Analyse of Interactive Graphics in HTML5 from the View of a Practical Application

STEFAN HUOVINE N OTTOSSON

Master’s Thesis in Media Technology (30 ECTS credits)
at the School of Media Technology
Royal Institute of Technology year 2011
Supervisor at CSC was Hannes Ebner
Examiner was Johan Stenberg

TRITA-CSC-E 2011:029
ISRN-KTH/CSC/E--11/029--SE
ISSN-1653-5715

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
SE-100 44 Stockholm, Sweden

URL: www.kth.se/csc
Analyse of interactive graphics in HTML5 from the view of a practical application

Abstract
HTML5 is a part of Web Applications 1.0. Just as its predecessors, it defines a set of elements, such as `<p>` for paragraphs, `<img>` for images, and now also new media types, including `<audio>` and `<video>`. These elements are used to mark entry points for the actual content that will be mediated to the user.

This report will focus on and compare the `<canvas>` and `<svg>` elements – which can be used for interactive graphics – in a specific web application.

Web standards are important because they allow for easy and free communication of ideas and applications on the web.

Analys av interaktiv grafik i HTML5 ur perspektiv från en praktisk applikation

Sammanfattning
HTML5 är en del av Web Applications 1.0. Precis som sin föregångare definierar den element såsom `<p>` för paragrafer, `<img>` (image) för bilder, och nu även andra medietyper, såsom `<audio>` (ljud) och `<video>`.

Denna rapport kommer att jämföra `<canvas>` och `<svg>`-elementen – som kan användas för interaktiv grafik – i en specifik webbapplikation.

Webstandarder är viktiga därför att de möjliggör enkel kommunikation av idéer och applikationer utan kostnad på webben.
**Table of contents**

Analyse of interactive graphics in HTML5 from the view of a practical application ........................................1

Analyse of interactive graphics in HTML5 from the view of a practical application ........................................2

Abstract ............................................................................................................................................................2

Analys av interaktiv grafik i HTML5 ur perspektiv från en praktisk applikation ............................................2

Sammanfattning ..............................................................................................................................................2

Introduction ..................................................................................................................................................1

The World Wide Web.................................................................................................................................1

  Short summary of its history ......................................................................................................................1

  Present state ..............................................................................................................................................1

Garion...........................................................................................................................................................2

  The company ............................................................................................................................................2

  The web application ..............................................................................................................................2

Problem ......................................................................................................................................................2

Aim ..............................................................................................................................................................3

Delimitations ..............................................................................................................................................3

  In general ................................................................................................................................................3

  Mock-up ..................................................................................................................................................4

  Prototype application ............................................................................................................................4

  Conformance experiment .....................................................................................................................4

  Performance experiment .......................................................................................................................4

Theory .........................................................................................................................................................6

About the literature study ..........................................................................................................................6

  Short summary of relevant web technologies .......................................................................................6

    Document Object Model ......................................................................................................................6

    Mark-up Languages ..............................................................................................................................6

  JavaScript, ECMAScript et cetera ..........................................................................................................8

  Short summary of relevant image technologies .....................................................................................8

  Graphics technologies ............................................................................................................................8

  Graphics standards ..................................................................................................................................9

  Animated graphics ..................................................................................................................................9

HTML5 .........................................................................................................................................................10

  Organs of standardisation ....................................................................................................................10

  The standardisation process ................................................................................................................11
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>The specification</td>
<td>12</td>
</tr>
<tr>
<td>The <code>&lt;svg&gt;</code> element and Scalable Vector Graphics</td>
<td>12</td>
</tr>
<tr>
<td>The <code>&lt;canvas&gt;</code> element and the Canvas 2D API</td>
<td>13</td>
</tr>
<tr>
<td>Web browsers</td>
<td>13</td>
</tr>
<tr>
<td>Web browser engines</td>
<td>14</td>
</tr>
<tr>
<td>Performing tests</td>
<td>15</td>
</tr>
<tr>
<td>Some HCI basics</td>
<td>15</td>
</tr>
<tr>
<td>Similar applications – Yahoo Pipes</td>
<td>15</td>
</tr>
<tr>
<td>Technical designs</td>
<td>15</td>
</tr>
<tr>
<td>Canvas</td>
<td>16</td>
</tr>
<tr>
<td>SVG</td>
<td>17</td>
</tr>
<tr>
<td>Method</td>
<td>18</td>
</tr>
<tr>
<td>Introduction to the methodology</td>
<td>18</td>
</tr>
<tr>
<td>Comprehensive description of the approach</td>
<td>18</td>
</tr>
<tr>
<td>Mock-up and prototype</td>
<td>19</td>
</tr>
<tr>
<td>Discussion</td>
<td>20</td>
</tr>
<tr>
<td>Conformance experiment</td>
<td>20</td>
</tr>
<tr>
<td>Reliability</td>
<td>21</td>
</tr>
<tr>
<td>Validity</td>
<td>21</td>
</tr>
<tr>
<td>Performance experiment</td>
<td>22</td>
</tr>
<tr>
<td>Reliability</td>
<td>23</td>
</tr>
<tr>
<td>Validity</td>
<td>23</td>
</tr>
<tr>
<td>Realisation</td>
<td>25</td>
</tr>
<tr>
<td>Mock-up and prototype</td>
<td>25</td>
</tr>
<tr>
<td>Material</td>
<td>25</td>
</tr>
<tr>
<td>Procedure</td>
<td>25</td>
</tr>
<tr>
<td>Conformance test</td>
<td>31</td>
</tr>
<tr>
<td>Material</td>
<td>31</td>
</tr>
<tr>
<td>Test subjects</td>
<td>31</td>
</tr>
<tr>
<td>Test Objects</td>
<td>31</td>
</tr>
<tr>
<td>Constants</td>
<td>32</td>
</tr>
<tr>
<td>Variables</td>
<td>33</td>
</tr>
<tr>
<td>Procedure</td>
<td>33</td>
</tr>
<tr>
<td>Performance test</td>
<td>34</td>
</tr>
</tbody>
</table>
Introduction
The World Wide Web is full of confusion, even about the standards that it relies on. In this report, an effort is made to use distinguished terms for specific matters.

From here on, the term HTML5 denotes the abstract hypertext mark-up language that can be written in both HTML and XHTML syntax, as described by its specification (Hickson & Hyatt, HTML5, A vocabulary and associated APIs for HTML and XHTML, W3C Working Draft 4 March 2010, 2010).

The term web application is often used to denote a web page that more or less resembles a computer program, where the actual application of a task is done by the user. While this is also true for the prototype developed during this degree project, the word application will also mean the crafting of a web page, done by a developer or a designer, using web standards such as HTML5, JavaScript and CSS. It is sometimes also used to describe the making of a standard, where HTML is an application of SGML, XHTML is an application of XML, and so on.

The World Wide Web

Short summary of its history
The world's first public web page went online in 1991 (CERN, 2008) and a later copy of it is archived by the World Wide Web Consortium (W3C). W3C was founded in 1994 and has since then developed many web-oriented standards such as the HTML, XHTML, SVG and PNG file formats.

http://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html

Figure 1: Link to the and archived version of the first public web page.

The first ever browser was called HyperMedia Browser/Editor (CERN, 2008). The name speaks for a browsing experience that is both interactive and not just limited to text, as in HyperText, which is a common term today.

But the evolution has been slow. In the beginning, the contents of web pages were static in the sense that page visitors could not manipulate it once it was downloaded. Interactivity was limited to clicking links, which would load other static web pages. Development soon began on technologies that would make web pages dynamic, without having to load a new page. This is sometimes called DHTML (Dynamic HTML). But DHTML is just a convenient word to describe a combination of technologies; it's not a standard or a file type. It's even not based on HTML, as there are alternatives to it. The meaning is simply the combination of content, presentation, the browser's internal model of the web page and client-side scripts to manipulate it.

Present state
Today we talk about Web Applications and Rich Internet Applications (RIA), which have naturally used proprietary technologies such as Flash, Java and Silverlight, because they offer new possibilities. But a paradigm shift could be happening now that open web standards are beginning to gain the lead that other technologies have had. For example audio, video and interactive graphic capabilities are now added to the mix, as mentioned earlier. The main topic for many web developers and designers today is HTML5, or its parent specification called Web Applications 1.0.
Web applications are in a sense easy to upgrade since they come from a central platform, unlike installed computer programs. On the other hand they are limited to the capabilities of the web browser that they are mediated through. Some of the biggest challenges today are graphics, client-side storage, performance and geo-positioning (Neuberg, 2009). It’s not yet trivial to make web applications look and behave like installed programs do.

**Gardio**

**The company**
Gardio AB is a Swedish company which was founded in 2010 and is working on a home/office security camera system with online services. By using motion-sensing cameras that generate images visible to the human eye (unlike many traditional systems), users have the opportunity to abort false alarms by examining these images. Communication about alarms and other events is done on the users’ preferred message channels.

**The web application**
The security system is highly customisable. Users will be able to define their own message flows by defining who will receive messages and on which channels, depending on the type of event and when it happens. Messages can also be chained with time delays. This task can be quite complex and all settings hard to overlook.

Therefore, Gardio is exploring a solution with a richer visualisation concept for their web interface. The prototype web application developed in this degree project serves both as a visual settings editor and a dynamic model over the message flow.

Another concern is that users must be able to use this web application without difficulty in any web browser of their choice. This project is set out to determine if and how this can be achieved using open web standards, most notably HTML5. The benefit of using open web standards is that they work directly in the browser without the need for additional software; neither for users, nor for developers. Convenience and low costs are universal selling points and a third aspect of using built-in browser capabilities is that it may also pass through restrictive software policies common to security-aware companies.

**Problem**
A very short definition of the problem would be:

*Based on relevant theory and specifications, how should the proposed idea be made into an HTML5 web application, and which practical limitations exist?*

The problem includes illustrating the abstract idea as an ideal mock-up, making the mock-up into a working prototype web application and defining its technical requirements. Conversely, it includes identifying the technical features and current implementation status of the technologies and standards that can be used to make this prototype.

In order to further elaborate the problem at hand, six questions of theoretical and empirical nature are posed:
Inledning

- What is HTML5, namely; what does its specification cover (in respect to the prototype) and where is its place in the ecosystem of web standards?
- What is the future potential and sustainability of HTML5; who is involved with the standard and how does the standardisation process work?
- Which technologies are in theory suitable for the prototype?
- To what degree do these technologies support the requirements for the prototype?
- To what extent is a solution based on HTML5 and the technologies required interoperable among web browsers today?
- How well would it perform?

The basic concepts of the prototype are rather general and the results of this study should be interesting to anyone who is making this kind of application in open web standards. The report does also outline contemporary requirements for in-line SVG, which is of interest to anyone planning to make a web application using <svg> elements directly inside an HTML or XHTML document.

Aim
The aim is to develop a prototype that works in today’s dominant web browsers and to create a solid foundation of knowledge to support a recommendation about how to proceed in the future with respect to web standards. In other words, a more general understanding of HTML5 should be conceived and the aspects that will be researched are connected to the prototype. If the research questions are answered, they should witness for or against the plausibility of deploying this web application using open web standards; today or in the foreseeable future.

This report should not be seen as a comparison between open and proprietary standards and technologies, but rather as guidance for using technologies in a way that has not been described in previous versions of open web standards.

It is important to note that the research relies on a standard that is still evolving. Therefore, current implementations of the standard (as of March – June, 2010) are expected to vary among web browsers. The aim here is to determine a common ground – if one exists today – and to describe how far standardisation has come.

Delimitations
In general
This report is written as a part of a degree project in Media Technology. It has some references to Human-Computer Interaction (HCI) and Computer Science. The essence lies in that the web interface can be correctly mediated via web browsers to end users. Since time is limited and a working prototype will be built from the ground, no attempt is made to create and evaluate multiple technical or interaction designs. Instead, the simplest possible application is developed according to internal discussions. The result is then described in detail in this report.

HTML5 is in itself a vast standard which defines how a handful of media technologies should be referenced from within a web document. The HTML5 standard lives within an ecosystem of adjacent web standards and technologies. Many of these standards – including HTML5 – are also still in development. When combined, this makes it more or less impossible to make a controlled evaluation
Inledning

of the whole HTML5 standard and its impact on web applications. Such evaluation transpires on a
daily basis by everyone who develops or uses web applications.

There is also an infinite amount of ideas to try and apply in HTML5, and just as many ideas may rise
again from using it. There is no way to make a universal judgment whether the Web Applications 1.0
family of standards is adequate for making all possible web applications.

This report will only describe portions of standards relevant to the specific web application. The basic
visual characteristics of the web application will be made as simple as possible and will be similar if
not identical to other practical examples. Embracing simplicity, looking at other examples and
building upon basic theorems is as far as this project goes in terms of Human-Computer Interaction.

The whole problem is designed to be descriptive and will for example deal with how performance
differs in different web browsers. The question of why performances differ is analytical and often
requires more time to answer than there is in a degree project (Stensmo, 2002). This is also where
the limit is drawn toward Computer Science.

Mock-up
A project themed on Human-Computer Interaction would certainly be even more focused on the
initial mock-up, perhaps resulting with several different propositions and actual user tests performed
on them. For this project, the mock-up has been created with basic HCI theorems in mind, but no
further analysis of the basic design has been made. The actual content of the web application is
however presented as a part of the project and it does affect the final complexity of the application.

Prototype application
The web application is intended for use in home or office environments by customers who have a
user account on Gardio’s web site. No attempt has been made to find eventual differences in
software use policies or habits between home and office environments. The primary target group is
Swedish residents, but statistics are also collected from world-wide and screenshots of the prototype
are translated into English for this report.

Just as with the mock-up, the prototype is developed internally, but the process and its result are
described in detail.

Conformance experiment
Several conformance tests are made in order to find a common ground, mainly in order to make
further experiments as objective as possible, but also in order to make a general description of the
state of current implementations. The most used web browsers according to world-wide statistics
will be subject to the experiment and web browsers that have technologies in common (for example
the layout engine) will still be treated as individual test subjects.

Performance experiment
Only basic shapes with no filter effects or other graphical effects are used for the performance
experiment. Graphical effects would require additional testing for browser interoperability on a par
with experiments already made for HTML5.
As with the rest of the report, results are descriptive and not analytical. For example, no attempt is made to distinguish how performance is affected by the JavaScript engine and the web browser engine respectively.

The experiment is done with technologies directly related to HTML5 and not with third party technologies.
**Theory**

**About the literature study**

*The theory chapter is based on three kinds of literature: the specifications by WHATWG and W3C, journal articles on previous standards, and public discussions on relevant topics from past and recent.*

The HTML5 standard is still not finished. Here follows an excerpt:

> “Implementors should be aware that this specification is not stable. Implementors who are not taking part in the discussions are likely to find the specification changing out from under them in incompatible ways. Vendors interested in implementing this specification before it eventually reaches the Candidate Recommendation stage should join the aforementioned mailing lists and take part in the discussions.”

(Hickson & Hyatt, HTML5, A vocabulary and associated APIs for HTML and XHTML, W3C Working Draft 4 March 2010, 2010)

The following table will be used when discussing the standards:

<table>
<thead>
<tr>
<th></th>
<th>Compatible</th>
<th>Incompatible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuation</td>
<td>Continuity</td>
<td>Discontinuity</td>
</tr>
<tr>
<td>Concurrent</td>
<td>Cooperation</td>
<td>Competition</td>
</tr>
</tbody>
</table>

*Table 1: Simplification from Egyedi and Loeffen (2001).*

**Short summary of relevant web technologies**

**Document Object Model**

The Document Object Model (DOM) is the browser’s internal representation of a web document. The structure of the web document, for example written in HTML, is interpreted and put inside the DOM. Every new version of a mark-up language thus also requires an update of the DOM.

Web applications that want to use and manipulate a web document in a browser do so through the DOM Application Programming Interface (API), which defines methods for that purpose. Many technologies are standardised, but in the end, it is the browser vendors that are in charge of the implementation of the DOM and its API (Guisset, 2005).

**Mark-up Languages**

Mark-up languages are languages designed to be interpretable by both humans and computers, by using common words to tag content and arrange it in a structured way. Tags make up elements, and most elements can contain more elements of specific kinds. Defining these relationships is one of the purposes with mark-up standards.

One goal for standardising mark-up languages used on the web is to aid accessibility and searchability. Web browsers and other programs can only recognise text if it is embedded in mark-up; not if it is embedded in other data types (Russel, 2008). For example, Google Translate cannot translate the text on a road sign in a photograph.
Text that is written inside mark-up elements is not stored in the HTML or XML DOM, but they are stored in separate nodes (W3Schools).

Web browsers interpret the element whose names it can recognise. HTML elements that are not recognised will be ignored, while unrecognised XML elements will cause parsing to halt.

HTML is an application of SGML, which means that the rules for making an SGML document were applied when creating all standardised HTML elements and their respective relationships and attributes.

On the contrary, XML is a simplified version of SGML with only some of its rules. This means that XML can be applied in order to create new standards. And there are many such applications of XML, perhaps because its smaller set of rules is more manageable for authors. Some examples are MathML, SVG and XHTML.

As the name suggests, XHTML is similar to but not quite the same as HTML. In essence, the elements and attributes are borrowed from HTML and the structure is much the same. But the rules that are a part of XML are not all the same as the rules applied in SGML when creating HTML. In other words, different (stricter) rules were used to create essentially the same elements; hence XHTML is a reformulation of HTML as an application of XML.

HTML5 is called an abstract language and is a new mark-up language that can be expressed both in HTML and XML syntax. The name implies that it is a succeeding version of HTML 4.01, but it is in fact also replacing XHTML 1.1 and DOM Level 2 HTML (a specification for the DOM) (van Kesteren, 2010).

To further mark the differences between HTML and XML, the following table has been interpreted from Wood (1999):

<table>
<thead>
<tr>
<th></th>
<th>HTML</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predefined tags</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>
Table 2: Differences between HTML and XML

<table>
<thead>
<tr>
<th>Feature</th>
<th>HTML</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tags can be omitted</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Case-sensitive</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Unicode mandatory</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>SGML-features</td>
<td>some</td>
<td>many</td>
</tr>
</tbody>
</table>

**JavaScript, ECMAScript et cetera**

JavaScript is a vastly used programming language and probably the most popular script language for web pages (Guisset, 2005). It is known under many names; the formal standard name is ECMAScript and proprietary variants include JavaScript, LiveScript and JScript (Crockford). It has its own specified functions and should not be confused with the DOM API (Guisset, 2005).

JavaScripts are run in the browser (that is, client-side only) and can be used to manipulate currently loaded web pages. Web browsers do usually not display errors in the code to the user unless it has a debug mode which is activated. If an error occurs, the rest of the page will still be interpreted if possible.

**Short summary of relevant image technologies**

**Graphics technologies**

Two-dimensional images on computers are commonly rendered with three basic technologies: raster graphics, vector graphics and object graphics.

Raster graphics are expressed with pixels – picture elements (Johansson, Lundberg, & Ryberg, 2006). Each pixel has a bit depth for each colour. If a picture is not compressed with a lossy algorithm, these pixel values make up the final detail in the image.

Vector graphics and object graphics are not inherently represented by pixels, but with mathematical concepts that can be redrawn in any scale. Vector graphics predated object graphics and used several vectors that together would look like smooth, continuous lines (Johansson, Lundberg, & Ryberg, 2006). Object graphics on the other hand can define more complex shapes, for example mathematical circles.

![Figure 3: Example of object graphics.](image-url)
At some point, a computer image rendered on-screen will inevitably be expressed with pixels. The difference between raster graphics and the other two technologies is that rasters are already expressed in pixels while the other are translated into pixels. When changing the resolution, it will be noticeable that rasters are samples while vectors and objects can be redrawn with no loss of information. Then it is the media that determines the quality, not the image data (Johansson, Lundberg, & Ryberg, 2006).

Vector graphics is a common term (not the least on the web) that is quite misused since it usually refers to what is actually object graphics (Johansson, Lundberg, & Ryberg, 2006). Object graphics will be the term used from here on in this report, even though for example SVG stands for Scalable Vector Graphics.

**Figure 4:** The difference of raster images (left path) and object images (right path) at different zoom levels. The top-right box is displayed at 100 per cent normal size.

**Figure 5:** Vector images consist of several vectors with start- and end-points in a coordinate system. Object images allow for more advanced objects than vectors; in this example a circle is defined by the coordinates of its centre-point and a radius.

**Graphics standards**

W3C offers graphics specifications for different types of situations and claims that Canvas and SVG are essential for web applications (W3C). Four standards are maintained by W3C.

PNG stands for Portable Network Graphics and is an image file type for lossless compressed raster images. It is "designed for the Web, with streaming and progressive rendering capabilities" (W3C).
Scalable Vector Graphics (SVG) is described as “HTML for graphics” (W3C). It is in fact a mark-up language based on XML and it also has its own DOM and API. SVG can refer to other media such as audio and video. One area of use is interactive charts and other data visualisations. Many features of SVG (W3C) are common ingredients of object graphics (Johansson, Lundberg, & Ryberg, 2006).

CSS stands for Cascading Style Sheets and can be used to change the presentation (for example colour and layout) of web page components (such as HTML, XHTML and SVG elements) (W3C). This is done by referencing element names, classes or id’s.

WebCGM can describe images composed of both vectors and rasters and has similar features as SVG (W3C). It is mainly used to visualise works of engineering.

Canvas is in itself not a standard for defining graphics; there are different API’s for different kinds of images. Common with all Canvas graphics is that they are rasters. W3C states that the Canvas 2D API "uses vector-based programmatic methods to create shapes, gradients, and other graphical effects, and because it has no DOM, it can perform very quickly." (W3C).

**Animated graphics**

When an image is animated interactively, for example with scripts on a web page, other terms are usually used to describe the technical difference between object and raster images. Object graphics have their objects retained in a model and are therefore sometimes called retained-mode graphics. Raster images are not stored in a model first but are instead drawn directly when commands are made. They are therefore called immediate or command-mode graphics.

According to a Google presentation (Neuberg, 2009), SVG is a high-level retained-mode API which aids interaction with objects. Canvas on the other hand is a low-level immediate-mode API that does not support mouse interaction inherently, but is on the other a better alternative for intense animations (Neuberg, 2009).

**HTML5**

**Organs of standardisation**

There are two main organisations working on HTML5: The World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG). The latter was formed in 2004 as its cofounders reacted against W3C’s work (WHATWG, 2010). The most active work on the specification seems to be done in the WHATWG. Both organisations have the current specification on their web sites, and they share the same editor.

WHATWG currently has nine members of which one acts as editor. Anyone can send comments and proposals to the editor or the public mailing lists, and the other members have oversight of the process. The standardisation process takes actual web practice in account and is tightly connected to the browser market and the web community. There are de-facto processes for proposing the addition and deletion of things in the specification. It basically means to get attention from those who implement the standard and authors who use it, and then convince the editor (WHATWG, 2010). New features are sometimes not added directly to the specification before they are actually implemented (Hickson, [whatwg] Resolutions meta tag proposal, 2010). Other features may be impossible to remove since they are already implemented and actively used (Hickson, [whatwg] postMessage's target origin argument can be a full URL in some implementations, 2010). There are
no voting or consensus mechanisms. The editor has however made active efforts in looking for how things are done on reality, both by implementers and authors (Hickson, [whatwg] Supporting MathML and SVG in text/html, and related topics, 2008).

At W3C the total number of HTML Working Group participants and represented companies is growing. There are currently over 400 participants. About 250 are called Invited Experts (W3C, 2010). Seven of WHATWG’s members are also members of the W3C HTML WG.

All members of W3C HTML WG or WHATWG do not have to support the specification for it to be published (Hickson & Hyatt, HTML5, A vocabulary and associated APIs for HTML and XHTML, W3C Working Draft 4 March 2010, 2010).

**The standardisation process**

There are four basic layers of involvement in the standardisation process:

**Specificators – Implementers – Authors – Users**

Specificators take all feedback in account. The idea at WHATWG has from start been to make a standard that reflects how HTML is implemented and used in reality (van Kesteren, 2010). The specification states that it is backwards-compatible, but that it separates old features so that authors do not use them again. Features are thus never deprecated. The HTML5 specification can be read by implementers and authors alike.

Implementers have in their interest to realise authors’ and users’ expectations. Implementers of web standards are for example web browser vendors. They may have financial strategies that differ in terms of approach toward open standards. Implementers are often deeply involved and take a leading role in the act of specifying the standards that they use, be it proprietary or not. In the case with HTML5, this is done collaboratively by most of the dominant vendors. Microsoft is due to judicial issues however only officially involved with the W3C edition, even though both editions resemble the same content and have the same editor.

Authors (web developers and designers) want to provide a good experience across browsers. Authors therefore must know how the implementations work in practice and what the users demand. Specifications are useful guidance, and some authors push the use of standardised solutions (which is indeed also done in this project). There are many reasons for authors to want standards to follow, but in practice they have to play by the rules given on the market.

Users want a good browsing experience. No direct connection between common end users and the others mentioned has been found. For example, specificators seem to only collect information actively by looking at authors’ work, and there is no mailing list specialised for end users.

It is commonly said, for example in a Google presentation (Neuberg, 2009), that Canvas and in-line SVG are well-supported by the most used browsers except Internet Explorer. The study suggests that they have at least gone through the roughest part of the HTML5 specification process. They are somehow implemented in most browsers today and will be available in all of the five most popular browsers in the future. All in all, it seems very unlikely that an application of either technology on a web page today will be any less compatible in the future.
The specification
The HTML5 specification will be finished, that is recommended by W3C, when two so-called user agents (for example web browsers) implement it fully (van Kesteren, 2010). This approach is a bit hard to comprehend since specification features and their implementations grow in symbiosis; logically it requires that the specification is always one step before actual implementations, and not the other way around.

Also, since the specification is constantly evolving with new features, a better question would perhaps be when the amount of features is considered enough. One goal with HTML5 is to make it an unversioned specification, and in that case there is according to the editor a possibility that there may an HTML6 specification (Hickson, [whatwg] HTML6 Doctype, 2010). Theoretically there would then be no end to its features.

From that point of view it is more interesting to know when exactly features are ready to use. This is not always easy to tell (and it will therefore be dealt with in the conformance experiment), but the general idea is to split major features into different specifications for easier overview.

All things considered, the top priority is to have interoperability between web browsers, and that is something that seems to be handled well.

The <svg> element and Scalable Vector Graphics
The <svg> element is new to HTML and XHTML, but the technology behind this graphical element has existed for a long time (recommended since January of 2003) in many browsers and is already defined by the SVG specification.

SVG is more than a simple element; it’s another mark-up language nested inside HTML and XHTML. Conversely, SVG allows for so called foreign objects such as HTML to be nested inside it, if it’s supported by the web browser. This means that complex applications are now possible.

The fact that SVG is related to XHTML and HTML introduces both positive and negative factors. It is an application of XML which makes it relatively easy to interpret by both humans and machines and it can take advantage of styling and scripts in a similar way. The negative part is that it was never made to be compatible with HTML or XHTML. For example, the title element exists in both SVG and HTML, but is used differently. And since SVG is based on XML, it relies on a different, stricter, set of rules than HTML.

Having in-line SVG that works in both HTML and XHTML context lowers the bar and makes it possible to use object graphics without having external files (Hickson, [whatwg] Supporting MathML and SVG in text/html, and related topics, 2008).

Reports regarding support for in-line SVG seem to be contradictory and simple examples from the HTML5 specification do not render at all in current web browsers. This is probably caused by the fact that many browsers do support SVG per se, but have not yet adapted the way it is done in-line in HTML5, or at least not all ways it can be adapted. It is also very hard to prove that all possible use cases do work; they would all have to be tested. It then becomes a question of verification or falsification.
The <canvas> element and the Canvas 2D API

The <canvas> element is a new feature in the HTML5 specification. It basically allows for a scriptable image element (Neuberg, 2009), an image that can be overwritten using special API’s. In practice, a Canvas image will behave like a normal image when zooming and it is inherently not possible to interact with parts of the image, since they technically are not individual objects.

The content model of the canvas element is embedded content, which means its contents are somehow defined by an external source. Namely, Canvas images have a context in which images are drawn. One of these contexts was originally a part of the HTML5 specification, namely the context currently referred to as the Canvas 2D API. This API was moved to its own specification in early 2010 (Smith, 2010).

Web browsers

There are many server-side tools for generating statistics of which browsers page visitors use. Some of these statistics are publicly available. Since there are many factors to deal with, one should not trust only one source, but compare many sources in order to get a sound overview. The following diagram shows a comparison of statistics available at the start of the research project:

![Browser stats February 2010](Figure 6)

All sources agree that there are five dominant web browser families on the market, in order: Internet Explorer, Firefox, Chrome, Safari and Opera. Opera has a market share of about one per cent in most surveys, and that seems to be a commonly accepted threshold, although very subjective.

The above diagram shows measurements made both worldwide and for Sweden. Another aspect that distinguishes web browser statistics is that they can either be based on a single site or multiple sites. Statistics for single sites are arguably more often biased than those of a wide span of sites. The following table divides the seven measurements on a grid with four possible categories:
Teori

<table>
<thead>
<tr>
<th></th>
<th>Site-specific</th>
<th>Inter-site</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>W3CSchools</td>
<td>Clicky, NetApplications,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>StatCounter (global),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>W3Counter</td>
</tr>
<tr>
<td>National</td>
<td>-</td>
<td>SIS, StatCounter (Sweden)</td>
</tr>
</tbody>
</table>

Table 3: Dividing statistical sources.

W3Schools is site-specific and arguably has standards-aware visitors due to the nature of the sites content. Internet Explorer is actually surpassed by Firefox on this site and detailed version statistics show as of this time officially unreleased versions of Firefox (4.0) and Chrome (6.0). If Internet Explorer 9 preview had been browsable (it has no address bar, only a dialogue box that has to be opened manually every time a new URL is to be entered) it would certainly have had showed up earlier in these statistics as well.

Statistics for Sweden seem to be fairly in line with the world in general, but then again the reliability of these numbers is rather low.

At the end of the research period, another round of statistics was collected. This time they are also compared to statistics from one year back, if available:

![Browser statistics chart]

Table 4: Browser stats for July of 2010. *Browser stats for July of 2009, for comparison.

**Web browser engines**

Web browsers consist of several important components, for example a web browser engine and a JavaScript engine. The web browser engine is responsible for layout and rendering of the document while the JavaScript engine executes JavaScript code.

There are actually only four web browser engines among the top five browsers; Chrome and Safari share the same one. This means that web documents can be expected to look identical in these two
browsers. No browser is currently sharing the same JavaScript engine, which means that JavaScript can behave differently and execute with different efficiency in all browsers. In the scope of this study, web browsers will be recognised as distinct subjects regardless of composition.

**Performing tests**
Testing browser implementations is usually done automatically with scripts. One very basic test is to first add a new element using a script and then see if it exists in the DOM. This can easily be done by exposing its Boolean value; true if it exists, false if it does not. When testing embedded content such as graphics, videos or sound, tests should go a bit further. For example, one could check that the Canvas 2D API or a particular sound codec is implemented – this can also be done in similar fashion. But the final test is of course to see that the graphics are actually rendered and that videos and sounds can be played correctly. In small experiments this can be done manually.

**Some HCI basics**
When activating and deactivating the alarm, the tempo will be high, but when constructing/programming the alert rules the tempo will probably be low. The prototype has been made with this in mind.

... 

**Similar applications – Yahoo Pipes**
Yahoo Pipes is used for making mash-ups of content from different web pages, such as YouTube and Google Maps. Since it deals with content of many different kinds, it is quite complex and has features that are very general, such as AND-gates.

... 

**Technical designs**
Depending on which technology to use, there are four essential ways to build the interface using HTML5 elements. The following pictures will show how. The dashed border is the border of the web page, while solid rectangles represent the border of elements. The so called viewport can be smaller than the actual graphics, which means that the graphics will be clipped. This is not illustrated but has to be considered in some cases.
**Canvas**

Figure 7: Using a single `<canvas>` element and drawing multiple paths in it. This is what the performance experiment uses for Canvas animation. Paths can have individual colour and other attributes when they are drawn but ultimately they will belong to the same raster image with no distinction possible. In other words the colours of the paths cannot be changed.

Figure 8: Making one `<canvas>` element for each path (this is how Yahoo Pipes works). This means that the elements have to be moved around and that they have to be big enough to encompass the paths.
Figure 9: Using one `<svg>` root element with several `<path>` elements. This is what the performance experiment uses for SVG animation. The coloured borders do not really represent the borders of the `<path>` elements since they do not have a viewport of their own. It is just to represent that they are individual elements. The `<path>` elements are visible throughout all of the `<svg>` element (black border).

Figure 10: Using several `<svg>` root elements with one `<path>` in each. Again, `<path>` elements have no border but are visible inside all of the `<svg>` element.
Method

This chapter acts as a bridge between theoretical studies and empirical attempts. It describes the methodology and intentions for what has been done. The next section will describe how the methodology actually worked out in practice.

Introduction to the methodology

In the very simplest wording, the project is about developing and analysing an HTML5 web application, from mock-up to prototype stage. Clearly, the research needs to be designed as some sort of attempt under controlled circumstances (Stensmo, 2002, s. 25).

The literature study is also a crucial part, not only before, but during the entire research period. It gives several important pieces towards answering the research question at hand. Major points of interests are that there are two technologies that can be used to make the prototype, what the browser market looks like and how the HTML5 specification process is working out.

However, the literature study leaves some areas with unsatisfactory answers. In order to have a better foundation for decision-making it is also interesting to know more about how the technologies actually perform during interaction and having as much detail as possible about how a working in-line SVG-solution should be crafted.

Comprehensive description of the approach

For research attempts, there are two general designs: experiments, which give general knowledge, and action research, which give ideographical, or specific, knowledge. Ideographical studies are characterised by having much data collected for a narrow subject, over long time and with different methods. Conversely, general research consists of a small amount of questions posed to a large amount of subjects (Stensmo, 2002, s. 29).

Since a new web application is being made, the general design of this project starts out as action research. Unlike an experiment, action research is much more abstract and does in fact not comprise a given methodology (Bell, 2009, s. 18). The approach can be used whenever specific knowledge is needed for a specific problem in a specific situation (Cohen & Manion, 1994:194 in Bell, 2009:18), and it means that the researcher is taking an active part in the practice at hand, together with experienced co-workers (Stensmo, 2002, s. 25). New insights that help define the problem and its solution are acquired during this process.

In this case, the intrinsic complexity and evolution of the web is a significant factor, and two more formal research methods are constructed in order to acquire a more general understanding. Namely, experiments are to be made on dominant web browsers’ standards conformance and animation performance.

For the experiments, there are some key factors to keep in mind. Performance should be measured in all browsers that support the technology, but it is not known if all web browsers support the exact same implementation of the performance test, which, in strict academic terms, affects the test in an unknown and subjective way. On the other hand one could construct a universal test for all web browsers that may not work in all of them. A third way is to find a non-standardised way that works in all browsers.
This project is standard-centric and so all standard compliant ways are to be sought. In case there is no solution that works in all browsers, the application that is to be used in the performance test depends on the priority of web browsers according to statistics collected.

The method for this project consists of four main steps to solve:

1. How can the web application be designed?
   - Creation of an ideal mock-up.
2. Which are the key functions?
   - Prototype development.
3. How can in-line SVG applications be made today?
   - Conformance experiment.
4. How do technologies compare in performance?
   - Performance experiment.

Each step has impact on the next. This means that each step can be seen as a pre-study for the next. The mock-up in particular is done as a smaller pre-study to get the project moving, while the other steps are given more time. Furthermore, the mock-up and prototype are more or less considered as a whole since they are developed internally and the composite outcome is what is being examined on a more theoretical level. The experiments are purely academic and described in detail from start to finish, each in separate sections.

When it comes to making internal decisions, it is generally the author who comes up with ideas that are sometimes implemented before they are informally brought up for discussion and eventual approval.

**Mock-up and prototype**

The development process is done in tandem with the literature study. The idea is that the activities will benefit from each other. By developing a web application it will be easier to understand what is written about the research topic and dive into the web developer community. Developing the prototype actively, having internal discussions about its interaction principles and technical features and exploring the HTML5 community are the central pillars of the action research. A shallow analysis of Yahoo! Pipes and other general graph applications is also to be made. Yahoo Pipes is after all a web based tool and should provide inspiring ideas for the prototype.

The boundary between investigation, development and research is not always sharp. This seems to be especially true for action research. In order to clarify what is to be done, the following exhortations are kept in mind:

- Describe all relevant features in as much detail as possible and motivate everything.
- Distinguish own and others’ creations, where ideas came from and what similarities and dissimilarities exist.
- Describe how conclusions are made.

A general rule for the development is to make the artefact as simple and general as possible in terms of form and functionality. This serves two goals: First of all, the purpose of the web application is to make users be able to perform a relatively complex programming task of their security system, which means that it must be easy to use and understand. Second, technical key functionalities should be
Metod

easy to derive and describe when the prototype is done, so that further testing and analysis can be made by implementing the same functionality in a set of different technologies.

After development is done, the result is analysed by structuring up the features that it uses and explaining why some features were eventually not implemented and how the prototype can be further developed.

Discussion

Since action research only consists of guidelines, it is difficult to measure its reliability and validity. The problem starts out as very composite and the goal is also somewhat diffuse. The abstract way of describing action research, as interpreted by the author, is to unwind the problem, understand its components and approach them, this time using stricter methods. This way, the goal will become more tangible as work proceeds and it will then be possible to have a more concrete discussion about it.

Following are some criteria that will help ensure that the project is moving in the right direction:

- The solution should work in the previously defined web browsers. This is tested and further described in a separate experiment.
- It should use web standards in order to be sustainable. This would also be quite easy to test in a reliable way if it were not for the unfinished status of the specification. The primary way to ensure this is to follow the standardisation process closely via public mailing lists. There are also validators that can validate the mark-up, but they are of course also not finished and are known to have errors.
- Except being usable, the web application should be efficient to run in a web browser. This is simulated in an experiment.

General graph applications can be used to compare the prototype superficially. It can also be compared briefly to Yahoo Pipes in terms of technological application and design once it is done. But the intent is as mentioned not to use Yahoo’s JavaScript libraries, code from Yahoo Pipes or its structural code design.

Another concern is that the prototype will probably change in unknown ways before being put to actual use. The way to approach this is to follow the most general principles and only add features that are absolutely necessary.

The prototype’s validity lies somewhere between the correct usage of the HTML5 specification (if it can be used) and the level of fulfilment as compared to what the final product should accomplish. This may be hard to realise since both of these benchmarks are variable. The only way to aid validity is, again, to make the solution as simple as possible so that it at least easy to validate.

Conformance experiment

The purpose of open web standards is to have interoperability between all major web browsers without depending on plug-ins or any additional software. Finding a solution based on HTML5 that actually works in practice is therefore implied in this research project.

Online articles often describe SVG compliance either in percentage or in more or less absolute terms. There are also test suites with myriads of use cases. They are not designed to give truly general
results, as they only deal with specific examples, although many. This is also more of an a posteriori method.

After empirical and theoretical research, it is clear that an experiment with a new angle on the situation should be constructed; an experiment that is designed to give more general information about the status of current implementation and has more character of a priori research. Since it is built on web standards, it should also be possible to reuse this experiment in the future in order to track changes.

The test object is built on an example for the HTML5 specification. While examples are non-normative, meaning that they do not show how one is supposed to write a web document, they should of course be valid. Specifications are then used to find all conceivable variables there might be in the code that can affect the result. The hypothesis is that at least one of the variables will affect how well some browsers interpret the document (the result should always be the same if all browsers were specification-conformant).

The experiment is observed manually by the author and what is checked is 1) an SVG graphic, 2) additional in-line text and 3) the document title. The actual variables are described in the Realisation section. The result is then represented by tables where combinations of variables giving positive result to all the three indicators are marked.

The aim is not to once and for all verify which browsers are HTML5-conforming with regard to in-line SVG. Verification can and is, even in this case, only possible for each specific setup; another example code could in theory give different results. The point is also not to prove that these variations are the cause of inconsistent reports found in the literature study. The aim is simply to point out how exactly in-line SVG can be accomplished in practice among today’s available browsers.

Test results are crucial to the next experiment, which preferably uses a common code base that works in as many browsers as possible. Test results are however not verified with the prototype.

All current web browsers and available next-generation previews among the dominant browser families are test subjects. Internet Explorer 8 is to be present in the result for completeness despite known lack of support.

**Reliability**
Verification is relatively easy for this experiment, since one only has to check for an SVG graphic, in this case a large green filled circle, along with a text string and a document title that matches expected values.

The test does not take in account that the circle might be rendered slightly differently in the different browsers, for example if it has different radius. Content placement is also not considered.

The experiment can be repeated again in the future since it is built using web standards. The problem is rather to re-test it in old versions of browsers, to make sure that the test results are accurate, since web browsers are so frequently updated.

**Validity**
The main question is if the code itself is valid HTML5. The basic source code is taken from the specification and alterations are made in co-ordination with the same.
Another question is if the tested code is representative for the purpose of the study. It does not deal with all SVG elements but only with one (the <circle> element, which is not used in the prototype), and the result also depends on a problem with namespace collisions that is not to be expected for Gardio’s web application (namely with the <title> element). Validity for this experiment depends on the idea that if any SVG element can be proven as implemented, then all SVG elements are implemented in that web browser. It also says that if the namespace collision is not accounted for, then it is not a correct implementation of in-line SVG.

**Performance experiment**

It has been shown that Canvas and SVG graphics have different spatial qualities. The question is if there are significant dissimilarities when it comes to temporal change, that is, animation. The literature study suggests that animated graphics applied in Canvas is inherently faster than graphics applied in SVG. This is part of the hypothesis for the performance experiment. Other than that, it is likely that performance will vary with the browsers and perhaps also with operating system.

The primary aim is to find out which of the two technologies seems to be the most efficient. This will then be summed up in the recommendation. The secondary goal is to find out which browser seems to be the least effective in rendering the faster technology. This is important to know when testing real-life situations.

In addition to that, other general characteristics of the performance can hopefully be extracted by looking at the charts.

The experiment uses the most web browsers with the highest market share as test subjects, and it is also run on the two most popular operating systems from the Windows and Mac OS families. Test machines are however not representative for any particular target audience. Test objects are the very same paths (algorithms) as in the prototype, but control points are calculated at random.

The result is printed on the same web page as the test and will then be interpreted in a diagram with “number of objects” as the X-axis and “updates per second” as the Y-axis. The test will use $n$ to the power of 2 paths, starting at $n = 1$ and ending at a value that will be decided in a pre-test so that it is clearly visible how the number of paths affect performance. The number of seconds to test will also be a compromise between how long the tests will take on the test computers and what time limits give accurate results.

Results do in reality depend on both the layout engines and the JavaScript engines of the subject web browsers, but it is impossible to make any distinction in this experiment. Therefore test subjects are only labelled after the browser family and version number.

It is also important to point out that this is not a performance test on raster graphics versus object graphics per se, because implementations may vary in browsers. The actual involvement of a web browser also introduces aspects that would obfuscate such a test, for example these tests are as mentioned dependent on JavaScript performance as well. In fact it is not even know beforehand if rendering can actually be quantified accurately, since browsers may very well skip rendering steps if it takes too long time.

In summary, the most obvious limitations of this experiment are that:
Metod

- Absolute values cannot be generalised as they are not representative.
- Performance is only measured for the web browser as a whole.
- Quantification is uncertain.

Reliability
There are many ways to systematically strengthen reliability. One way that works especially well with this experiment is serial tests (Stensmo, 2002, s. 31). Serial tests are also called the test-retest method and deal with the stability of the results. By doing the exact same test several times, one can spot results that are out of the ordinary.

Since it is not absolutely certain that graphics render on every API call, the unit of measurement is not called “frames per second”, but “updates per second”, where “updates” refers to the number of times the positions of the control points are recalculated and API call is made.

The test can be repeated again in the future since it is built using web standards, just as the previous experiment.

Validity
There are also many ways to strengthen validity. Concurrent validity means that two different measurements are made at the exact same time with different tools (Stensmo, 2002, s. 32). That is not possible in this case, since nothing can happen at the same time in a computer. Predictive validity means that two different ways of measuring are used at different times (Stensmo, 2002, ss. 32-33). First a prognosis or prediction is made with one method, and then another test is made with a different method. If values between the prognosis and the following measure match, it speaks for the validity of the prediction. In this situation however, there is no theoretical distinction between the two tests; they are not meant to follow in a particular order. If it had been possible, this experiment would instead have used the concurrent method.

There is a known issue with the SVG test, namely that it has to destroy old paths and create new ones each time the number n is increased. This takes time and creates overhead in the measurement. A third version of the test, that does not have this defect but on the other hand has to have its result interpreted manually, is used in the browser that suffers the most from the overhead. The result of this test can be seen as a prediction to the results of the other tests.

For the sake of this study, web pages are considered to be displayed on any modern computer, with different operating systems and system specifications. There is no way to determine what kind of capacity the web browser will have at its disposal, and also no way to determine what other processes may be running simultaneously, exploiting this capacity.

In order to minimise noise from background processes, browser network activity et cetera, the subject computers should be rebooted and allowed to fully load their normal state after logon before conducting the tests.

In real use-cases, the animations will not be predetermined and executed automatically, but instead react to user gestures through mouse movement events. The context webpage will also be different, with other (more) resources loaded. The algorithm that defines how the graphics will behave has been carefully written to be as simple as possible. This might however also change before the web application is presented to a real user audience.
Metod

In the prototype, control points are relative to a number of CSS attributes, accessed via the jQuery library, that define the appearance of its nodes. This could affect performance for real use cases, presumably negatively, since looking up several CSS values seems to be more complex than making up one random number per path.

Even when measuring “updates per second”, the unit of measurement bears little connection to the perceived reality. What refresh rate is perceived as good depends on both hardware, for example the characteristics of the computer monitor and human factors. These factors are beyond the scope of this experiment, and it should therefore not be used as a quality measurement.
Realisation
This chapter describes how the method was put to practice.

Mock-up and prototype

Material
- OS: Windows 7
- Mock-up tools
  - Mockingbird – web-based wireframe tool from Some Character
  - Paint – graphics tool from Microsoft
  - Common office material
- Web browsers (latest versions of the dominant web browser families)
  - Chrome 4 & 5
  - Firefox 3
  - Internet Explorer 8
  - Opera 10
  - Safari 4
- Microsoft Visual Web Developer 2008 Express Edition
- JavaScript libraries
  - Raphaël JavaScript Vector Library (v 1.3.1)
  - jQuery (v 1.4.2)
  - jQuery UI (v 1.7.3)

Procedure

The old mock-up
It was not certain from the beginning what kind of application was actually needed. There already existed an old mock-up, but the idea was to make an entirely new kind of solution. The old mock-up depicted a traditional table-based design which could easily be put together into a traditional web document, but it was not very dynamic or easy to comprehend. The idea was to make an application that was more of a visual programming tool on a par with Yahoo Pipes (although specialised to the task and not quite as complex).

A practical example
Easy as it might sound to make a diagram, there were many details that needed to be defined. Yahoo Pipes was looked at as an example of how this kind of application could work in practice.

The first analysis of Yahoo Pipes was very shallow and was not of technological nature (a technological study was made after the prototype had been made). This means that no effort was made to find out which technologies were used and how the page was structured at this point. What were looked at was how content was distributed logically and how the interaction worked. For example, how was content grouped in different boxes, what logical meaning do connections imply and how are connections made?

First of all, the mere amount of content and options found in Yahoo Pipes was considered to be much too complex for an application that is specialised to a single task. For example, there were
boxes that have no content of their own, but provided logic like a switchboard. Connections could sometimes also be made directly to components inside boxes, and not only to the boxes themselves.

Many features were however simple and inspiring. The following principles were derived or planned to be derived from Yahoo Pipes:

- The nodes look like typical windows from an operating system with a window frame and an X in the top-right to close the window.
- Nodes are automatically resized when their contents change.
- New nodes can be dragged onto the working area and then connected manually.
  - *This was first simulated with buttons (no dragging), but the feature was later discarded in favour of a simpler solution.*
- The end node, called *Pipe Output* in Yahoo Pipes, is placed out automatically and can be connected to from several branches.
  - *This was attempted with the final alert to the security company, but it was later regarded unwise to join branches of the tree model for this application.*
- No logic is put inside the connectors.
  - *For example, a connection between two messages could have a time delay, but delays were instead put in the message boxes.*

**The new mock-up**

After mixing original ideas with principles from the practical example, a new mock-up began to take form. The first procedure was an attempt to simply extract the items from the old mock-up and distribute them into boxes with different placement in the new tree diagram. Basically this went by smoothly; the only question was which order the boxes should be put in and how they would be allowed to relate to each other. The connecting paths represented the logic “has a” or “is followed by” or “and”, depending on which the connecting boxes were.
Figure 11: The final version of the mock-up before real implementation began. The disposition of content between boxes was set, but would later change a bit. Interaction and connection behaviour was very abstract at this time.

After a period of internal discussions based on mock-ups and an early proof-of-concept, a decision was made to adapt the proposed solution. The exact distribution of content between boxes was not set at this point, but was instead discussed further when using the interactive prototype.

The contents could, with some exceptions, be represented by input elements that have been around in HTML for a long time. They were also represented in the mock-up program that was used, and since the prototype would undergo a design evaluation process before becoming a finalised product, it was decided to rely as much as possible on these predefined interaction elements for the prototype.

For selecting time delay, a counter would be useful; instead a select menu is used. There is also no generic list with selectable items, so this has been custom-created.

**State of the standard**

At this point it was clear that the connecting paths were the objects of interest. This kind of interactive graphics had not been possible to make in a standardised way in previous versions of HTML and XHTML. SVG had however been along for a long time as a solitary standard, but not implemented to the same degree as the other standards. The same was true for Canvas. Thus, after some further research, Canvas and in-line SVG became candidates for further testing.

Simple empirical research was conducted in all browsers. There were many Canvas tutorials and it seemed to work fine overall, with the exception that Safari 4.0.5 was unable to zoom <canvas>
elements. In-line SVG was harder to make work. There was example code in the HTML5 specification that did not work at all when tested, and there were third-party examples that worked in all browsers where SVG had previously been implemented. Sometimes early tests made by the author only worked in some browsers, and online reports and articles about topics were quite contradictory. Because of these difficulties with in-line SVG and the lack of good information, it was clear that something had to be done, and a conformance experiment was later conceived. The situation called for no further analysis of Canvas.

The question was then which technology the prototype should use, or if both solutions should be attempted. Since the final application should be compatible with the most used web browsers at the time, namely Internet Explorer (8), and statistics collected in the beginning of the project showed that none of these technologies were natively available widely enough among end users, a conclusion was made that none these technologies could be used as-is in the prototype. An alternative was found and the idea was then to separate the graphics from the rest of the code so that a change of technology would be easy to make in the future. More focus was instead given to the upcoming experiments, that would still use Canvas and in-line SVG.

The new solution for the prototype was a JavaScript library called Raphaël. It uses object graphics, which were at the time considered as more having potential as a candidate for this prototype according to the literature study. More specifically it uses stand-alone SVG where implemented and switches to the proprietary standard VML in Internet Explorer.

While Raphaël became extremely useful in the development of this prototype, it was limited to the intersection of capabilities between SVG and VML. It was also an extra library that would have to be fetched from the server every time a page is browsed.

All things considered, Raphaël was confirmed to be capable enough to implement the features that were specified during the development period and also had features that had a potential use in the future.

**Design**

One of the first technical design issues that arose was if the connections should be drawn on one big element or if each connection should reside within its own root element. The former design was chosen since it seemed like the simplest solution. After that, no attention was given to the latter one. No investigation of the underlying design Yahoo Pipes was made at this time.

As mentioned earlier, interaction was still abstract at this point. The idea was that connectors could be somehow dragged and dropped on other boxes. They would then be attached to the frame of the box, but not fixated to a specific point. Instead, both endpoints of a connection would float around, seemingly arbitrarily. This was never attempted in practice since it would be unnecessarily time-consuming to implement. After the fact, it would also have made the performance experiment harder to validate.

The whole idea of connecting boxes manually was later abandoned for much simpler interaction. Since all possible combinations are known beforehand, each box knows which children it can have, and so these children can be created directly from their parent and their connections are made automatically.
Figure 12: The picture depicts a serial flow, but during the prototype development period, it was never finally decided if branching should be allowed at all. Translated from Swedish for this report. The order is implied by the paths start- and endpoints.

Things that have been decided internally (with lesser reference to HCI studies) include:

- Content separation:
  - If days, times and events separate or not.
  - If messages and addressees separate or not.
  - If delays should have separate boxes or be stored inside content that can be delayed.

- Ordering:
  - The order between days, times and events.

- Making connections.
  - Manual or automatic.

- Parallels between objects.

- Root nodes.
  - If the root should be the day, time, event or a special root box.
Genomförande

- If there should be one or many roots per work area.

- Terminators (have no children).
  - If they should be mandatory or not.
  - Allow one per branch or have one that is unique.

There has been some professional feedback on the interaction design subsequent to the development period. The main issue is that new users might not understand how to create boxes, or that it is even possible. Dragging is too advanced and nothing suggests that interaction can be made. To solve this, connectors should be clickable and make new boxes appear at convenient placements. They should also look like buttons or somehow suggest that they are clickable.

Summary
The following decisions were made:

- The overall size of the tree diagram should not be dictated, as it was still not known how it should best be regulated.

- The order of the boxes was set to: Rule – Event – Time – Message (three kinds, of which two could be followed by more messages).

- The Rule box became the root, and it would be unique. It was at first possible to have several subsequent events for a rule, but was later found to be unwise. The rule box was still not removed or combined with the event box after this fact, since it had its own contents that did not fit together with event settings. This sometimes seemed redundant, but redundancy could arguably help new users understand what is being visualised.

- Alerts to security company were terminators in the sense that they could have no children, but they were also not mandatory due to the way Gardio’s service works.

- Connections were made (and unmade) automatically, since this was easier and more effortless than making them manually. This was possible since the application is specialised to the task.

- Days and times were combined into one object. Unlike rules and events they were not unique could in other words be parallel.

- The following templates should later be implemented (but were not done as a part of the project):
  - Standard alarm: An alarm event with one time (everyday and around the clock) followed by two decision messages and one alert to security company, all in sequence.
  - Alarm status check: An event that would activate when the alarm was not armed at 09:00, Monday-Friday, followed by one information message.
**Conformance test**

**Material**
Notepad

**Test subjects**
- Chrome 7.0.517.41
- Chrome 8.0.552.23 developer preview
- Firefox 3.6.12
- Firefox 4.0 beta 6
- Internet Explorer 8.0
- Internet Explorer 9.0.7930.16406 beta
- Opera 10.63
- Safari 5.0.2

**Test Objects**
A total number of 14 files were created and put on a web server. Five of them had the file type “html” and the other nine had “xhtml”, which made the server send them with content type “text/html” and “application/xhtml+xml” respectively.

The HTML5 code inside these files had a total of nine configurations; one un-edited example from the HTML5 specification (both file types), four configurations for test 1 (both file types) and another four for test 2 (xhtml only).

The code variations for the test objects are explained in detail below.

**Un-edited example mark-up**
This example was found in the HTML5 specification.

```html
<!doctype html>
<title>SVG in text/html</title>

<p>
    A green circle:

    <svg> <circle r="50" cx="50" cy="50" fill="green"/> </svg>
</p>
```

**Modified mark-up for test 1**
This was the first attempt at modifying the code in a standard-compliant way in order to achieve better results than for the original mark-up.

The “doctype” was capitalised and the `<html>`, `<head>` and `<body>` tags with respective closing tag were added, since those were the requirements for the old standard. (These changes were variables as they never changed – variables are explained later).

```html
<!DOCTYPE html>
<html>
```
Further modified mark-up for test 2
In order to further improve results for files that were sent as “application/xhtml+xml”, the XML namespace attribute was added to the <html> element.

```html
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>SVG in text/html</title>
</head>
<body>
    <p>A green circle:
        <svg> <circle r="50" cx="50" cy="50" fill="green"/> </svg>
    </p>
</body>
</html>
```

Constants
The constants are summarised below.

**Test 1 and 2**
- Tags
- Uppercase DOCTYPE
- Web server: www.student.nada.kth.se

**Test 2**
- XML namespace
Variables
The variables are summarised and explained below.

File types – content types

<table>
<thead>
<tr>
<th>File type</th>
<th>Content type</th>
</tr>
</thead>
<tbody>
<tr>
<td>html</td>
<td>text/html</td>
</tr>
<tr>
<td>xhtml</td>
<td>application/xhtml+xml</td>
</tr>
</tbody>
</table>

Doctype data
No extra data:

```html
<!DOCTYPE html>
```

Obsolete permitted doctype string – HTML version:

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

Obsolete permitted doctype string – XHTML version:

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

SVG namespace
Without namespace:

```html
<svg>
```

With the latest recommended namespace and version of SVG:

```html
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
```

Procedure
The procedure was iterative. Pre-tests were conducted on several occasions with the original mark-up and summed up 2010-07-22. These tests used older versions of the web browsers.

Test 1
Test 1 was first conducted 2010-07-29. This time browsers were updated to the latest version at the time. The previously explained test object variations for test 1 were used at this time, along with the original mark-up, and results were documented again. According to the HTML5 specification there were four so-called “permitted obsolete doctype strings” for HTML, and two for XHTML. The strings chosen for this test were the most recent versions with the most extra information.

Choices for HTML:

- "-//W3C//DTD HTML 4.0//EN"
- "-//W3C//DTD HTML 4.0//EN http://www.w3.org/TR/REC-html40/strict.dtd"
- "-//W3C//DTD HTML 4.01//EN"
- "-//W3C//DTD HTML 4.01//EN http://www.w3.org/TR/html4/strict.dtd"
Choices for XHTML:

- "//W3C//DTD XHTML 1.0 Strict//EN http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
- "//W3C//DTD XHTML 1.1//EN http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"

Test 2
Test 2 was first conducted 2010-08-02. It was designed to improve results for XHTML, which was from the literature study known to have working examples for more browsers than had a positive result in this test. An answer was looked for in these examples and in the HTML5 specification. It came to mind that examples had an XML namespace, which test 1 did not have. This was probably overlooked when reading the specification, because the XML namespace is an attribute of the <html> element, which is not mandatory. No updates for the web browsers were available for manual download and they had not been automatically updated either, so the versions used were the same and the previous tests did not have to be repeated.

Revising the whole experiment
Some time after the original experiment was performed, it came to the authors' knowledge that it had been based on an uncertain factor. In the first round of tests, test files were opened locally and were not fetched from a web server. This could have given subjective results as no content type was given by a server, and it was unknown how the test subjects would interpret this. Therefore, all tests, including pre-tests for the original files, were conducted again. It is the details for this final experiment that has been explained in this report. The differences were that the content type was known and more current set of browsers were used. Tests were conducted 2010-10-30 for all browsers except Internet Explorer 9 beta, which was tested 2010-11-05. Since other browsers versions were used, it is not possible to tell whether or not the alterations improved validity.

Performance test

Material
Notepad

Test subjects
- Chrome 5.0
- Firefox 3.6.8
- Opera 10.61
- Safari 5.0.1

Test objects
Computer A – Acer TravelMate 4280:
OS Windows XP SP2
CPU Intel Core 2 Duo; 1,66 GHz
RAM DDR 2 – 3 GB – 667 MHz

Computer B – Apple iMac 24-inch:
OS Mac OS 10.6.4
CPU Intel Core 2 Duo; 2.93 GHz
RAM DDR 3 – 4 GB – 1067 MHz

Canvas test A and B
SVG test A and B
Real-time tests for Canvas and SVG

All tests available at:
http://www.student.nada.kth.se/~steho/tests/

Procedure
The performance tests were conducted on two computers simultaneously: one was running Windows XP and one was running Mac OS 10.6 – the most represented versions of the largest operating system families according to statistics. The hardware on which these operating systems ran was not chosen deliberately.

At the beginning of the experiment session, all web browsers were checked, updated or installed to have the most current versions at the time.

All current versions and previews were aimed for the tests, but the preview version of Internet Explorer 9.0 was not available on these two operating systems, as it required Windows Vista SP2 or later. This would cause reduction of important data in a way that would make the comparison incomplete. At that point it was decided that if this browser was ever to be tested in the future, it would be better to test all browser previews on the most recent operating systems, together. In other words, the Chrome and Firefox betas that were also targeted were not tested at this time either. Time was also sparse and this would on the other hand ensure that all tests could be finished during the same session.

A preliminary test was made in Firefox on both computers in order to estimate the total time required for testing this new set-up. Firefox was used because it had generated the longest runtimes/lowest updates per second during the development of the test suite. The time limit for test A was then set to 30 seconds and the number of updates for test B was set to 100. 30 seconds was recognised as generating good enough values while not taking too long time. 100 updates was decided upon beforehand to generate about 60 seconds of runtime for 1024 SVG paths test B in Firefox.

Before testing began, both computers were rebooted and user programs were shut down to minimise disturbances that could affect test results. Screensavers were disabled and an eye was kept at the machines while the tests were running to see that nothing unexpected happened.

To further insure that nothing went wrong, a prognosis was estimated using the real-time updates-per-second-test. This was done on the entire test interval for Firefox, with both Canvas and in-line SVG and on both computers. Estimated values resembled test results very closely, and therefore estimated values were only observed for the first (1) and last (1024) number of objects during the rest of the session. Every prognosis was satisfactory.
Testing took about 3-4 hours and could be completed in one session.
Results

Prototype
The mock-up has transformed into a working prototype that acts as an example of how the proposed idea could be made into an HTML5 web application – although the prototype itself does not use HTML5 for rendering paths. Alternatives for an HTML5 solution are represented theoretically. The project has also accumulated theoretical and empirical factors of significance.

The following technical information is defined as a result of the development process:

- All possible relationships between boxes are known beforehand. In other words, each content type has a specified set of child content types.
- Connecting paths are created and deleted automatically along with the child box.
- Connecting paths are not clickable and do not have to be.
- The flow is implicitly forced to become a tree diagram with no interconnecting branches.
  - One root per document.
  - Only leaves can be deleted.
- Cubic curves are used throughout the entire application.
  - Tethered to box corners: from the bottom-right corner of the parent to the upper-left corner of the child.
  - The positioning of the control points is fixated relative to their nearest endpoint. In other words their positions are calculated by adding or subtracting a constant value; not by generating further variables or using complex algorithms.
  - Only one endpoint of each path can move at a time, but several paths may be animated simultaneously (since there may be branches).
- The contents are dynamic and so is the size of the boxes. Connections are redrawn when boxes are resized.
- Two standard setups exist. They are both linear combinations, and the longest flow has six boxes and five paths.
- There is no theoretical limit to how many boxes there can be in total, and some boxes can in theory have an unlimited number of children.
- The prototype does not overlay paths on boxes; they are always beneath.

Conformance test
Browser icon marks correct title, text and rendering of a green circle for given combination.

Test 1:
Table 5: Results for test 1.

Test 2:
Table 6: Results for test 2. Only XHTML files have been changed for test 2.

Performance test

Note that the initial number of objects is 1.
Genomförande

Opera 10; Computer B

Canvas
In-line SVG
Analysis

Prototype

The results of the prototype development tell something about how the proposed idea should be made into an HTML5 web application. By defining its technical requirements, it is now possible to know exactly how SVG and Canvas can be used to put it into practice.

Cubic curves have been proven as suitable for the prototype. Technically, straight lines would for most purposes give the exact same result with less complexity, but curves give a more organic feel and are arguably easier on the eye. Although they explain the relationship between boxes implicitly, it would be easier to understand the direction of the flow if they were somehow marked with arrows.

Simplicity is of importance. Interaction has been simplified by automatically handling paths and not giving this task to the user. This is achievable because all possible relationships are known beforehand. The interaction design is also the reason to why constructed message flows must become tree diagrams and why only leaves can be deleted. This means that paths will never have to be clickable or otherwise mouse-interactive in a direct sense, which will give SVG no inherited advantage in that regard. However, the theory of object-oriented graphics suggests that the animation should be optimised so that only affected objects are updated. This can always be utilised. Object-oriented graphics would also make it easier to highlight or otherwise draw attention to specific connections, if this would be desired in the future.

The algorithm that defines and animates the paths has been set. Fixated control points and only one endpoint moving at a time will aid performance and also make it possible to conduct an accurate performance test. If for example both endpoints could be animated simultaneously it would be more difficult to construct a performance test that is true to real use cases.

Theoretically both Canvas and SVG elements can be used to overlay certain objects to create a layer effect. For Canvas, a new element would have to be created for each object. For in-line SVG, one root element should still be enough since X/HTML code can be nested inside this element. Layer indices can also be edited in both cases. Advanced layering should not come with any negative impact, but then again none of these more advanced alternatives have been proved in practice in this study.

The total number of paths is unknown and the number of parallel paths is also unknown. In other words, not only is the upper number of paths unknown, but also the number of nodes that will have to be animated at once. Whether or not boxes are dragged around with the mouse, they could also cause the paths to reanimate when the contents change and the box is resized. This is why only affected paths should be animated.

It is however not likely that the sheer number of paths will be much larger than those of the default flows in a typical use case, but the upcoming design process or other future updates could on the other hand increase the complexity of the graphics. For example, a second path could be added to make a shadow.
Conformance test
The results of the conformance experiment tell something about practical limitations for application. By identifying the current implementation status of in-line SVG, it is now possible to give a more general picture over the interoperability of this technology.

XHTML context
For in-line SVG in XHTML, only one current web browser is conforming without having to use the XML namespace in the <html> element — which is an optional element. With the XML namespace given, all current browsers except for Internet Explorer 8.0 (which has no support for SVG whatsoever, but is mentioned for completeness) support in-line SVG, but only if the SVG namespace is also given. The other variable — the obsolete permitted doctype string — is irrelevant in this matter. It should be noted though, that only uppercase “DOCTYPE” was tested, so it is not sure to say that the doctype header as a whole can be disregarded.

HTML context
Future versions of the three most used web browser families do support in-line SVG in HTML as well. The families are Chrome, Firefox and Internet Explorer. Chrome also supports in-line SVG in HTML in its current version. In every case, the simplest version of the document passes the test, thus all variables are irrelevant.

Summary
Only XHTML, that is, content type application/xhtml+xml, can be used across all major web browsers today. Both SVG and XHTML are applications of XML, and so this is not very surprising. It seems however that in-line SVG will work better with HTML in the future, since all tests were positive in the browsers that conformed to it in any way. This is perhaps less expected, but shows that the effort in making the two mark-up specifications compatible is indeed being taken serious.

Performance test
The results of the performance experiment tell something about practical limitations for choosing and designing applications. By comparing how Canvas and SVG perform in different browsers, it is now possible to tell which technology seems to be faster and which browser is currently the slowest and therefore should be targeted for testing.

Expected Values
Generally test results very closely within the expected range. Difference about 1 FPS, usually the expected value was higher than the result. Expected values were hurriedly rounded up when fluctuating, which may explain this.

The test is analysed below from several angles.

Test A
This test used a fixed time limit to measure the number of updates. The time limit was 30,000 milliseconds. As explained before, it is not possible to break an operation thus accurately; this explains the time overhead in the results. Generally, the overhead was higher the more objects there were to animate, which was expected and noticed in early pre-tests.

On Computer A (running Windows XP), the maximum overhead was observed in Firefox and had a value of 212 ms. Firefox had a three-digit overhead for 1024 objects of both Canvas and SVG, and
was the browser with the most overhead overall. Opera was the only other browser to have three-digit overhead, with a maximum of 155 ms.

On Computer B (running Mac OS 10.6), the maximum overhead was observed in Opera and had a value of 92 ms.

**Test B**
The parameter to test B, number of updates, was chosen to achieve about one minute of rendering time for 1024 SVG objects in Firefox. Rendering times were about 56 seconds for 1024 SVG objects and about 29 seconds for 1024 Canvas objects. It was significantly less in other browsers.

**Comparing series 1 and 2**
Serial tests 1 and 2 were combined using mean values. Generally numbers were very stable to begin with, meaning that the difference was usually one to two seconds with some exceptions. This means that the experiment suffered from little background noise and speaks for the reliability of the experiment.

SVG test B on computer A seemed however to suffer from a more significant disturbance with inconsistencies of up to 30 updates in Chrome. A handful of isolated differences could otherwise reach up to 7 updates per second.

**Comparing test A and B**
Tests A and B were also combined using mean values. With some exceptions, the different methods of measurement were also very stable to begin with, having most numbers differ only by one update or less. This speaks for the validity of the experiment when it comes to comparing the methods of measurement.

The most notable exceptions were found in Chrome on Computer A and Opera on Computer B. Chrome showed seemingly random inconsistencies between SVG test A and B for 1-64 objects with a maximum difference of 6.5 updates per second. Canvas test B showed equal or higher values for all object quantities in Opera. The maximum difference was 3 updates per second.

**Comparing Canvas and SVG**
Generally SVG is faster with lower number of objects to update. The two technologies then converge in performance at different points, before diverging again when Canvas takes advantage.

**In general**
In most cases the performance graph is stable for low number of objects and eventually drops exponentially. The exception to this general trait is Safari on Computer A, where both technologies seem to have a performance peak with more than one object to update. The reason cannot be explained, and it is not that relevant either, but it raises the question if there could be more peaks, dips or other irregularities between sample points.

**Summary**
It is quite unexpected that SVG is sometimes more efficient than Canvas – it contradicts the hypothesis and what is generally said about the matter. It is not within the scope of this study to analyse why this might be. Another question is why Canvas surpasses SVG for large amount of
Slutsatser

objects. Perhaps this is where Canvas is actually supposed to excel – when the workload becomes very high – and not for small amounts of operations. This is however also uncertain.

There is a known defect in the SVG test that could be an explanation (it has to destroy and create new objects every time the number is increased, which means more overhead the more objects there are), but on the other hand numbers were still consistent with the pre-test that does not have this defect.

Browsers have different performance. Chrome 5 is by far the fastest browser while Firefox 3 is somewhat slower than the rest. This means that Firefox 3 should be targeted for testing.

A secondary, abstractly formulated, goal with the experiment was to study the performance graphs for general traits. The graphs show that performance never hit zero during set circumstances, but came very closely as they decrease exponentially. It probably means that steps that take too long are skipped by design. Tested web browsers did not seem to become completely unresponsive, as graphics continued to update, albeit at a noticeable slow rate. The alternative would have been that every update was rendered on screen, but at an incredibly slow manner. It is unknown if this is a trait of the JavaScript engines or the web browser engines, but it is nonetheless a positive one. Again, the experiment was not explicitly designed to test this aspect (but the upper amount of objects was chosen in a way that would generate results close to zero in early pre-tests).
Conclusions

Recommendation

The standard

HTML5 is a way to get together different technologies that are relevant on contemporary web pages. It has extended well beyond the original intent to give easy access to information by now allowing for both rich and interactive visualisations that can indeed be used to realise Gardio's application and much more.

The ecosystem of open standards is constantly growing to encompass all that is found on contemporary web sites. They are also being tightened together and made compatible with each other which allows for advanced web applications to be easily developed. This will surely compel all major browser vendors to implement as much as possible from these standards and not only some or parts of them.

The sustainability of HTML5 is high because it deals with both new and old features. New solutions are often proven in practice and discussed publicly before they are standardised, so that it is known how they can implemented and that they fill a practical function. Old solutions that are replaced by new ones are not deprecated, which means that they will also continue to work.

The market

A solution based on HTML5 can be realised today in four of the five most used web browsers, which together constitute for about 50 per cent of the market. This is not considered enough. There are ways to extend support to the fifth browser, which roughly holds the other 50 per cent. One solution which is based on object graphics utilises another mark-up language in Internet Explorer and is used in the prototype. Another solution based on raster graphics exists, but has not been attempted in this project.

For in-line SVG, the solution has further requirements. Namely, it has to be sent as a document with content type application/xhtml+xml, that is as an XHTML document.

The application

The graphics does not have to be mouse-interactive, which means that both SVG and Canvas can be inherently used without the need of aiding scripts.

The most complex standard rule uses five paths of which at most two are animated at once. While the performance experiments are neither made to be representative for typical users, nor give exact values for actual rendering frame rates, they indicate that animating fair amounts of paths is something that can be done using todays’ hardware and technologies. What the tests showed was that the script for the in-line SVG solution updated faster for lower numbers, but not in a way that would give Canvas a demerit. As mentioned, the test gives no generalisable absolute values, but in these particular cases, at least 16 paths could be animated without significant performance drop. Performance should not be an issue for the most probable use cases. If however a very large amount of paths were to be rendered simultaneously, it seems like browsers in general will not become completely unresponsive, which is a positive trait. Visual performance will however drop noticeably.
The tests do suggest that Firefox 3 should be targeted for real-application testing on both Windows XP and Mac OS 10.6 (the most used versions of Windows and Mac OS at the time of this study), since it had the lowest general performance. Opera 10 and Safari 6 were slightly faster while Chrome 5 was mostly superior. It is important to have in mind that this suggestion refers to specific versions of browsers, not entire families. New versions of each browser family might improve performance considerably.

From a performance perspective, it is recommended to only render the paths that are actually moving. This is inherently achievable with object graphics – which in this case means in-line SVG – while it can also be done by designing the application with several Canvas elements. The latter has not been tested during this study.

It is not known whether zooming the application will be a probable use case, but this would then give a slight benefit to object graphics because it renders more accurately, although it is in no way necessary for making the application understandable.

The main implications are summarised below, ordered by level of urgency:

1. Object oriented animations can be used to only animate the paths that need to be animated. This is not expected to be a problem for most use cases, but rather because of heavy graphics.
2. In-line SVG can only be used with content type application/xhtml+xml, while Canvas can be used for both HTML and XHTML.
3. Zooming will distort graphics but the actual connection will be visible. Cosmetic concern.
4. Interacting directly with paths is not necessary in the prototype.

Summary

HTML5 and related technologies can be used for this prototype in most web browsers but the one most wide-spread. In the future all dominant web browsers will support both Canvas and in-line SVG.

For now it is recommended to use another solution. The prototype works in all present versions of the top five browsers.

It is not known when an HTML5 solution could be deployed, as it depends on Internet Explorer version 8 to lose market share to an HTML5-compatible browser. It is likely that a large share of Internet Explorer-users will upgrade to the latest version quickly, but according to statistics some amount of users will also lag behind.

Going forward

It is recommended to continue to keep things simple because inconsistencies in browser conformance may exist for some time to come. When testing with users or otherwise making up new features, make notes of which concerns are presentational and which have to do with content. These are central pillars in web standards and are kept apart. Relate requested features to either of these categories.

Be careful with claims on conformance. Test results are only relevant for the actual test. If no tests are referenced and only general claims are made, be extra careful. For SVG graphics, check for in-line
SVG support in both HTML and XHTML. Also make sure that the Canvas 2D API is implemented, and not only the Canvas element.

**Discussion**

**About the outcome**
The HTML5 specification is still in development, but that is true for all web technologies. The literature study shows that the specification is progressing very intimately with practical implementations. It is possible but unlikely that the basic concepts described in this report will change, as most browsers already implement them and future versions will too.

Canvas can be made to work in some browsers that do not support the standard by emulating its functionality with a script, and composite Canvas graphics can in fact also become mouse-interactive using scripts. The prototype uses one JavaScript library to exploit different standards in different browsers. One could argue that there are always workarounds for the inoperability-issue of web standards, but then again most features could also be made using third party plug-ins, which too have their benefits and detriments.

**About the project**
In the beginning the term action research felt very vague and was hard to comprehend and shape into a meaningful way of work. Action research is also usually explained with attempts on human subjects, which does not help understanding in this case.

A large part of the project was devoted to getting a basic understanding the situation. It was difficult to find hard facts in the literature study. Perhaps this is because the web is still young and therefore a relatively new research topic, in comparison traditional fields of science. On the other hand, a fresh view helped line out basic difficulties that could otherwise have been overlooked.

In review, it is however clear that action research was the right way to deal with the problem at hand, but the process would indeed have been more efficient if the area of research had been more well-known beforehand. New literature was discovered and read throughout the entire project, which is enlightening but ineffective. While general understanding grew rapidly, many ideas had to be discarded along the way.
**Literature list**


51
Litteraturlista

http://www.w3.org/2000/09/dbwg/details?group=40318&public=1

http://www.w3.org/TR/xhtml1/

http://www.w3.org/standards/webdesign/graphics


http://www.w3schools.com/dom/dom_nodes.asp

http://www.w3.org/TR/html5-diff/

http://wiki.whatwg.org/wiki/FAQ


http://www.whatwg.org/charter

