Lively for Qt: A Platform for Mobile Web Applications

Tommi Mikkonen
Tampere University of Technology
Korkeakoulunkatu 1
FI-33720 Tampere, FINLAND
tommi.mikkonen@tut.fi

Antero Taivalsaari
Sun Microsystems Laboratories
P.O. Box 553 (TUT)
FI-33101 Tampere, FINLAND
antero.taivalsaari@sun.com

Mikko Terho
Nokia Devices
Visiokatu 3
FI-33720 Tampere, FINLAND
mikko.j.terho@nokia.com

ABSTRACT

The convergence of desktop, mobile and web application development has resulted in new types of software systems. These new systems are built to leverage the World Wide Web, and they allow the use of dynamically downloaded applications and services from any type of terminal, including desktop computers and mobile devices. At the same time, the JavaScript language has become the *de facto* programming language of the Web. In this paper we introduce *Lively for Qt* – a practical JavaScript application platform that supports the development of highly interactive mobile web applications and mashups that run both in the web browser and as standalone "phonetop" applications. Additionally, we present our vision for mobile web applications, as well as summarize the experiences and lessons learned during the development of the new system.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques – *modules and interfaces, software libraries, user interfaces.* D.2.6 [**Software Engineering**]: Programming Environments – *graphical environments, integrated environments, interactive environments*.

General Terms

Design, Experimentation, Languages.

Keywords

Web applications, mobile software, Lively for Qt.

1.INTRODUCTION

The convergence of desktop, mobile and web application development has resulted in new types of software systems and runtime environments. These new systems are built to leverage the World Wide Web, and they allow applications and other content to be downloaded dynamically from the Web. Applications commonly run in the web browser and do not necessarily require any explicit installation. Furthermore, these new systems enable the use of applications and services from any type of terminal, including desktop computers and mobile devices. Despite the problems that arise from CPU speed differences, memory capacity differences, network bandwidth limitations and screen size variations, it is safe to assume that it will eventually be possible to use the same web applications and services in desktop computers and mobile devices.

Given that JavaScript programming language support is included in every commercial web browser, JavaScript [5] has rapidly become the *de facto* programming language of the Web. Although JavaScript was originally designed as a scripting language – a language targeted to relatively simple scripting tasks – its use has rapidly spread to "real" programming tasks, serious application development and even systems programming. These days, it is not uncommon to see JavaScript applications that consist of tens of thousands of lines of code.

This paper introduces *Lively for Qt* (http://lively.cs.tut.fi/qt) – a practical JavaScript application platform that supports the development of compelling, highly interactive mobile web applications and mashups that run both in the web browser and as standalone "phonetop" applications. The system runs on top of *Qt* (http://www.qtsoftware.com/) – a cross-platform application framework that was recently acquired by Nokia. Versions of the Qt framework have already been announced for Nokia's device platforms.

In addition, we present our vision for mobile web applications, as well as briefly summarize the experiences and lessons learned during the development of the Lively for Qt system. The work builds upon our earlier experiences in developing the *Sun Labs Lively Kernel* (http://research.sun.com/projects/lively) [7, 13] – a "zero-installation" web application platform that runs in the standard web browser without any installation or plug-in components. The key idea in creating the Lively Kernel was to implement a highly interactive application development environment, which – from the end user's viewpoint – is just a web page. Our earlier efforts on porting the Lively Kernel to mobile devices have been presented in [9].

The Lively for Qt system introduced in this paper is the logical successor of the Sun Labs Lively Kernel in that it implements a JavaScript application environment that is capable of hosting Rich Internet Applications that are fully interactive and malleable; in short, lively. However, a key difference between the two systems is that in Lively for Qt we leverage an existing, rich, mature, well-documented application framework as the basis of the system. In the Lively Kernel, the development APIs offered by the underlying platform (the web browser) were far more limited. Another important difference is the increased focus on mobile devices. Although the Lively for Qt system can run on a wide variety of target platforms, including conventional desktop computers, our goal is to build a practical web application and mashup environment specifically for mobile devices.

The structure of this paper is as follows. Section 2 introduces our vision for mobile web applications, and provides background and general motivation for the rest of the paper. Section 3 provides an overview of the Qt application framework, and then introduces the Lively for Qt system. Section 4 introduces some sample applications and discusses the development style associated with them. Sections 5 and 6 discuss related work and revisit the vision presented in Section 2 in order to summarize the experiences and lessons learned in the scope of the vision. Finally, Section 7 concludes the paper.

2.A PLATFORM FOR MOBILE WEB APPLICATIONS: THE VISION

The authors of this paper have considerable experience in designing and building practical application platforms for mobile devices. The first author was closely involved in the introduction of the Symbian platform for Nokia devices. The second author was the original designer behind the Java ME platform [12] at Sun Microsystems. The third author was the co-lead of the original Nokia Communicator project – the very first smartphone in the world.

We believe that next-generation mobile application platforms will be built to leverage the Web from the very beginning. Unlike the earlier mobile application platforms, in which applications were delivered mainly in binary (native) form, in the future applications will be downloaded dynamically and used without explicit installation. Applications can be deployed instantly worldwide. Ideally, applications should simply run in the web browser, or a similar web client environment that has been specifically tailored for mobile devices; the applications should be fully compatible with full-fledged web applications running in desktop web browsers.

Unfortunately, there are still major obstacles in creating a platform that realizes the “One Web” objective advocated by the World Wide Web Consortium [10]. Even though it is fairly safe to assume that eventually the same web applications and services will run in desktop computers and most other types of client devices, it is not yet feasible to run complex web applications (e.g., complex Ajax applications) inside mobile devices using a web browser. For instance, as we reported earlier [9], running a moderately sized JavaScript application inside the web browser of a mobile device turned out to be impractical due to performance reasons; a moderately sized application took over a minute to load, and event handling in interactive applications often took a few seconds per each pointer or key event. Therefore, there is still a need for a more intimate coupling between the native capabilities of the mobile device and the web application platform.

We believe that in the next two to five years, the winning recipe for a commercially successful mobile web application platform is the following:

1. Take a rich graphics framework that supports a rich suite of widgets and provides programmatical support for direct, interactive manipulation and flexible graphical transformations such as scaling, rotation, and so on. Ideally, 3D graphics should be supported as well.
2. Take JavaScript, the world's most widely used dynamic language, and make the graphics APIs available to it; use a modern, high-performance JavaScript virtual machine, so that there will be enough power for serious applications.
3. Make it easy to access the Web from the platform using asynchronous HTTP networking and other commonly used networking standards; provide JavaScript APIs for processing XML, JSON (JavaScript Object Notation), DOM and other frequently used formats and structures easily.
4. Add support for mobile-specific JavaScript APIs for areas such as wireless messaging, location, Bluetooth and camera support. Such APIs are familiar, e.g., from the Java ME™ Mobile Service Architecture (MSA) Specification [8].
5. Make the platform available both inside the web browser and natively. All the applications should be able to run both inside the web browser and as “phonetop” applications that behave like native applications.
6. Add a fine-grained security model so that network-downloaded applications can be executed safely; provide minimal access to APIs for those applications that are downloaded from untrusted sources, and more extensive access to APIs for applications from trusted sources.

Although our vision above revolves heavily around web technologies, we believe that the development style used in the new platform should not be based purely on declarative web technologies such as HTML or CSS. Rather, there is a need for *procedural development style* and programmatic APIs as well. Currently, the JavaScript APIs offered by the web browser are still extremely limited compared to APIs offered by mature software platforms such as Microsoft Windows or MacOS X. The APIs are also very limited compared with the APIs of mature mobile software platforms such as the Symbian OS or the Java ME platform. The need for rich, mature, well-documented APIs supporting a procedural programming style is largely why we have chosen the Qt framework as a starting point for the Lively for Qt system introduced in the next section.

3.LIVELY FOR QT

In this section we present the *Lively for Qt* system that is the current manifestation of our vision described above. We start with an introduction of the Qt framework, and then introduce the key components and features of Lively for Qt. In later sections of the paper, we assess the Lively for Qt system in light of the vision presented above, as well as compare the system with competing platforms in this area.

3.1. Introduction to Qt

Qt (http://www.qtsoftware.com/) is a mature, well-documented cross-platform application framework that has been under development since the early 1990s. Qt supports a rich set of APIs, widgets and tools that run on most commercial software platforms, including Mac OS X, Linux and Windows. Examples of desktop applications built with Qt include Adobe Photoshop Elements, Google Earth, Skype and the KDE desktop environment for the Linux operating system. In addition, Qt has been used in various embedded devices and applications, including mobile phones, PDAs, GPS receivers and handheld media players.

General features of the Qt framework. From the technical viewpoint, Qt is primarily a GUI framework that includes a rich set of widgets, graphics rendering APIs, layout and stylesheet mechanisms and associated tools that can be used for creating compelling user interfaces that run in a wide array of target platforms. Widgets range from simple objects such as push buttons and labels to advanced widgets such as complete text editors, calendars, and objects that host a complete web browser. Dozens and dozens of widget types are supported.

The GUI features of Qt adapt to the native look-and-feel of the target platform. For instance, on Mac OS X, all the widgets look like native Macintosh widgets, while on Windows applications utilizing the same widgets will look like native Windows applications. An essential part in enabling cross-platform GUI behavior is flexible support for widget positioning using *layouts*. Qt's layout components can adapt to different sizes, styles and fonts used by the host operating system. In general, layouts give significant advantage when the program is translated to other platforms and languages. The program adapts automatically to changed text sizes and resizes widgets in an aesthetically pleasant way. Additionally, since Qt supports full *internationalization*, all the locale-specific components (such as a calendar widget) automatically adapt to the current regional settings of the target platform.

In addition to its GUI capabilities, Qt offers APIs for networking, file access, database access, text processing, XML parsing and many other useful tasks. A multimedia framework called *Phonon* is included to support audio and video playback. Qt networking libraries provide support for *asynchronous HTTP communication* familiar from Ajax [2].

Qt and web development. What makes Qt relevant from the viewpoint of web development is that Qt libraries include a complete web browser based on the WebKit (http://webkit.org/) browser engine. The necessary DOM and XML APIs are also included to parse, manipulate and generate new web content easily. In addition, Qt includes a fully functional ECMAScript [4] (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 [6].

The web browser integration in Qt works in a number of different ways. For instance, it is possible to instantiate any number of web browsers inside a Qt application using the *QWebView* API. The *QWebView* class provides a widget that can be used to view and edit web documents inside applications. The data in web documents can be manipulated using the built-in DOM and XML APIs. To support Qt applications that run in a web browser, a plugin called *QtBrowserPlugin* exists for embedding the Qt environment into any commercial web browser such as Mozilla Firefox or Apple Safari. The plugin makes it possible to run Qt applications inside a web browser, either as standalone Rich Internet Applications or alongside (or embedded in) conventional DHTML and Ajax web content.

Qt and mobile devices. Trolltech, the company developing Qt, was acquired by Nokia in 2008. Nokia is currently in the process of making Qt libraries available on their phone platforms. Nokia's market share will make Qt an extremely interesting target platform for mobile applications as well. Soon after the Trolltech acquisition, Nokia announced support for Qt on the Maemo Linux (http://maemo.org/) and Series 60 Symbian (http://www.s60.com/) platforms.

3.2. Lively for Qt – Introduction

Lively for Qt (http://lively.cs.tut.fi/qt) – or more formally the Lively Kernel port for the Qt platform – is a web application platform that runs on top of the Qt framework. The system supports the development of JavaScript applications that run both in the web browser and as standalone mobile "phonetop" applications. Applications can leverage the rich APIs offered by the Qt platform, and they support direct manipulation, instant display updates and all the typical 2D graphics capabilities familiar from desktop computing environments. Applications can be downloaded dynamically from the Web, and they require no explicit installation.

In addition to its application execution capabilities, Lively for Qt can also function as an integrated development environment (IDE). These features are familiar from the Sun Labs Lively Kernel [7, 13] – the precursor of Lively for Qt. Lively for Qt includes a number of tools such as a JavaScript class browser, object inspector and source code debugger that allow applications to be edited and modified on the fly. Although the use of such interactive development tools is not very practical on mobile devices with limited screens and keyboards, the presence of reflective capabilities means that applications can also be modified over the air or via a USB cable. Such capabilities can be extremely convenient, e.g., for remote debugging or bug fixing of applications that have been downloaded earlier.

Lively for Qt is available under an MIT-style open source license. The system can run on any platform for which a port of the Qt framework is available. As a target device for our mobile porting activities, we have used the Nokia N810 device – an internet tablet that has a high-resolution (800 by 480 pixel) color display and a touch screen. It is possible to use the Lively for Qt system also on all the major desktop operating systems, including MacOS X, Linux and Windows. Furthermore, the system can run in any web browser using a Qt browser plug-in.

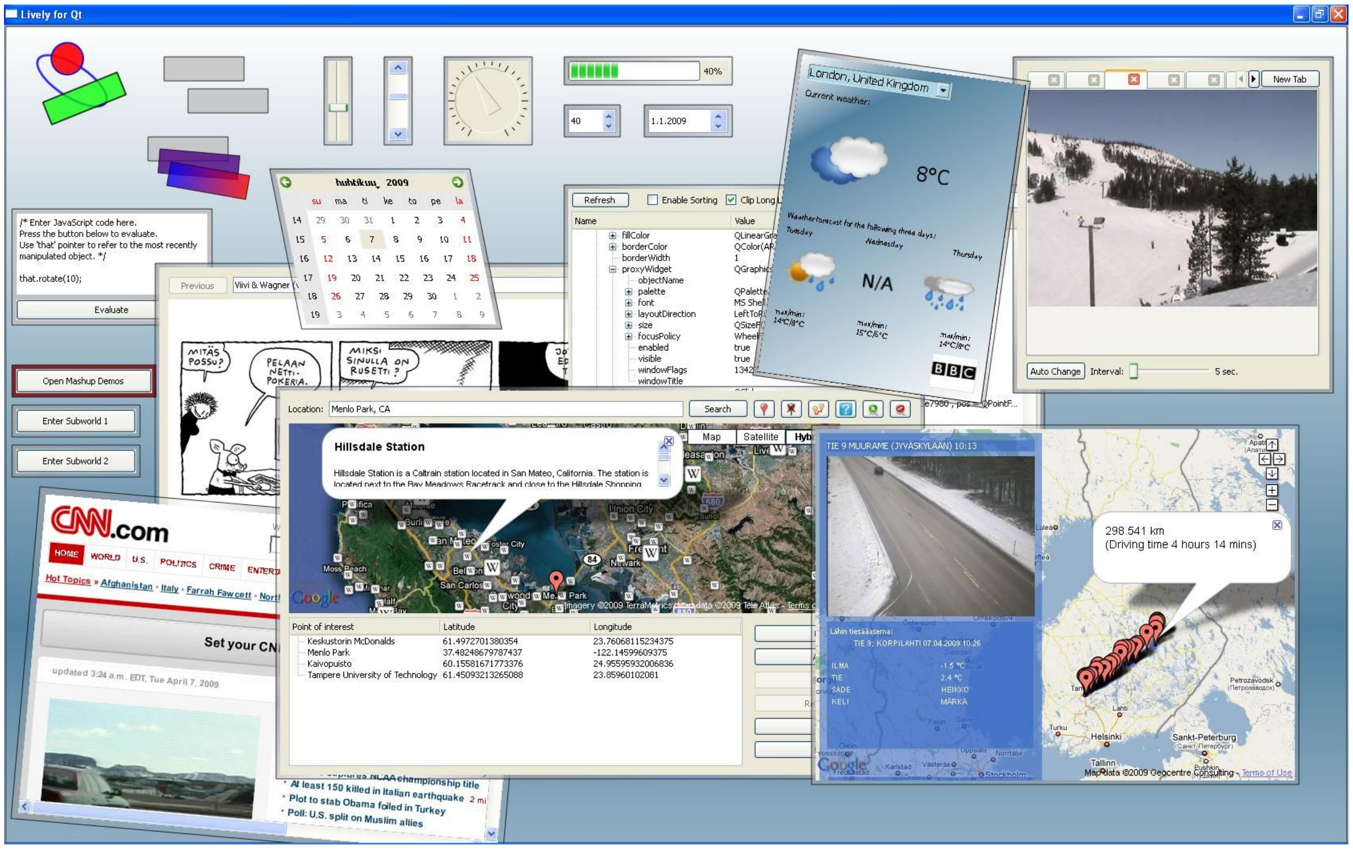
A screen snapshot of Lively for Qt running on a PC is shown in Figure 1. This snapshot shows Lively for Qt running a number of applications, widgets and tools simultaneously. All the objects on the display are fully interactive and changeable; in short, lively.

3.2. Lively for Qt – Key Components

Architecturally, Lively for Qt follows the high-level design of the Sun Labs Lively Kernel in that it consists of the following three key components:

1. *A JavaScript engine*. We have used a JavaScript engine as a fundamental building block for the system. The Lively for Qt system itself, as well as all the applications and tools have been written in JavaScript.

   The JavaScript support in Lively for Qt is built around the *QtScript* scripting engine that has been part of the Qt framework since version 4.3. In the current version of Qt



```
/* Enter JavaScript code here.
Press the button below to evaluate.
Use 'that' pointer to refer to the most recently
manipulated object. */

that.rotate(10);
```

Hillsdale Station is a Caltrain station located in San Mateo, California. The station is located next to the Bay Meadows Racetrack and close to the Hillsdale Shopping

Point of interest	Latitude	Longitude
Keskustorin McDonalds	61.4972701380354	23.76068115234375
Menlo Park	37.48248679787437	-122.14599609375
Kaivopuisto	60.15581671773376	24.95595932006836
Tampere University of Technology	61.45093213265088	23.85960102081

Figure 1. Lively for Qt running a number of applications, widgets and tools simultaneously

(version 4.5), the JavaScript engine is still built around a simple interpreter, and is therefore rather slow. However, future versions of Qt will include a modern, high-performance JavaScript virtual machine that leverages just-in-time compilation.

2. *Desktop quality graphics APIs*. As in the Lively Kernel, we believe that a fundamental requirement for a modern web programming environment is the availability of a powerful graphics API that provides support for direct drawing, direct manipulation, and a rich set of predefined graphical widgets. Furthermore, for improved efficiency and development flexibility, we believe the APIs should support *procedural* (and not only declarative) development style.

   The graphics capabilities of Lively for Qt have been implemented on top of the *QGraphicsView* framework provided by Qt. This framework makes it possible to define graphical objects that are managed automatically by Qt. Repainting, Z-order management, layouts, stylesheets and graphical transformations such as object rotation or scaling are supported out of the box. Furthermore, the system supports dozens and dozens of predefined graphical widgets that can be used programmatically from JavaScript.

3. *Asynchronous HTTP networking*. All the networking requests in Lively for Qt are performed asynchronously, utilizing the *QNetworkRequest* and *QNetworkAccessManager* classes of Qt. The use of asynchronous networking is critical in modern web environments, so that network requests can be performed without any adverse impact on the interactive response of the system.

In addition to the three key building blocks summarized above, Lively for Qt includes a large number of additional APIs that are courtesy of the underlying Qt framework. The APIs (originally written in C++) have been made visible to the JavaScript environment using a tool called *QtScriptGenerator* (http://labs.trolltech.com/page/Projects/QtScript/Generator).

Furthermore, as was mentioned already earlier, the system includes various interactive development tools (e.g., a source level debugger supporting breakpoints, single stepping and call stack visualization) that can be used for inspecting and modifying the system on the fly.

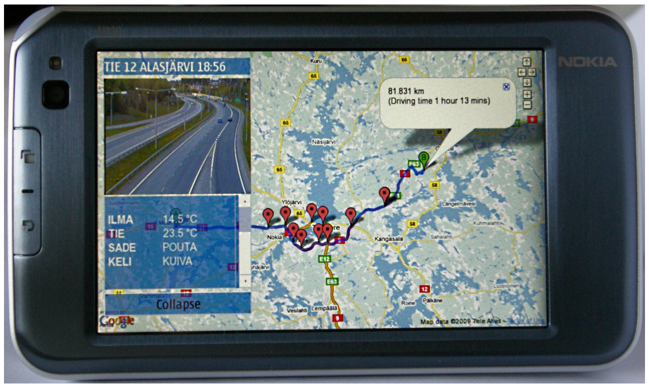
In the implementation of Lively for Qt, we have utilized a JavaScript library called *Prototype* (http://www.prototypejs.org/). The Prototype library provides numerous convenience functions and additional syntactic sugar for defining JavaScript classes in a more structured fashion. Application development principles with Lively for Qt will be discussed in more detail in the next section.

4. APPLICATIONS AND DEVELOPMENT STYLE

We have built a large number of applications with Lively for Qt. Since Lively for Qt is a web application environment, most of these applications utilize various existing, widely-used services available on the Web. Many of the applications are *mashups* that combine data from multiple web sites. The applications range from map-based applications to image and comic scrapbooks, media players and games. Below we briefly summarize some of the applications and discuss the development style used in developing them.

4.1. Sample Applications

QtWeatherCameras: Live Road Weather. The *QtWeatherCameras* application shown in Figure 2 is a mashup that utilizes the Google Maps JavaScript API [6] and the live road weather camera information available from Finnish Road Administration (http://www.tiehallinto.fi/). The application uses the Google Maps API to calculate an optimal route between two chosen points on the map of Finland. The application then obtains information about the nearest road weather cameras along the route, and displays those cameras as markers on the map (see Figure 2). When the user clicks on any of the markers on the map, a live image and current weather conditions from the selected camera are fetched.



ILMA	14.5 °C
TIE	23.5 °C
SADE	POUTA
KELI	KUIVA

Figure 2. *QtWeatherCameras* application

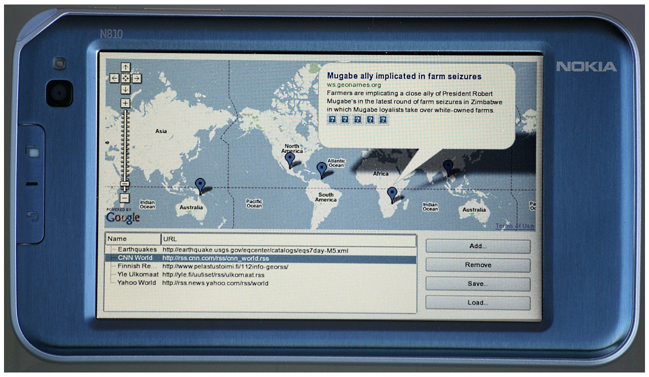
QtMapNews: Geotagged RSS Feed Viewer. The *QtMapNews* application shown in Figure 3 is a mashup that displays geotagged news items and other geotagged information utilizing the Google Maps API. The application includes a *QTreeWidget* (tree view) component that lists a selection of predefined geotagged RSS feeds:

– Earthquakes: All the Magnitude 5 or greater earthquakes in the world in the past seven days.

– Emergencies: The last one hundred incidents/emergencies in Finland based on information available from Finnish Rescue Service (http://www.pelastustoimi.fi/).

– News: Geotagged news from CNN, Yahoo, Yle (Finnish Broadcasting Service) and other news services around the world.

The user can add more RSS feeds by pressing a *QPushButton* labeled "*Add...*", which will open a simple dialog to enter a new RSS feed. If the new feed is not a geocoded GeoRSS feed, the *QtMapNews* application uses a publicly available RSS-to-GeoRSS converter service on the Web to geocode items contained within the RSS feed. After the geocoding process, the items in the feed are displayed on the map as markers. When the user clicks on a marker, an overview of the news item is displayed on the map.



Mugabe ally implicated in farm seizures
ws.geonames.org
Farmers are implicating a close ally of President Robert Mugabe's in the latest round of farm seizures in Zimbabwe in which Mugabe loyalists take over white-owned farms.

Name	URL
Earthquakes	http://earthquake.usgs.gov/eqcenter/catalogs/eqs7day-M5.xml
CNN World	http://rss.cnn.com/rss/cnn_world.rss
Finnish Re...	http://www.pelastustoimi.fi/112info-georss/
Yle Ulkomaat	http://yle.fi/uutiset/rss/ulkomaat.rss
Yahoo World	http://rss.news.yahoo.com/rss/world

Figure 3. *QtMapNews* application

An interesting additional feature of the *QtMapNews* application is that it includes an embedded web browser to display more detailed information about the selected map item. Whenever the user clicks on a map item that contains an URL, a web browser view is opened inside the *QtMapNews* application (on top of the map) to display the contents of that web page. The web browser is implemented using Qt's *QWebView* widget that is displayed instead of the map view when necessary.

QtFlickr: Animated Flickr Photo Viewer Based on Twitter Trends. *QtFlickr* (see Figure 4) is a photo viewer application that fetches images from *Flickr* (http://www.flickr.com/) photo service based on keywords (photo tags) that are obtained automatically from the *Twitter* (http://www.twitter.com/) microblogging service, based on current Twitter trends (http://twitter.com/trends). Images are displayed using timer-based animation (rotation).

The general idea of this application is to automatically display images that reflect the most actively microblogged topics in the world. For instance, when the screenshot of the application shown in Figure 4 was taken, the most actively discussed topic in the world was the swine flu (H1N1).

Figure 4. QtFlickr application

4.2. Development Style Illustrated

In Lively for Qt, it is possible to develop applications in two basic ways: (1) using a conventional source code editor or (2) interactively using the tools (such as the class browser or object inspector) built into the Lively for Qt system itself. For mobile devices, application development usually takes place on a separate development workstation. The applications are downloaded to the mobile device dynamically when the execution of the application begins.

The development style associated with Lively for Qt is based on *imperative, procedural development style* familiar from desktop software development. All the application code in Lively for Qt is written in JavaScript, utilizing the rich APIs provided by Qt. This is in contrast with traditional web technologies, which rely heavily on *declarative* languages such as HTML and CSS. In this respect, our applications bear close resemblance to applications developed with Rich Internet Application (RIA) platforms. such as *Adobe AIR* [14] or *Microsoft Silverlight* [11].

Two different coding styles are supported: one based on pure ECMAScript (ECMA standard 262 [4]) without any additional syntactic sugar, or the more structured class definition syntax provided by the *Prototype* library (http://www.prototypejs.org/). The code samples shown here use the former coding style.

```
function FlickrWidget(parent) {

  // FlickrWidget is a subclass of QWidget
  QWidget.call(this, parent);

  // The image references
  this.flickrUrl = 'http://api.flickr.com/'
    +'services/feeds/'
    +'photos_public.gne?format=rss2';
  this.imageUrls = new Array();

  // The visible UI components
  this.currentTagLabel = new QLabel("", this);
  this.imageLabel = new QLabel(this);
  this.imageLabel.setSizePolicy(
    QSizePolicy.Ignored, QSizePolicy.Ignored);
  this.imageLabel.alignment = Qt.AlignCenter;

  this.imagePixmap = new QPixmap();
  this.nextPixmap = new QPixmap();

  // The timers for downloading and rotation
  this.changeTagsTimer = new QTimer(this);
  this.changeTagsTimer["timeout"].connect(this,
    this.changeTagsTimerTimeout);
  this.changeTagsTimer.start(30000); // 30 seconds

  this.fetchImageTimer = new QTimer(this);
  this.fetchImageTimer["timeout"].connect(this,
    this.fetchImageTimerTimeout);

  this.rotTimer = new QTimer(this);
  this.rotTimer["timeout"].connect(this,
    this.rotTimerTimeout);

  this.angle = 90;

  // The layout components
  var hBoxLayout = new QHBoxLayout();
  hBoxLayout.addWidget(new QLabel("Tags:"),0,0);
  hBoxLayout.addWidget(this.currentTagLabel,1,0);
  this.layout = new QVBoxLayout();
  this.layout.addLayout(hBoxLayout);
  this.layout.addWidget(this.imageLabel,1,0);

  this.resize(300,300);
  this.getTwitterTrends();
}
```

Listing 1: The main function (JavaScript class) *FlickrWidget*

In this brief paper there is not enough room for a complete source code example to illustrate the development style in detail. The code sample in Listing 1 illustrates the definition of the main class of the *QtFlickr* application discussed earlier.

Function *FlickrWidget* shown in Listing 1 defines the photo viewer class and its constructor. The class is defined as a subclass of Qt's class *QWidget*, allowing the application to flexibly behave both as a standalone main application (main window) as well as a widget that can be embedded in other Qt components.

The *FlickrWidget* constructor sets up the UI components and connects the components to the required actions. Two layout components are created to arrange widgets within the application window. A *QHBoxLayout* instance is used for horizontally lining up the *QLabel* widgets shown at the top of the application window. A *QVBoxLayout* object then vertically arranges the *QHBoxLayout* object and the *QLabel* object holding the image (*QPixmap*) to be displayed. Two separate *QPixmap* objects are used for images: the first one holds the current image and the second one the image to be displayed next.

Three *QTimer* timer objects are utilized to execute functions in regular intervals. The first timer called *changeTagsTimer* handles the downloading of image tags from Twitter using Qt's *QNetworkAccessManager* class. The second *QTimer* called *fetchImageTimer* is used for downloading the next image asynchronously from Flickr on the background after the current image has been displayed for five seconds. The third timer called *rotTimer* is used for rotating the current image. Qt's *connect* function is used for connecting the timers to the callback functions that are invoked when the timers are triggered.

Listings 2 and 3 illustrate the coding style used in the asynchronous network requests. Function *getTwitterTrends*, shown in Listing 2, is used for obtaining the latest microblogging trends from Twitter. At first a URL pointing to a trend file (a JSON file available from Twitter's web site) is defined. The actual asynchronous HTTP GET request is then sent using the class *QNetworkAccessManager*.

```
FlickrWidget.prototype.getTwitterTrends =
function() {
  var url = 'http://search.twitter.com/'
    + 'trends.json';
  var accessMgr = new QNetworkAccessManager(this);
  accessMgr["finished(QNetworkReply*)"].connect(
    this, twitterReplyFinished);
  accessMgr.get(new QNetworkRequest(
    new QUrl(url)));
}
```

Listing 2: Function *getTwitterTrends*

Parameter *twitterReplyFinished* defines the callback function that will be invoked when the asynchronous network request has been completed. The function *twitterReplyFinished*, shown in Listing 3, processes the JSON file that contains a list of Twitter trends. The JSON string is parsed and the tags in it are stored in an array. When the individual tags have been obtained, a function called *loadImageFeed* is then invoked to load images from Flickr. The *loadImageFeed* function uses the *QNetworkAccessManager* class to download a list of image URLs asynchronously. The definition of the *loadImageFeed* function and its callback function is beyond the scope of this paper; however, the behavior of those functions is analogous to the functions shown in Listing 2 and 3.

```
twitterReplyFinished = function(reply) {
  var trendJSONString =
    reply.readAll().toString();
  var trendJSONObject =
    eval('(' + trendJSONString + ')');
  var tags = new Array();
  for(i=0;i<trendJSONObject.trends.length;i=i+1) {
    tags.push(trendJSONObject.trends[i].name);
  }
  this.loadImageFeed(tags);
}
```

Listing 3: Function *twitterReplyFinished*

To support image animation (rotation), the *QTimer* object stored in the *rotTimer* variable invokes a function called *rotTimerTimeout* (shown in Listing 4) every 50 milliseconds. The function utilizes a *QTransform* object to rotate the currently displayed image. Qt's fast transformation mode is used instead of smooth transformation to improve animation performance on mobile devices at the cost of the quality of the displayed images. If the angle of rotation is 90 or 270 degrees, the image is projected sideways and is invisible to the user. At that point the image can be switched to the next one.

Since image rotation is rather computation-intensive, it is not well suited to low-end mobile devices. We have used it in this application, because it gives a rather realistic view of the limited processing power and the graphics capabilities of the mobile device and its software stack.

```
FlickrWidget.prototype.rotTimerTimeout=function(){

  // When current image is drawn sideways,
  // switch to the next image
  if (this.angle % 90  == 0 ||
      this.angle % 270 == 0) {

    // Switch to the next image
    this.imagePixmap =
      new QPixmap(this.nextPixmap);
  }

  // Calculate transformation (rotation)
  var trans = new QTransform();
  trans.rotate(this.angle, Qt.YAxis);
  trans.rotate(this.angle, Qt.ZAxis);

  // Transform and display the image
  this.imageLabel.setPixmap(
    this.imagePixmap.transformed(
      trans, Qt.FastTransformation));
}
```

Listing 4: Function *rotTimerTimeout*

5.DISCUSSION AND RELATED WORK

Currently there are still major obstacles to the widespread use of mobile web applications. These obstacles revolve around CPU and memory limitations, network bandwidth or cost issues, and even more importantly, screen size limitations. Many web services and applications that run well in a desktop browser are next to unusable in mobile devices.

Over the years, numerous solutions have been proposed to overcome the challenges. These include custom solutions to create the "mobile web", i.e., custom-built web technologies targeted specifically to mobile devices. Such technologies include the infamous Wireless Application Protocol (WAP) (http://www.wapforum.org/), and various special gateways that automatically transliterate and customize web services and content for use in mobile devices with different screen sizes. Many popular web sites today offer special versions of their services and applications that have been tailored to mobile devices.

We believe that eventually there will be "One Web", as envisioned in the W3C Mobile Web Best Practices document [10], meaning that the same information and services should be available to users irrespective of the device they are using.

However, currently One Web is still a dream. Because of various limitations in the mobile space, it is not yet feasible to run full-fledged web applications in a web browser on a mobile device. The performance of the web browser simply is not good enough for mobile use. A partial reason for this is the extensive use of declarative languages, which makes it difficult for the application developer to have any control over performance. Furthermore, the graphics capabilities of the browser – while well-suited for the presentation of document- and form-structured information – are rather inefficient when used for interactive applications. More generally, the lack of rich developer APIs inside the web browser is a serious impediment for developing real, full-fledged applications.

Until the problems above have been solved, a custom runtime environment is required instead of an off-the-shelf web browser in order to create a practical environment for mobile applications. Examples of such custom runtimes include *Adobe AIR* [14], *Microsoft Silverlight* [11] and Sun's *JavaFX* [1]. All these systems offer a rich set of developer APIs that can be used from a dynamic programming language. Our Lively for Qt system is similar to these other Rich Internet Application systems, except in that our system places more focus on interactive development capabilities and on the development of mashups that can flexibly combine content from multiple web sites. Furthermore, the use of Qt has given us additional benefits that are summarized below.

6.BACK TO THE VISION: LESSONS LEARNED AND FUTURE WORK

The Lively for Qt system, in its current form, fulfills the majority of the goals defined as part of our vision presented in Section 2. Basically, we have created a JavaScript application environment that offers rich set of graphics and networking APIs. Additionally, the platform supports a wide variety of other APIs that are provided by the underlying Qt environment. Furthermore, the environment can run on a large number of target platforms, including the web browser using a Qt plugin component.

The use of Qt has given us a number of significant benefits. These include:

– high-performance, cross-platform graphics architecture that supports direct manipulation, efficient event handling, layouts, stylesheets, and flexible graphical transformations such as rotation and scaling;

– a large number of predefined, mature, well-documented widgets that adapt to the native look-and-feel of the target platform;

– built-in web browser support with rich APIs for processing and generating web content.

Without Qt, the implementation of a system offering comparable capabilities would have been dramatically more challenging and tedious.

The current Lively for Qt system is still work in progress. Additional work is required especially in two areas: security and mobile-specific APIs. The security issues in web application development are well known, so we do not delve into details here. Basically, the problems boil down to the lack of a fine-grained security model that could reliably distinguish between trusted and untrusted network-downloaded applications, and grant access to security-critical features and APIs (such as the local file system) accordingly. Insufficient namespace isolation in JavaScript is another major problem. For a more detailed discussion on web application security issues, refer to [3] and the publications of the Open Web Application Security Project (http://www.owasp.org/) and the Web Application Security Consortium (http://www.webappsec.org/).

Mobile-specific APIs such as a camera API, Bluetooth API, SMS messaging, multimedia messaging and global positioning (location) APIs will be part of the Qt environment later, and we will integrate such APIs in our system as soon as they become available. Furthermore, the introduction of a high-performance JavaScript engine in a future version of Qt should give us a major performance boost in running large applications. With a built-in high-performance JavaScript engine, it should become practical to run applications with 3D graphics as well. An interesting example of a scripted 3D graphics environment is available from Trolltech Labs (http://labs.trolltech.com/gitweb?p=WolfenQt).

7.CONCLUSION

This paper has introduced *Lively for Qt*: a practical JavaScript application platform that supports the development of highly interactive mobile web applications and mashups that run both in the web browser and as standalone “phonetop” applications. The system runs on top of *Qt* – a cross-platform application framework that was recently acquired by Nokia. In addition, we presented our vision for mobile web applications, as well as summarized the experiences and lessons learned during the development of the Lively for Qt system.

In conclusion, we believe that web applications will open up entirely new possibilities for software development by making it possible to deploy applications instantly worldwide and to run them without installation. With the work presented in this paper, we have demonstrated that mobile devices can host web applications that support rich user interaction, advanced graphics, integrated development and dynamic application deployment. We hope that this paper, for its part, encourages people to continue the work in this exciting new area.

8.ACKNOWLEDGMENTS

This research has been supported by the Academy of Finland (grant 115485).

9.REFERENCES

[1] Clarke, J., Connors, J., Bruno, E., *JavaFX: Developing Rich Internet Applications*. Prentice Hall (Java Series), 2009.

[2] Crane, D., Pascarello, E, James, D., *Ajax in Action*. Manning Publications, 2005.

[3] Cross, M., *Developer's Guide to Web Application Security*. Syngress Publishing, 2007.

[4] *ECMA Standard 262: ECMAScript Language Specification* (3rd edition, December 1999). Web link: *http://www.ecma-international.org/publications/ standards/Ecma-262.htm*.

[5] Flanagan, D., *JavaScript: The Definitive Guide* (5th Edition). O'Reilly Media, 2006.

[6] Gibson, R., Erle, S., *Google Maps Hacks*. O'Reilly Media, 2006.

[7] Ingalls, D., Palacz, K., Uhler, S., Taivalsaari, A., Mikkonen, T., The Lively Kernel – A Self-Supporting System on a Web Page. In *Proceedings of the 2008 Workshop on Self-Sustaining Systems* (S3'2008, Potsdam, Germany, May 15-16, 2008), Lecture Notes in Computer Science LNCS5146, Springer-Verlag, 2008, pp.31-50.

[8] *Java ME Mobile Service Architecture (MSA) Specification*. Java Community Process Specification Request JSR 248. Web link: http://jcp.org/en/jsr/detail?id=248.

[9] Mikkonen, T., Taivalsaari, A., Creating a Mobile Web Application Platform: The Lively Kernel Experiences. In *Proceedings of the 24th ACM Symposium on Applied Computing* (SAC'2009, Honolulu, Hawaii, March 8-12, 2009), pp.177-184.

[10] *Mobile Web Best Practices 1.0*. World Wide Web Consortium Recommendation Document (July 29, 2008). Web link: *http://www.w3.org/TR/mobile-bp/*.

[11] Moroney, L., *Introducing Microsoft Silverlight 2.0* (2nd edition). Microsoft Press, 2008.

[12] Riggs, R., Taivalsaari, A., Van Peursem, J., Huopaniemi, J., Patel, M., Uotila, A., *Programming Wireless Devices with the Java™ 2 Platform, Micro Edition* (2nd Edition). Addison-Wesley (Java Series), 2003.

[13] Taivalsaari, A., Mikkonen, T., Ingalls, D., Palacz, K., Web Browser as an Application Platform. In *Proceedings of the 34th Euromicro Conference on Software Engineering and Advanced Applications* (SEAA'2008, Parma, Italy, September 3-5, 2008), IEEE Computer Society, pp.293-302.

[14] Tucker, D., Casario, M., De Weggheleire, K., Tretola, K., *Adobe AIR 1.5 Cookbook*. O'Reilly Media, 2008.