Sun Labs Lively Kernel
Lappeenranta Code Camp

Tommi Mikkonen
Tampere University of Technology
tommi.mikkonen@tut.fi

Antero Taivalsaari
Sun Microsystems Laboratories
antero.taivalsaari@sun.com

Background

- History of computing and software development is full of disruptive periods and paradigm shifts.
- The computing industry reinvents itself every 10-15 years.
- Examples of disruptive eras:
  - Minicomputers in the 1970s
  - Personal computers in the 1980s
  - Mobile software and Web 1.0 in the late 1990s

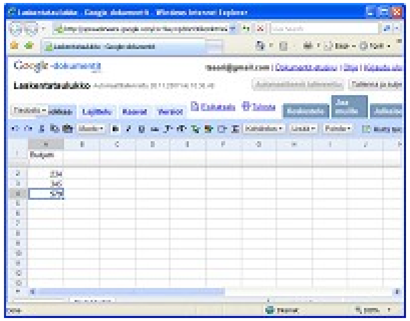
The Next Paradigm Shift!

- The widespread adoption of the World Wide Web is reshaping our world in various ways.
- Documents, photos, music, videos, news and various other artifacts and services have already started migrating to the Web.
- Many industries (e.g., publishing and entertainment) are currently undergoing dramatic transformations.
- The software industry is on the brink of a similar transformation, or a paradigm shift.

Evolution of the Web

1) Simple pages with text and static images only
(e.g., http://www.google.com)

2) Animated pages with plug-ins
(e.g., http://www.cadillac.com)



3) Rich Internet Applications
(e.g., docs.google.com)

What's Next?

Web Applications – Implications

- Web-based software will dramatically change the way people develop, deploy and use software.
- No more installations!
  - Applications will simply run off the Web.
- No more upgrades!
  - Always run the latest application version.
- Instant worldwide deployment!
  - No middlemen or distributors needed.
- No CPU dependencies, OS dependencies, ...
  - The Web is the Platform.

Unfortunately...

- The web browser was not designed for running real applications.
  - It was designed in the early 1990s for viewing documents, forms and other page-structured artifacts – *not* applications.
  - Programming capabilities on the web were an afterthought, not something inherent in the design of the browser.
- Various Rich Internet Application (RIA) technologies have been introduced recently to retrofit application execution capabilities into the web browser.

Best Known RIA Technologies

- At this point, the following Rich Internet Application development systems are best known:
  - Ajax
  - Ruby on Rails
  - Google Web Toolkit & Google Gears
  - JavaFX
  - Adobe AIR (Apollo)
  - Microsoft Silverlight

Landscape of RIA Technologies

Browser-based

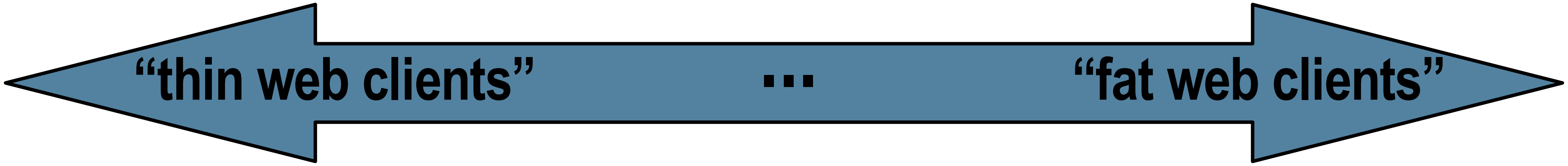
- Ajax
- Google Web Toolkit
- Sun Labs Lively Kernel

Plugin-based

- Flash & Flex
- (Java FX, AIR)
- (Microsoft Silverlight)

Custom runtime

- Java, Java FX
- Adobe AIR
- Silverlight



“thin web clients” ... “fat web clients”

- Run in a standard browser
- No plug-ins needed
- Platform-independent
- Browser-based UI

- Browser plug-in required
- Custom UI

- Custom execution engine required
- Runs outside the browser
- Custom/native UI

Technologies in the web browser serve as the lowest common denominator!

The Lowest Common Denominator

Technologies Supported by all the Web Browsers

- *HTML*. Widely established hypertext markup language for the creation of web pages.
- *CSS (Cascading Style Sheets)*. A stylesheet language that is used to describe the presentational aspects of a document. Allows stylistic aspects of a web page to be defined independently of its content.
- *DOM (Document Object Model).* Platform-independent way of representing a collection of objects that constitute a page in a web browser.
- *JavaScript*. Predominant scripting language; supported by all the commercial web browsers.
- *XMLHttpRequest*. An interface that allows a web application to download data asynchronously, without blocking the UI.

Comments on Web Technologies

- There is surprisingly little coherence between the different web application development systems.
  - In some ways, these systems have only one thing in common: they are all different.
- Some common themes:
  - Convergence towards *JavaScript* and asynchronous HTTP networking (XMLHttpRequest).
  - Many systems are *hybrid* combinations of existing technologies – HTML, DOM, CSS, JavaScript, PHP, XML, ...
  - Many of them are heavily dependent on tools.
  - Many of them are still prototypes, in different stages of development.

Why JavaScript?

- JavaScript is ubiquitous.
  - Supported by all the commercial web browsers.
  - “The programming language of the Internet”
- JavaScript has developer appeal.
  - Familiar to people with C, C++ or Java background.
- JavaScript is truly dynamic.
  - No more edit-compile-link-run-crash-debug cycles.
  - Applications can be created, deployed and modified without ever leaving the web browser.
- JavaScript has potential.
  - Momentum still growing. Performance will improve.

Pushing the Limits of the Web Browser:

The Sun Labs Lively Kernel Project

Three Assumptions

- The Web is *the* Application Platform
- The Web Browser is the Operating System (at least for end user applications)
- JavaScript is the *de facto* Programming Language of the Web

Sun Labs Lively Kernel

- The Lively Kernel is a web application development environment written entirely in JavaScript.
- Runs in a regular web browser with no installation or plug-ins whatsoever.
- Supports real applications on the Web, with rich user interface features and direct manipulation capabilities.
- Enables application development and deployment without installation or upgrades.
- Allows application development within the web browser.

The Lively Kernel in a Nutshell

Key components:

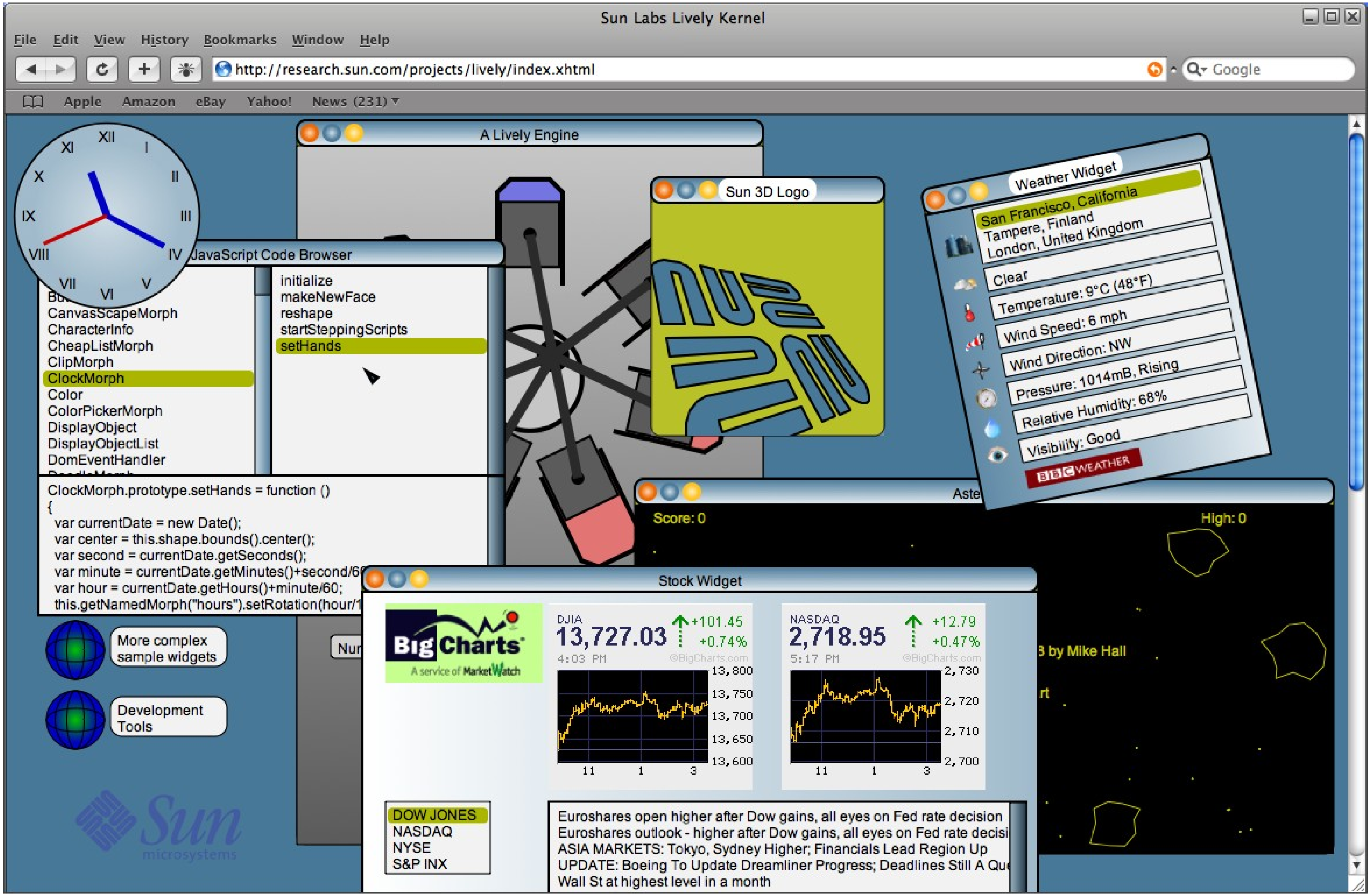
- JavaScript programming language
- Asynchronous HTTP networking
- Desktop-style graphics architecture with zooming
- Morphic application framework and widgets

Built on technologies that already exist
in the browser – no plug-ins required!

Morphic User Interface Framework

- The Lively Kernel is built around a user interface framework called *Morphic*.
- Morphic was originally designed for the *Self* system, and was later used also in the *Squeak* Smalltalk system.
- Every graphical object in the system is a *morph*.
- Morphs reside in a *world* – a visual container of objects that can be manipulated in various ways.
- Morphic provides exceptionally flexible mechanisms for object scaling, rotation, zooming, etc.

Demos!



Bu...
CanvasScapeMorph
CharacterInfo
CheapListMorph
ClipMorph
ClockMorph
Color
ColorPickerMorph
DisplayObject
DisplayObjectList
DomEventHandler

initialize
makeNewFace
reshape
startSteppingScripts
setHands

```
ClockMorph.prototype.setHands = function ()
{
 var currentDate = new Date();
 var center = this.shape.bounds().center();
 var second = currentDate.getSeconds();
 var minute = currentDate.getMinutes()+second/6
 var hour = currentDate.getHours()+minute/60;
 this.getNamedMorph("hours").setRotation(hour/1
```

San Francisco, California
Tampere, Finland
London, United Kingdom

Clear
Temperature: 9°C (48°F)
Wind Speed: 6 mph
Wind Direction: NW
Pressure: 1014mB, Rising
Relative Humidity: 68%
Visibility: Good

DJIA 13,727.03 +101.45 +0.74% 4:03 PM ©BigCharts.com

NASDAQ 2,718.95 +12.79 +0.47% 5:17 PM ©BigCharts.com

DOW JONES
NASDAQ
NYSE
S&P INX

Euroshares open higher after Dow gains, all eyes on Fed rate decision
Euroshares outlook - higher after Dow gains, all eyes on Fed rate decisi
ASIA MARKETS: Tokyo, Sydney Higher; Financials Lead Region Up
UPDATE: Boeing To Update Dreamliner Progress; Deadlines Still A Qu
Wall St at highest level in a month

How is the Lively Kernel Different?

- No plug-ins! All you need is the browser.
- No installation!
- No binaries!
- Everything written in JavaScript using a uniform set of APIs.
- Built-in IDE capabilities – applications can be developed using the Lively Kernel itself using nothing more than a web browser.
- In general, the system is fully interactive and “lively”

Where is the Lively Kernel Headed?

- The Lively Kernel was released to the public as an Open Source project in October 2007.
- Available under GPL license at:
  - http://research.sun.com/projects/lively
- Current research directions:
  - support for on-the-fly creation of web sites and mashups
  - better end-user programming / IDE capabilities
  - running the system on mobile devices
  - building more complete applications

Browser as a Platform: Experiences

Summary of Problem Areas

- During our project, we have discovered problems in various areas related to the use of the web browser as an application platform:

  1) Usability and user interface issues

  2) Networking and security issues

  3) Browser interoperability and compatibility issues

  4) Development style and testing issues

  5) Deployment issues

  6) Performance issues

  7) Software engineering issues

Usability and User Interface Issues

Highlights:

- Limited direct manipulation capabilities
- Poorly suited I/O model between JavaScript and the browser (via DOM)
- Poorly suited networking model between the client and the server
- “Legacy buttons” in the browser
- Poor support for well-known mechanisms such as cut/copy/paste, drag-and-drop, etc.

Networking and Security Issues

Highlights:

- “Same origin” networking policy restrictions
- Only a limited number of simultaneous network requests allowed
- No local storage support / no access to the local file system
- In general: The “one-size-fits-all” sandbox security model provides only limited access to host platform capabilities

Browser Compatibility Issues

Highlights:

- Incompatible DOM implementations
- Incompatible JavaScript implementations
- Incompatible graphics library implementations
- Disregard for official standards
- Lack of official standards (e.g., lack of advanced JavaScript libraries, no agreement on the future of the JavaScript language itself)
- Plug-in availability

Development Style and Testing Issues

Highlights:

- JavaScript is an extremely permissive, dynamic language -> incremental development and testing style required
- No static type checking
- Incompatible programs allowed -> code coverage testing is very important
- JavaScript APIs are still limited in various areas such as audio, storage, mobility, etc.

Deployment Issues

Highlights:

- It is not clear what constitutes a “release”
- Applications are online 24x7 – when is it safe to update them?
- “Perpetual beta syndrome”
- “Nano releases”

Performance Issues

- JavaScript virtual machines are still very slow
- Browser graphics libraries (e.g., SVG engines) are also slow
- Bindings between different components are slow
- When people start writing more serious web applications, performance issues will become more evident
- On the positive side:
  - There are a lot of opportunities to improve performance
  - Current JavaScript VMs are surprisingly reliable and almost impossible to crash

Software Engineering Issues

Highlights:

- Web development is still an *ad hoc* activity
  - ... Just like software development was until the 1970s and 1980s before rigorous software engineering principles were introduced.
- Web applications have reintroduced many problems that were eliminated from SW development years ago
  - Lack of modularity, use of global data structures, widespread use of side-effects, tangled control flow.

Modularity Problems on the Web

- Web sites and apps tend to be highly unmodular.
  - By default, everything in a web site is public.
  - No clean separation of the public features of a web site from its implementation details.
  - Information hiding mainly through obfuscation/obscurity.
  - No information hiding support in JavaScript (prior to v2.0)
- No widely established interface description mechanisms or languages available.
  - It is difficult to change the implementation details without affecting the public use of a site.
  - This is a serious problem especially in the development of *mashups* which relies on massive third party reuse.

Use of Global Data Structures and Side-Effects

- The web browser is built around the Document Object Model (DOM).
- The DOM is effectively a large, global data structure (attribute tree) that is shared between the browser and other components (e.g., JavaScript engine).
- The DOM is commonly manipulated by means of *side-effects*.
  - The application changes DOM attributes and the browser responds to changes at the next suitable point in time.
- Not only this mixes up procedural and declarative style, but it is also error-prone, inefficient and subject to various browser incompatibilities.

Control Flow Issues (Spaghetti Code)

- Control flow of web applications tends to be difficult to follow.
  - Different types of technologies (HTML, JavaScript, CSS, XML, PHP) mixed up freely.
  - Hard-coded references used liberally.
  - Obfuscation commonly used in lieu of information hiding.
- The problems are exacerbated by the fact that web applications cannot be (easily) checked statically.
  - Incomplete programs and broken references allowed.
  - No transitive closure of programs available statically.
  - No support for static verification or type checking.

Web Development vs. Conventional Software

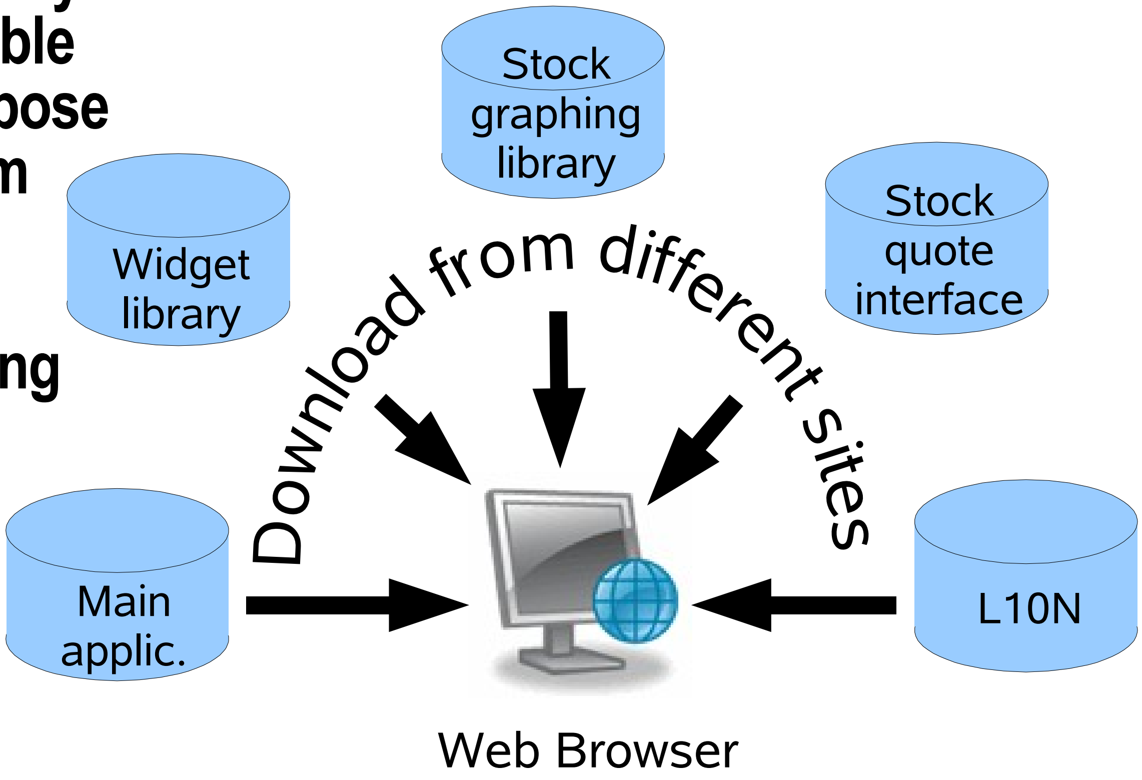
Web Development	Conventional SW Development
- Documents	- Applications
- Page / form oriented interaction	- Direct manipulation
- Managed graphics, static layout	- Directly drawn, dynamic graphics
- Instant worldwide deployment	- Conventional deployment
- Source code and text favored	- Binary representations favored
- Development based mostly on conventions and “folklore”	- Development based on established engineering principles
- Informal development practices	- More formal development
- Target environment not designed for applications	- Target environment specifically intended for applications
- Tool-driven development approach	- A wide variety of development approaches available

Future Vision: Software as a Mashup

In the future, software will likely be built by dynamically combining the best available components for each purpose by downloading them from anywhere on the Web.

No static linking; everything downloaded on demand.

Software development will be an inherently “social” activity between developers who do not necessarily know each other.



Mashup Development Tools

- Dapper (http://www.dapper.net/)
- Google Mashup Editor (http://code.google.com/gme/)
- IBM Mashup Center (http://www.ibm.com/software/info/mashup-center/)
- IBM Project Zero (http://www.projectzero.org/)
- Intel Mash Maker (http://mashmaker.intel.com/)
- LiquidApps (http://www.liquidappsworld.com/)
- Microsoft Popfly (http://www.popfly.com/)
- Mozilla Ubiquity (https://wiki.mozilla.org/Labs/Ubiquity)
- Open Mashups Studio (http://www.open-mashups.org/)
- Yahoo Pipes (http://pipes.yahoo.com/)

Conclusions

- Like it or not, the Web is increasingly the platform of choice for advanced software applications.
- Web-based applications have major benefits: no installation or upgrades needed, instant worldwide deployment without middlemen.
- Web-based applications will dramatically change the way people develop, deploy and use software -> paradigm shift!
- Since the Web was not designed for applications, there are still a lot of interesting problems to solve.
- The web browser must evolve to become a better environment for applications and mashups.

Thank You! Questions?

http://research.sun.com/projects/lively

lively@sun.com