the client-side mashup generation using native processing capabilities. For instance, mashups intended for mobile devices often utilize a custom-built client environment. Similarly, tools used for composing mashups need not be run inside the browser, but they can also be implemented as extensions of generic software development environments.

Our approach to the development of client-side mashups is based on a runtime environment for client-side mashups. The fashion we have implemented the system builds on top of Qt and our earlier activity, Lively for Qt (http://lively.cs.tut.fi/qt/) [1], as well as experiences on mobile mashups discussed in [2].

3 Qt as a Mashup Environment

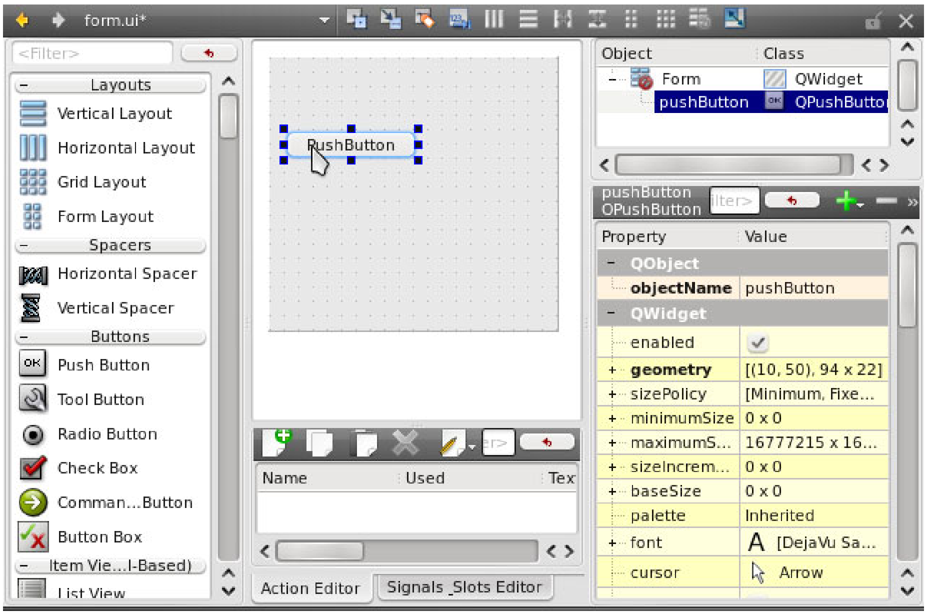
Qt (http://www.qtsoftware.com/) is an industry-scale cross-platform environment that supports a rich set of APIs, widgets and tools that run on most commercial software platforms. In addition, Qt has been used in various embedded devices and applications, including in particular mobile phones, but also PDAs, GPS receivers and handheld media players.

From the technical viewpoint, Qt is primarily a GUI framework. It comprises a rich set of widgets, graphics rendering APIs, layout and stylesheet mechanisms. In addition, tools are provided that can be used for creating compelling user interfaces that run in a wide array of target platforms.

Qt has been gradually expanded to offer a range of connectivity facilities as well. Networking, filesystem access and web browsing capabilities are all available, together with scripting, multimedia and XML processing frameworks. While Qt is primarily targeted for C++ development, bindings to its libraries are also available for other languages, officially for Java and unofficially for several others, perhaps most notably Python.

3.1 Qt Creator

Intended to make Qt development easier, Qt Creator is a multi-platform IDE for several languages. While it primarily supports C++, other languages such as ECMAScript are also supported. The architecture is entirely plug-in based, including the components shipped with the editor itself. Some of the most important features are integrated help for the Qt Framework, a code editor and Designer, a tool for visual GUI design. Designer is intended for building traditional desktop interfaces and provides limited support for creating complex, non-standard interfaces. It is best used for its original purpose and requires a fair amount of understanding how interfaces created with Qt work, as its widget and tool names are derived from their Qt class counterparts. Qt also has a concept of styles akin to CSS, but only a text editor is provided to the user in this respect. However, the most important part of the tool, laying out widgets, is made easily approachable with a standard drag-and-drop interface, shown in Figure 1.



Property	Value
QObject	
objectName	pushButton
QWidget	
enabled	✓
geometry	[(10, 50), 94 x 22]
sizePolicy	[Minimum, Fixe...
minimumSize	0 x 0
maximumS...	16777215 x 16...
sizeIncrem...	0 x 0
baseSize	0 x 0
palette	Inherited
font	A [DejaVu Sa...
cursor	Arrow

Fig. 1. Qt Designer default interface

From the technical point of view, Qt Creator is very extensible with its plug-in interface, but the interfaces are not documented well. In effect, a plug-in author has to study the interfaces through the numerous existing plug-ins and examples provided by the framework. The Designer plug-in, on the other hand, has only a handful of plug-in interfaces. While there is decent documentation for all of them, there is not enough built-in extensibility. Due to such technical restraints, implementation of extra functionality had to be made by directly modifying the Qt Designer code, in effect branching it. Additionally, a separate Qt Creator plug-in was created to facilitate creation of mashups through a combination of visual programming and traditional scripting. Combined, these plug-ins enable loading, modification and display of data with minimal effort.

3.2 Qt for Client-Side Mashups

There is a strong connection between Qt and web applications. Qt libraries include a complete web browser based on the *WebKit* (http://webkit.org/) browser engine. Moreover, the necessary DOM and XML APIs are included to parse, manipulate and generate new web content easily. In addition, Qt includes a fully functional ECMAScript [3] (JavaScript) engine called *QtScript*. The presence of a JavaScript engine is important, since JavaScript – along with XML – is the *lingua franca* of the Web that is used by popular web service APIs such as the Google Maps API (http://code.google.com/apis/maps). Powerful debugging tools are also available for QtScript, making it more attractive for developers. However, these debugging tools have not yet been fully integrated to the Qt Creator IDE, making their usage more difficult than the ones intended for C++ debugging.

Qt also offers excellent connectivity facilities to network resources, as well as an SQL database interface and filesystem access classes. In particular, network access is made as simple as making requests, without having to care for the protocol, provided that it is supported. HTTP, FTP, TCP and UDP are supported out of the box, as well as SSL-secured connections.

Based on the above, Qt lends itself to a client-side mashup environment. The mashups can leverage the rich Qt APIs for information visualization and processing. In addition to binary image and video formats such as GIF, JPEG, PNG and MPEG-4, textual representations such as XML, CSV (Comma-Separated Value format), JSON (JavaScript Object Notation) and plain JavaScript source code play a central role in enabling the reuse of web content and scripts in new contexts. Qt provides excellent capabilities for processing such information.

In terms of pure functionality, Qt has a definite upper hand over pure web-based mashup development. Its WebKit component enables the usage of any web-based mashups, and allows for their further manipulation through the WebKit's DOM interface and JavaScript engine. Existing mashups can also be reinforced with local filesystem content and results can be locally cached and saved, allowing for faster access on subsequent runs. Entirely new possibilities also realize – building mashups using data sources readily available on the operating system, such as address book data, GPS location and user-created content such as images, video, documents and other such content, becomes possible. Examples of mashups utilizing this information could be fetching user-relevant content from the web, like music, while the user's personal preferences could be analyzed locally from their music library. Other examples include using user's location data as an input for a web service offering travel information of the surroundings, such as famous sights.

Qt's main advantages are its high performance and clear API, which encapsulates most of the difficult and tedious tasks involved with network connectivity or filesystem access. Additionally, the cross-platform nature of Qt enables deployment to a wide range of platforms. However, to make development faster, a higher concentration on QtScript is needed instead of vanilla C++, while still maintaining the native performance of UI rendering and data access through script bindings.

4 Client-Side Mashup Tools

The mashup creation process for Qt Creator could be described as follows – take data, manipulate it and assign it to a display widget. Because Qt Creator already offers a powerful GUI design tool, Qt Designer, mashup integration with it was decided upon. A data model editor, similar to that of Yahoo! Pipes with a visual programming interface, was also created. In the following, this visual editor is called Mashup Editor. Although some extensions had to be built into Qt Designer to enable building mashups, the modifications are not called by any name in particular. Figure 2 illustrates the intended mashup creation process.

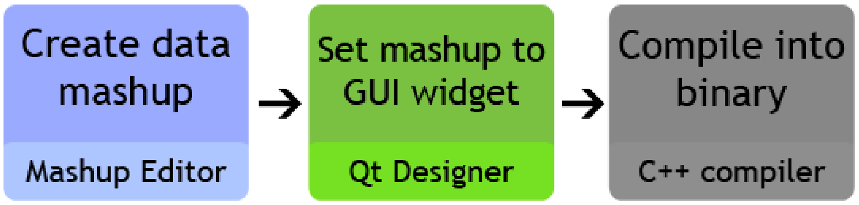
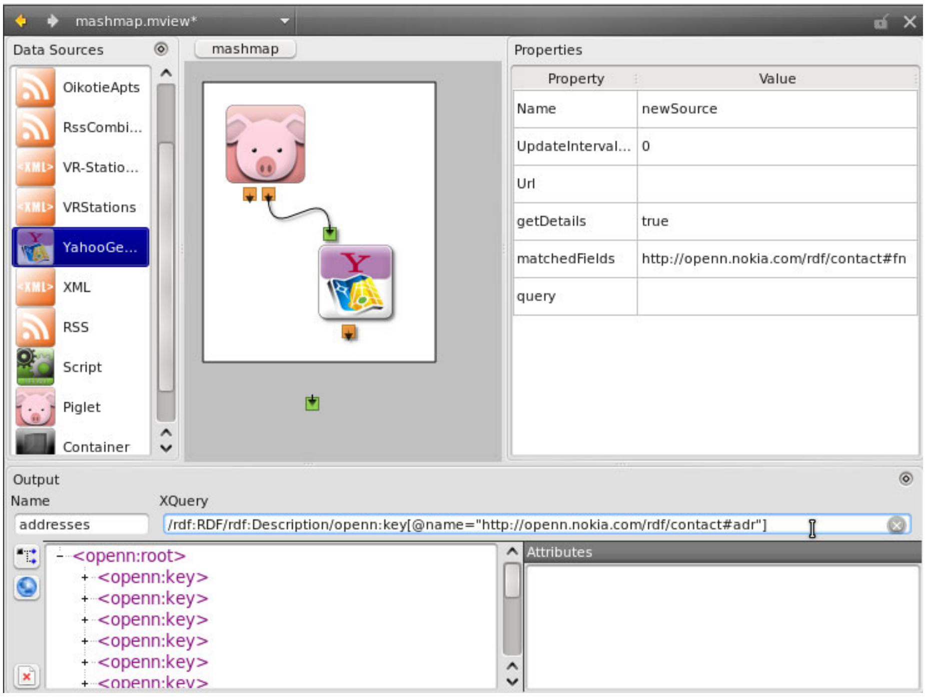


Fig. 2. Creating a mashupping application

The mashup creation part is not always necessary, as previously created mashups are reusable. Also, Qt Creator is able to use the compilers installed into the system and offers and simple button interface for compilation, further easing the process. Therefore, creating simple data-displaying programs can be as simple as dragging and dropping content to widgets and pressing a button to compile the generated code. Illustrations of both the Mashup Editor and the Qt Designer extensions can be found in the following sections.

4.1 Mashup Editor

Mashup Editor is a visual programming tool that enables loading, modification and display of data in a graph-like interface. The editor was created from the ground up as a plug-in for Qt Creator, so that existing functionality of the IDE and interaction between form design and mashup creation could be harnessed. The tool creates two types of XML files. *Mashup files* describe how various data and script elements interact with each other, and *Mashup views*, which describe how the visual mashup elements, denoting either a source of data or a script, are laid out on the screen. This separation was made because the content of the view files is not needed in order to process the mashup data. A screenshot of the editor used to create this markup is shown in Figure 3.



Property	Value
Name	newSource
UpdateInterval...	0
Url	
getDetails	true
matchedFields	http://openn.nokia.com/rdf/contact#fn
query	

Name	XQuery
addresses	/rdf:RDF/rdf:Description/openn:key[@name="http://openn.nokia.com/rdf/contact#adr"]

```
- <openn:root>
    + <openn:key>
    + <openn:key>
    + <openn:key>
    + <openn:key>
    + <openn:key>
    + <openn:key>
```

Fig. 3. Screenshot of the Mashup Editor for the Qt Creator

As the tool was originally intended to be used to aid in software prototyping, the data type was limited to text. XML based documents are preferred, because Qt contains good XML support, and because a lot of the Web content is XML based. Furthermore, converting other types of data to valid XML is usually a rather straightforward process. However, no limitations to the document contents are made, and the data processor of the mashups could just as well process binary files. Regarding the scripting, QtScript – essentially JavaScript with some minor deviations regarding libraries – is used, as it is an integral part of Qt itself. It is enhanced with a collection of libraries created with the experimental QtScriptGenerator, a tool which creates bindings from most of the available Qt objects into QtScript, exposing more Qt functionality to the script engine.

In Figure 4, a simple mashup, which combines the contents of two RSS feeds into one XML listing, is shown. In the row above, two RSS icons have their output nodes *connected* to the input nodes of a script processor. A single output can have multiple outgoing connections, but an input can have only a single incoming connection. Additionally, an output node of an XML element, such as the RSS feed, can be modified with XQuery expressions, which are natively supported by Qt. The script element contents can be either written by the user or a previously saved script can be assigned. When creating a new script, all boilerplate code is automatically generated to assist the user. New inputs and outputs can be created to script elements, and to other appropriate elements as well.

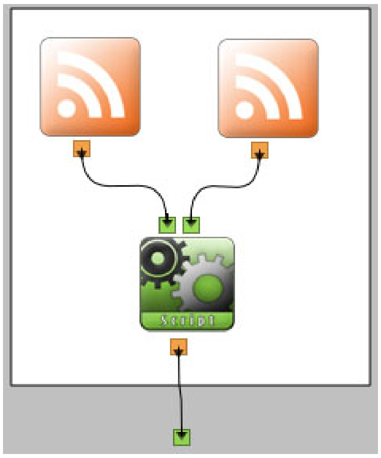


Fig. 4. A mashup system concatenating two RSS feeds

A powerful feature of structuring mashups and building them from pre-made parts is that mashup elements can be placed inside containers, and containers can be also placed inside each other. The containers act like other mashup elements – they have input and output nodes, and allow such connections to be made. Otherwise, they behave like black boxes – only the information returned as output from them is seen by the rest of the elements. This considerably facilitates building complex mashups or reusing existing components.

After mashups are saved, they can be set as templates, after which they are available as an otherwise limited list of building blocks. The mashup elements available by default are XML, RSS, Container, Script and YahooGeoCoder. Additionally, support for a triple-store database called Piglet exists. The capabilities of the most primitive items, XML and RSS, are simple. They are able to load data from an URL understood by QNetworkAccessManager, or local files, specified by their full path. YahooGeoCoder is an example implementation of a REST-based element, which is able to receive place names or addresses as inputs, sends data to Yahoo's Geocoding service [4], and returns GPS coordinates.

A developer can create new native items, and new script templates can be introduced as well by installing them in the appropriate directory.

To facilitate manipulation of XML based data, a drag-and drop-based XQuery editor exists. While not able to handle complicated logic, several common actions such as element and attribute selection are in place. Drag and drop is extensively used in other parts of the Mashup Editor interface, too. Examples of such behavior include dragging and dropping web browser bookmarks and text containing URLs into the editor, which automatically creates XML elements with URLs pointing to the given sources.

With the features listed above, the mashup editor is able to produce XML-compliant files that describe how data is to be processed. Next, we take a look at the extensions built into Qt Designer, which enable attaching these mashup instructions to an UI created with Qt Designer.

4.2 Qt Designer Extension

Extensions built on top of Qt Designer enable the user to set data on a number of item view widgets [5]. Due to the lack of an extension interface built for such a purpose, the whole Qt Designer code was branched and necessary modifications were built on top of that. While this posed some new challenges, such as maintainability and a large overhead due to lack of interface documentation, a relatively easy-to-use drag-and-drop interface could be built.

The extension implements a similar interface for data models as seen earlier in Figure 1. This data model list can be accessed through a toolbar button or a keyboard shortcut, in similar fashion with the existing tools. The mashup files that are enumerated in the data model list come from two different locations: user's home directory and the current Qt project's directory. The user's home directory contains mashup templates, which can also be utilized directly from within Designer, while the current project directory typically contains all project-specific mashups. While the user cannot directly modify mashup templates, an editor for project-specific ones can be opened by double-clicking the desired item.

To assign a mashup into a widget, the user needs to first create an item view widget that inherits the QAbstractItemView class on the form canvas by dragging and dropping it there. Qt ships with a number of such widgets, and the user is free to subclass their own, provided that they follow the Model-View-Delegate pattern used in Qt [6]. Next, the user is able to drag and drop a mashup on top of the view item. Provided that the mashup engine can load the data as instructed in the XML files, the Designer interface will display the XML based per-item output in the view.

As an example of subclassing QAbstractItemView, a Nokia Maps Item View widget was created, which is able to parse location data in several XML based formats and project these points as points of interest (POI) on the map. Such a widget cannot display most text-based data in a sensible manner, but for location-based content, it is very effective.

The Model-View-Delegate pattern also allows for the user to set delegates into views, which is also implemented in drag-and-drop style in the extension. Delegates are simply components that instruct how the data is displayed for per widget, or by its rows and columns. The Designer extension implements the ability to set a per-widget delegate, which essentially means that the whole widget will be rendered with the instructions of a single delegate. More detailed delegate processing can be done on the code level.

When a user double-clicks on an item view widget or presses a button in the toolbar of the editor, the data and list toolbox is replaced with a list of available delegates. The delegates are stored as binary files in the QPicture format provided by Qt [7], which lists 2D painting instructions in a sequence.

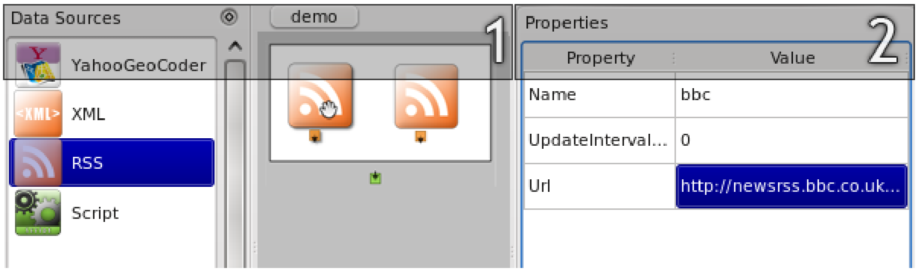
Creating these delegates is difficult, however, since no editor is available for creating QPicture content. While delegates are fairly straightforward to create in C++ with an average lines of code less than 50, it still requires a knowledge that is out of place for a GUI editor. The reasons for not building a delegate editor are addressed later in the paper.

5 Sample Applications

To demonstrate two different types of applications that can be created with the aid of the mashup editor, two samples are provided. One uses a map widget to display data, while the other relies on standard item view widgets and delegates to display latest news items from an RSS feed.

5.1 RSS Newsfeed Combiner Mashup

The first sample application is a simple list of news items from two RSS services – BBC World and AP News. While RSS is a very popular and fairly simple XML based syndication format, it offers a comparison with already existing tools that allow users to take existing content, modify it, and display it on a widget. Because wizards to create new projects are built into the mashup editor, we concentrate on how the mashup is built with the editor. To add both of the BBC World and AP News feeds to the mashup canvas, the user drags and drops two RSS elements to the canvas (Figure 5, left). To assign URLs to them, the user can copy the URL into each element's respective panel (Figure 5, right).



Property	Value
Name	bbc
UpdateInterval...	0
Url	http://newsrss.bbc.co.uk...

Fig. 5. Adding RSS feeds to the canvas and modifying their properties

Next, the user needs to combine the data of the two feeds by concatenating them. For this duty, the user can add a new script element (Figure 6) and select which template to use by double-clicking on it (Figure 7). A template for concatenating XML content exists, and is logically called Concatenator. The user can also write their own script and freely modify the script after selecting the template, as a copy is made out of them.

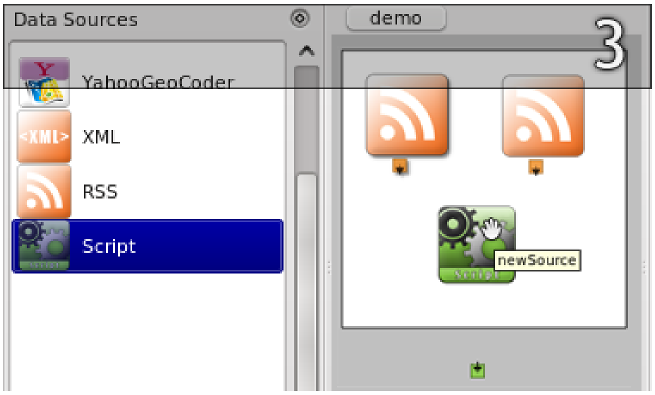
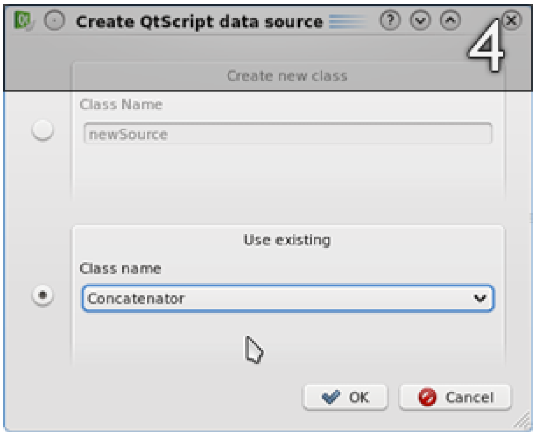


Fig. 6. Adding a script processor to the canvas



Create new class

Class Name

newSource

Use existing

Class name

Concatenator

Fig. 7. Creating a new script out of a template

Next, two input and a single output are created for the script element to provide it with data. The output is connected with the output of the container mashup itself (Figure 8, left side) and the inputs are similarly connected with the outputs of the RSS items (Figure 8, right side). The connection happens in a drag-and-drop fashion, starting from either end of the node pair and finishing at the other end.

Finally, the user simply drags and drops the data model on top of the list view, double clicks it and assigns a delegate which is able to render the RSS data in a sensible manner. As creating delegates themselves is somewhat difficult, the editor currently ships with a number of delegate templates, including renderers for the popular RSS and Atom feed types. This process is shown in Figure 9.

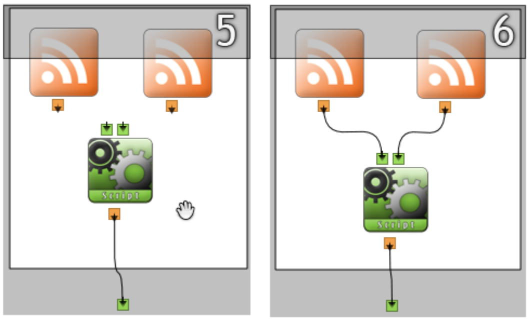
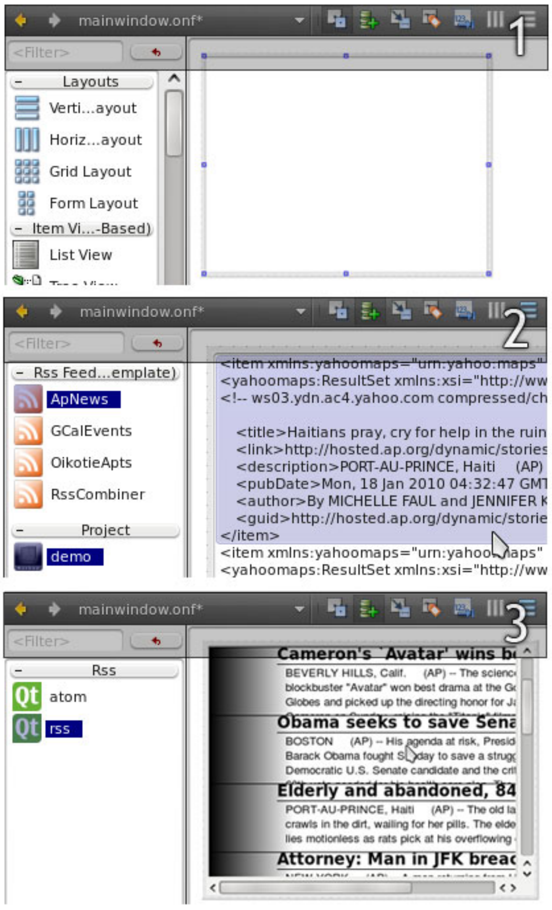


Fig. 8. Connecting mashup subsystems together



```
<item xmlns:yahoomaps="urn:yahoo:maps"
<yahoomaps:ResultSet xmlns:xsi="http://ww
<!-- ws03.ydn.ac4.yahoo.com compressed/ch

   <title>Haitians pray, cry for help in the ruin
   <link>http://hosted.ap.org/dynamic/stories
   <description>PORT-AU-PRINCE, Haiti    (AP)
   <pubDate>Mon, 18 Jan 2010 04:32:47 GMT
   <author>By MICHELLE FAUL and JENNIFER K
   <guid>http://hosted.ap.org/dynamic/storie
</item>
<item xmlns:yahoomaps="urn:yahoo:maps"
<yahoomaps:ResultSet xmlns:xsi="http://ww
```

Cameron's `Avatar' wins b
BEVERLY HILLS, Calif. (AP) -- The scienc
blockbuster "Avatar" won best drama at the G
Globes and picked up the directing honor for J

Obama seeks to save Sena
BOSTON (AP) -- His agenda at risk, Presid
Barack Obama fought Sunday to save a strug
Democratic U.S. Senate candidate and the cri

Elderly and abandoned, 84
PORT-AU-PRINCE, Haiti (AP) -- The old la
crawls in the dirt, wailing for her pills. The elde
lies motionless as rats pick at his overflowing

Attorney: Man in JFK brea

Fig. 9. Dragging and dropping a data model on a widget

To briefly analyze this process, it can be still enhanced by removing the need to add inputs and outputs manually – it is only required because the current implementation lacks functionality in this respect. However, even at the prototype stage of the tool, this sample outlines that only a few minutes of work is required to build a simple but fully working program that is able to display data. Experience has shown that most cases are not so simple, but even in non-trivial cases, the majority of time was spent in fetching the desired data from a broken XML-like file and reformatting it to make it compliant. Such examples included generating route suggestions from a Finnish national railway web service (http://www.vr.fi), as well as using a Finnish combined public transit website (http://www.matka.fi) for a similar task. Writing scripts to parse the data took several hours in both cases, which illustrates one of the standing problems with the whole concept of mashups.

5.2 GPS Coordinates from Addresses of Contacts in Local Database

The second sample application uses a database called Piglet, which is a triple-store database originating from Wilbur [8], a toolkit for programming web applications with XML and Resource Description Framework (RDF) [9] support. Piglet is also based on RDF and as such, its user can add any kind of metadata into any kind of object. Out Piglet database included a contact book, in which each contact had an address in textual format. This text could then be sent to a Yahoo's Geocoding service in order to fetch the location in GPS coordinates. Each user is also associated with a distinctive URI within Piglet, which allows users to point and click on the points of interest on the map, opening up more information about each user. However, a similar scenario is loading up any data with a postal address on a map, so this example is not restricted to a certain data engine. The illustrations below display creating this mashup.

The user first adds a Piglet element to the canvas and adds an XML namespace filter to select only contacts (Figure 10). Then, the user can add a new output and create an XQuery for it to filter out everything but the addresses. Next, a Geocoder is added, and connected to the Piglet's address output (Figure 11). Finally, a script element is added, using a template XMLMerge, which takes two XML inputs, called *inner* and *outer*, and inserts the *inner* input's element into the *outer* data source. The element names can be specified in the properties (Figure 12).

After connecting the Geocoder result node with the *inner* input and the unmodified data from Piglet in the *outer* input (Figure 13), the mashup is finished. The differences lie just in usage of data sources and scripts. Figure 14 shows what a compiled program using the created mashup looks like.

The mashup engine library heavily relies on XML parsing and QtScript, both of which are likely to cause a performance penalty. The first sample application, combining two RSS feeds into one and displaying it, was implemented with standard Qt for comparison. Because the performance hit happens almost exclusively on initialization when plugins have to be loaded and mashup files have to be parsed, the initialization time for a mashup-engine powered feed displayer was measured – 2227ms on average. A pure C++ implementation took only 1149ms, so the execution time is nearly doubled. Most of the extra time accounted for is spent loading external libraries that handle additional QtScript functionality provided by QtScriptGenerator and parsing XML and script files. Were these additional QtScript libraries part of Qt itself, this loading time would not be experienced. In the light of these numbers, a more effective instruction storage format is also desirable over XML, together with optimizations to the mashup engine itself, in particular regarding library loading.

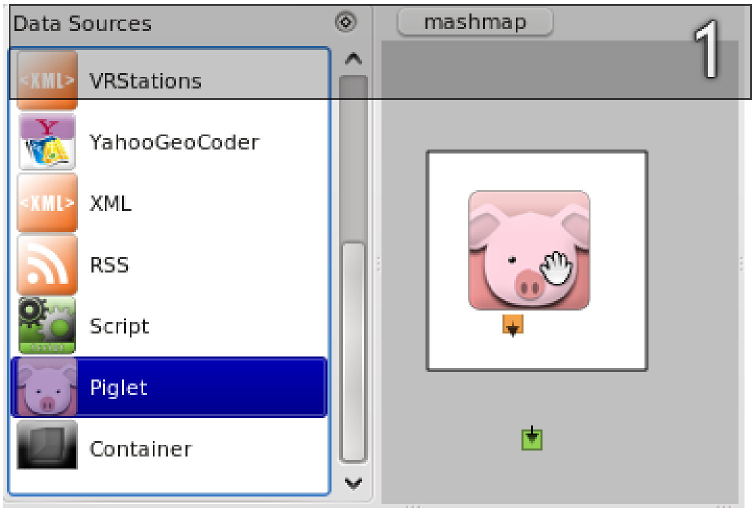
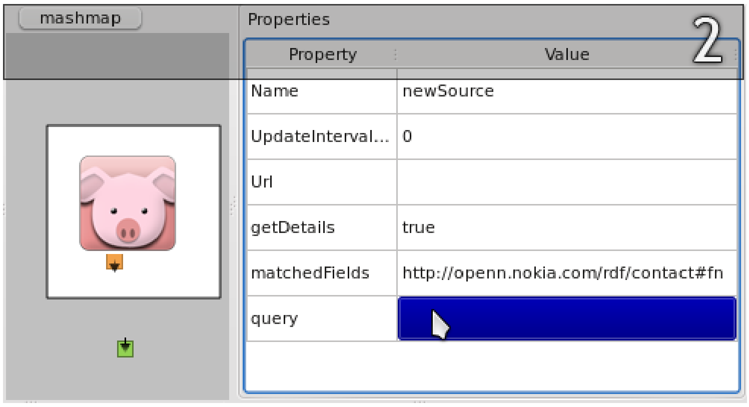


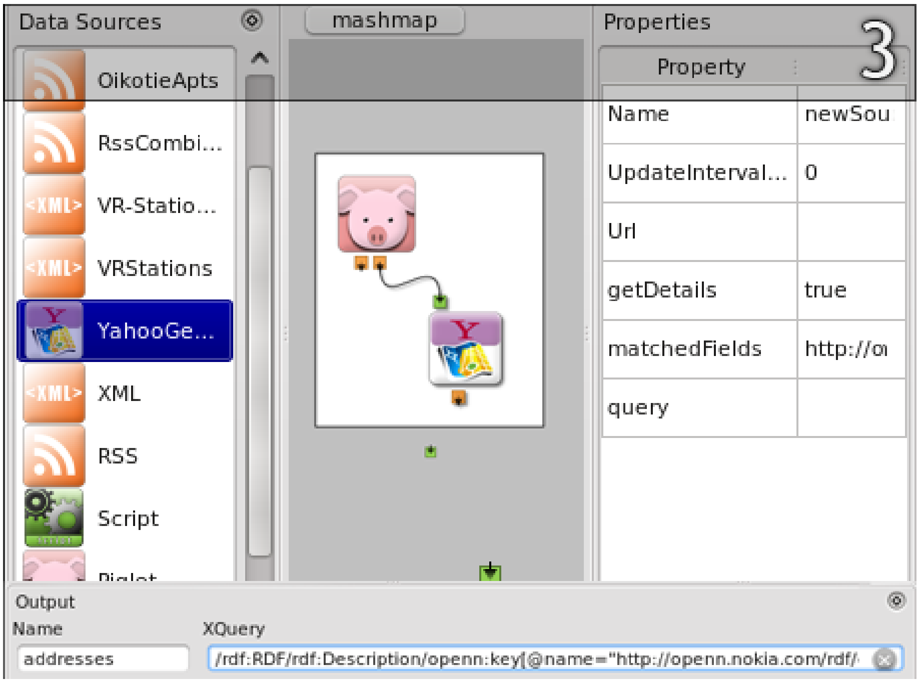
Fig. 10. Creating Piglet data source



Property	Value
Name	newSource
UpdateInterval...	0
Url	
getDetails	true
matchedFields	http://openn.nokia.com/rdf/contact#fn
query	

Fig. 11. Modifying Piglet data source properties

The vanilla implementation of the RSS feed reader was approximately 500 lines of code. Most of the code deals with XML processing and delegate rendering that are non-reusable by themselves. With the mashup engine and the Qt Creator extensions, the required amount of C++ programming is reduced to *zero* lines, provided that the wizards in Qt Creator are used. Even without the wizards, the user only has to write about 4 code lines per program, consisting of boilerplate initialization code.



Property	
Name	newSou
UpdateInterval...	0
Url	
getDetails	true
matchedFields	http://o
query	

Name	XQuery
addresses	/rdf:RDF/rdf:Description/openn:key[@name="http://openn.nokia.com/rdf/

Fig. 12. Adding a Geocoder and fetching address data from the Piglet

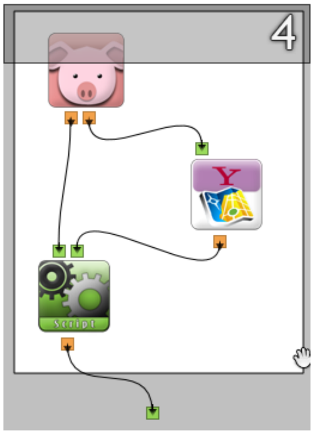


Fig. 13. Mashup that takes address data of contants, gets GPS coordinates, and appends them

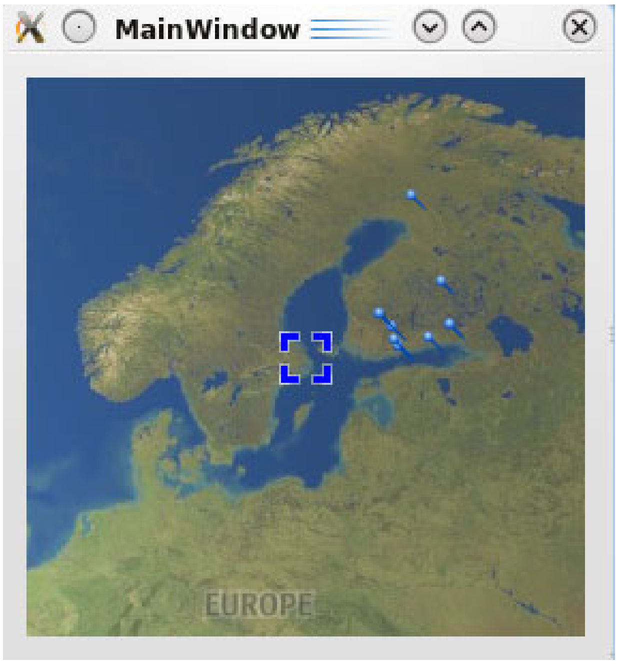


Fig. 14. Finished application using contacts address mashup

6 Discussion

At present the tool we have implemented is experimental software, and it is not ready for prime time use. However, even in its present condition it demonstrates that one can rapidly develop client-side mashups that combine data from the Internet. Moreover, since we are using a special-purpose runtime environment, the tool is not bound to the restrictions of the browser but can freely access different web sites and internal data that is only available in the device.

6.1 Implementation Restrictions

There are certain use cases to which even the current prototype of the mashup editor is more than suitable for. Modifying data with XQuery and ECMAScript is easy for anyone with a rudimentary understanding of the said technologies, and for those who are not, templates are available for use. Consequently, extending functionality is easy, which is something that most of the existing mashup systems have also done well. Map mashups could be argued to be easier than with competing technologies, as coordinate-containing data can be simply dragged and dropped on the widget. For interaction, normal programming is still required, but Qt Designer provides some ailments to this respect as well, namely visual connection of signals and slots. Displaying valid XML-compliant data is also very easy, provided that there is a delegate available. However, therein lays one of the biggest drawbacks with the current approach - the difficulty around creating delegates for item view widgets.

Some item views, such as a map widget, do not even require additional delegates to display data, but for the majority of cases, this poses a greater problem as displaying even text is difficult. Although initially planned, no delegate editor was built as Nokia changed the development direction of Qt, including next-generation item views, which did not follow the Model-View-Delegate pattern.

In addition, a declarative UI toolkit especially targeting mobile devices, called Qt Quick, is being built as well, which allows for creating interfaces with a simple markup, in addition to a GUI editor, reuse of created interfaces, as well as programming of interaction within the UI. One of the core differences between traditional and declarative development is that the latter is based on 2D graphics rendering, while the former is based on standard widgets provided by the underlying operating system. As such, the declarative approach allows for far more flexibility and possibilities, while the widgets make it possible for the UI to adapt to the operating system being deployed to. As a language, Qt Quick resembles JSON notation of JavaScript and it has been perceived as easy to learn also to those with no prior JavaScript experience, due to its very hierarchical structure.

Currently, Qt Quick seems to be the most interesting future development direction. The mashup editor and Qt Quick seem to complement each other as technologies, as Qt Quick currently supports displaying data from models, and building UIs around already available data requires no C++ knowledge, further easing the development of mashup applications. In addition, making mashups even with plain Qt Quick is possible, thanks to its XML and XQuery-processing capabilities. In real-world stress tests, Qt Quick performs 60 FPS on devices such as N900 without additional optimization work, so the CPU power of the device is freed for data acquisition and processing for the need of mashups and application business logic.

For what comes to the implemented Qt Creator extensions, they are on the prototype stage. While they work reasonably well on both OS X 10.6 and Ubuntu 9.10 with distribution packages also created, a fair amount of work is to be done for productization. The groundwork is laid out, however, and the mashup engine itself is only in need of performance optimizations.

6.2 Mashup-Related Restrictions

Another, much more difficult problem is related to the nature of mashups. That is, while manipulating data from various sources, there is little to no guarantee that the structure of the data would not suddenly change due to an upgrade in the corresponding web site. As most of the cases where mashups are desired, are fetching data from a Web source, a widget could stop working if the web author even slightly modifies the structure of the site markup. The mashup templates were created to reduce the influence of this problem and the idea was that such templates could be easily shared through a web service between users of the mashup tool. Upon a content change, any community member could then proceed to fix the affected mashup files and other users could then update their repositories respectively. While this approach does not even attempt to defeat the original problem, there is no definite solution – for instance, a web page author might move the content to an entirely different location, rendering even advanced heuristic-based content retrieval attempts futile.

A further typical mashup related problem associated with the nature of the web sites is that they are usually not valid HTML or XHTML. Research shows that only a fraction of all pages on any given site are completely valid [10], which results in errors in parsers that do not offer error processing facilities. While the native XQuery support for Qt is otherwise adequate, it offers minimal error handling facilities, making it difficult to use with a great many test cases. Experience has shown that regular expressions and even concatenated substring matches are much more effective at finding the desired data. While a visual editor for generating these does not exist yet, it is one of the future development targets.

6.3 Tools and Technologies

Comparable tools exist in terms of visual data creation, like Yahoo Pipes. The difference lies in web-based interface of Yahoo Pipes, and the fact that Pipes is concentrated solely on data mashups, as opposed to the mashup tool built on top of Qt Creator, which also enables usage of created data in user interfaces. A limited comparison can be also made to tools that help creating desktop widgets, such as Dashcode for OS X. Dashcode offers powerful tools modifying the visual appearance of widgets and programming them with JavaScript. While web-based data can be used in the interface, such as RSS feeds, the data itself cannot be modified in a visual way as in the Qt Creator extensions. However, it can be argued that Dashcode's UI tools are easier to use for novices than those in Qt Designer – a test with Dashcode to create an RSS feed with a personalized interface took less than half an hour, with no prior experience with the tool. On the other hand, creating a similar application with vanilla Qt Designer amounts to hours even from an experienced developer, due to the fact that data has be set into the widgets programatically. The mashup extensions set Qt Creator on par with tools like Dashcode, as it is only a matter of minutes to build a data model from web sources and assign it to a view.

Compared to other technologies, client-side mashups created with Qt are associated with higher performance and a more complete access to the operating system if needed through the powerful Qt API. As any existing web applications can be readily utilized through Qt's WebKit API with similar performance to web browsers utilizing the WebKit, such as Apple Safari, Qt seems to be an ideal choice, if additional client-side functionality is desired. Obviously, when considering a client-side mashup runtime environment, it is clear that security features need attention before large-scale use. At present, considerations on how a convenient model could be created have been based on J2ME [11], but we do of course acknowledge that in that context the distribution model of applications is completely different.

Another approach for a mixture of web and native content would be implementation of a plug-in for web browsers that is able to render Qt content. Building a basic version of such a plugin is not difficult, but problems arise in particular with security. Furthermore, using web browser as a platform again somewhat defeats the purpose of client-side mashups, although gained performance that arises from the use of Qt can always be seen as a positive side. Additionally, constructing websites with an unlimited set of tools, as opposed to the current approach of modifying document-based content with scripts to make websites look more functional, is attractive by itself.

Competing technologies such as Flash, often used for mashup as well as web application development, are thought of as an easier alternative than C++. Research supports that C++ is difficult to learn [12] and because of its complexity, it is also less productive. With the Mashup Editor, new UI technologies, scripting and other advances, these problems may be eliminated or at least become less pronounced. Even with the current, experimental QtScript bindings for the GUI widgets and such, it is possible to build complete applications with pure QtScript. While script-side documentation and development tools still need improvement, it opens up new possibilities for cross-platform development.

7 Conclusions

In this paper we have discussed a mashup editor for client-side mashups that are well-suited for mobile environment. The system was developed using tools and techniques that the Qt framework provides. Even in its current state, the Qt Creator extensions can be used to successfully create simple applications with a considerable reduction in the lines of code – for basic applications that simply display data, no code has to be written at all.

The comparison between a mashup engine-driven and a vanilla Qt implementation reveals performance issues, but also a promise of better reusability and reduced effort in software development. While the vanilla implementation does not use many lines of code either in absolute terms, it can be argued to be a considerable difference to a developer new to C++ development, which is typically considered difficult.

Interesting future directions also include better integration to device-specific facilities such as a cell phone contact book and data engines provided by the operating system, such as Akonadi (http://pim.kde.org/akonadi/) available on the K Desktop Environment (http://www.kde.org/). Moreover, provided with a declarative fashion to compose user interfaces in the form of Qt Quick, the developer is offered an extended toolset for the development of interesting, interactive mobile mashups.

References

1. Mikkonen, T., Taivalsaari, A., Terho, M.: Lively for Qt: A Platform for Mobile Web Applications. In: Proceedings of the Sixth ACM Mobility Conference, Nice, France, September 2-4 (2009)
2. Nyrhinen, F., Salminen, A., Mikkonen, T., Taivalsaari, A.: Lively mashups for mobile devices. In: Proceedings of the MobiCase 2009 Conference, San Diego, CA, USA, October 26-29 (2009)
3. ECMA Standard 262: ECMAScript Language Specification, 3rd edn. (December 1999)

4. Yahoo Inc. Yahoo! Maps Web Services – Geocoding API (2010)
5. Nokia Corporation. Qt 4.6: View Classes (2010), `http://qt.nokia.com/doc/4.6/model-view-view.html` (reviewed January 18, 2010)
6. Nokia Corporation. Qt 4.6: An Introduction to Model/View Programming (2010), `http://qt.nokia.com/doc/4.6/model-view-introduction.html` (reviewed January 18, 2010)
7. Nokia Corporation. Qt 4.6: QPicture Class Reference (2010), `http://qt.nokia.com/doc/4.6/qpicture.html` (reviewed January 18, 2010)
8. Lassila, O.: Enabling Semantic Web Programming by Integrating RDF and Common Lisp. In: Proceedings of the First Semantic Web Working Symposium. Stanford University (July 2001)
9. World Wide Web Consortium. Resource Description Framework (RDF) Model and Syntax Specification. World Wide Web Consortium (1999)
10. Marincu, C., McMullin, B.: A comparative assessment of Web accessibility and technical standards conformance in four EU states. First Monday 9(7-5) (2004)
11. Riggs, R., Taivalsaari, A., Van Peursem, J., Huopaniemi, J., Patel, M., Uotila, A.: Programming Wireless Devices with the Java 2 Platform, Micro Edition, 2nd edn. Java Series. Addison-Wesley (2003)
12. Lahtinen, E., Ala-Mutka, K., Järvinen, H.-M.: A Study of the Difficulties of Novice Programmers. In: Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, Caparica, Portugal, pp. 14–18 (2005)