Creating Rich Internet Applications with Modern Tools

Leo Martin Westby

Master of Science in Informatics
Submission date: June 2015
Supervisor: Terje Rydland, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

With the advent of small devices such as smartphones and tablets that can display web pages, there has been a growing interest in creating web pages with functionality and complexity comparable to native applications. These types of web pages are called rich internet applications. Creating rich internet applications instead of native applications that target a specific device has the potential for saving much development time as well as reaching more users.

The goal of this thesis is to investigate the technology currently available for creating rich internet applications and build a prototype for Emissions, a web-based simulation tool for learning that will be used by Norwegian eighth graders to learn about space-related activities. The prototype developed will be evaluated to see if the tools that exist today are sufficient to create complex web-based applications.

The prototype was developed as a pure web application in HTML5 using the WebRTC API for setting up video calls between users and a server written in Node.js for saving state and sending WebRTC signals between clients. Evaluation of the prototype found that only a few of the internet browsers available today support advanced HTML5 features such as video chatting and playback of WebM encoded video files.

Based on these findings, the technology that exists today was not considered sufficient for creating complex rich internet applications that are portable across multiple platforms. However, if a less portable approach is chosen by specifically targeting a handful of modern browsers, it is possible to create rich internet applications with the same functionality as native applications.

Sammendrag

Med lanseringen av smarttelefoner og nettbrett som kan vise internettsider har det blitt mer populært å utvikle nettsider med samme funksjonalitet som integrerte programmer. Denne typen internettsider kalles innholdsrike internettapplikasjoner. Å lage internettapplikasjoner istedenfor integrerte applikasjoner som må skreddersys til en spesifikk type enhet sparer potensielt mye tid og øker muligheten for å nå fram til flere brukere.

Målet med denne oppgaven er å undersøke teknologien som er tilgjengelig i dag for å lage innholdsrike internettapplikasjoner og lage en prototype for Emissions, et nettbasert simuleringsverktøy som skal lære norske ungdomsskoleelever om romrelaterte aktiviteter. Den ferdige prototypen vil bli evaluert for å se om teknologien som eksisterer i dag er tilstrekkelig for å lage internettapplikasjoner av høy kompleksitet.

Prototypen ble utviklet i HTML5 og brukte WebRTC rammeverket for å implementere videosamtaler mellom brukere i nettleseren. En server utviklet i Node.js ble brukt for å lagre tilstand og sende WebRTC-signaler mellom klientene. En evaluering av prototypen fant at bare noen få av nettleserne som er tilgjengelige i dag støtter avanserte HTML5 funksjoner som videosamtaler og avspilling av videofiler kodet i WebM-formatet.

Basert på denne evalueringen ble ikke teknologien som eksisterer i dag ansett som god nok til å lage plattformuavhengige internettapplikasjoner. Hvis man derimot velger en plattformavhengig tilnærming ved å utvikle spesifikt for et lite antall moderne nettlesere, kan man lage internettapplikasjoner med samme funksjonalitet som integrerte programmer.

Preface

The contents of this document describe the work done during the 5th year of a master degree in software development at NTNU by master candidate Leo Martin Westby under the supervision of Assistant Professor Terje Rydland. The work was done over the course of two semesters starting in September 2014 and ending in June 2015. I would like to thank my supervisor Terje Rydland for his valuable guidance and feedback during the project period and the people at Andøya Rocket Range for their cooperation with creating Emissions.

Leo Martin Westby
1. juni 2015, Trondheim

Contents

List of Figures

List of Tables

List of Abbreviations

API	Application Programming Interface
ARR	Andøya Rocket Range
CO2	Carbon Dioxide
CSS	Cascading Style Sheets
DOM	Document Object Model
ECG	Electrocardiogram
GPS	Global Positioning System
HTML	HyperText Markup Language
IP	Internet Protocol
IPv4	Internet Protocol version 4
JS	JavaScript
JSON	JavaScript Object Notation
mV	Millivolt
NAROM	Norwegian Centre for Space-related Education
NAT	Network address translation
NTNU	Norwegian University of Science and Technology
RIA	Rich Internet Application
STUN	Session Traversal Utilities for NAT

TURN	Traversal Using Relay NAT
UML	Unified Modeling Language
URL	Uniform Resource Locator
VoIP	Voice over IP
W3C	The World Wide Web Consortium
WebRTC	Web Real-Time Communication
Sv/h	Sieverts per hour
XML	Extensible Markup Language
XUL	XML User Interface Language

Chapter 1

Introduction

The original world wide web was a platform for displaying static content such as text and images where user interaction was limited to entering text into standardized forms and following hyperlinks to other web pages. To display new information or give feedback to the user, the entire web page had to be refreshed. The advantage of traditional web pages is that they follow a uniform and simple standard that allows a user to visit a web page without installing additional software on his computer, which saves time and hard disk space.

For tasks requiring more interactivity and computing power, desktop applications were used. The drawback of desktop applications is that they must be installed on a user's computer before they can be used, a process which can take several minutes and reserves space on the user's computer. If it turns out the application did not offer the right tools to complete the task, the user must then uninstall the program by using an uninstaller, a process which can also be cumbersome. Some uninstallers are also incomplete, leaving behind shoRTCuts, library files and registry entries that the user must manually delete if he wishes to avoid clutter.

Rich internet applications (RIAs) try to combine the advantages of traditional web pages and desktop applications and eliminate the disadvantages. RIAs are web applications, which use data that can be processed both by the server and the client. The data exchange takes place in an asynchronous way so that the client stays responsive while continuously recalculating or updating parts of the user interface. On the client, RIAs provide a similar look-and-feel to desktop applications by offering a variety of operating controls and transparent usage of the client and server computing power and of the network connection (Busch and Koch, 2009).

1.1 Background

NAROM is a government funded organisation that initiates, develops, and performs educational activities related to space activity such as space technology, space physics, atmosphere and environment. The organisation is located on Andøya Rocket Range (ARR) on the island of Andøya in Northern Norway.

NAROM is currently working together with NTNU to create Emissions, a web-based simulation tool that will be used during high school science classes to teach Norwegian children about space related concepts. The tool will be created by master students at NTNU, each student focusing on different parts of the implementation.

NAROM hopes that Emissions will raise awareness for the Norwegian space center on Andøya and that by demonstrating some of the of work that goes on at the space center, more Norwegians will be interested in pursuing a career in space related technology.

1.2 Research questions

This thesis will answer the following four research questions:

1. Is the technology that exists today sufficient for creating cross-platform rich internet applications with functionality comparable native applications?
2. What technology exists today for creating rich internet applications and what are the advantages and disadvantages of these technologies?
3. What technology will work best for implementing Emissions?
4. What are the functional and non-functional requirements of Emissions?

To answer the first question I will create the prototype for Emissions, a RIA that will be used in schools for teaching. The prototype will require complex functionality that is typically only found in desktop applications, more specifically updating information in real time, communicating bidirectionally with a server and allowing users to chat with other users of the same application via an integrated video chat.

Hence, evaluating the finished prototype on multiple platforms will give some insight into whether or not the technology that exists today is sufficient for creating cross-platform RIAs with functionality comparable native applications.

To answer the second question I will investigate recent literature related to RIAs as well as blogs and newsgroups to figure out which tools developers working in the industry today are using to build RIAs and what the advantages and disadvantages of each of these tools are.

To answer the third question I will use the requirement specification and my technology evaluation to find the technology most suited for creating Emissions.

To answer the fourth question I will work together with NAROM to create a requirement specification covering both the functional and non-functional requirements of Emissions.

1.3 Approach

The research approach used in this thesis is a theoretical study of rich internet application technologies and a practical experiment of these technologies in use. The research will be conducted according to the following steps:

1. Define the problem and research approach.
2. Investigate recent literature done on the subject of rich internet applications.
3. Elicit the functional and non-functional requirements of Emissions in cooporation with NAROM.
4. Conduct a technology evaluation to figure out which technology is most suited for implementing Emissions.

5. Implement a prototype for Emissions.
6. Evaluate the prototype in order to figure out the current state of RIA development.

1.4 Thesis structure

This thesis is divided into seven chapters with the first chapter being the introduction that describes the background for the thesis, the research questions and the research approach.

The second chapter will give an overview of some of the recent research done on the topic of rich internet applications.

The third chapter will describe the design and requirements of Emissions.

The fourth chapter will outline the technological choices that were made prior to developing the prototype.

The fifth chapter will explain and show the implementation of the prototype.

The sixth chapter will evaluate how well the prototype implements the requirement specification.

The seventh and final chapter concludes by answering the research questions asked in the introduction and describes the work that must be done before the prototype can be used in a real scenario.

Chapter 2

Related Work

The development of cross-platform rich internet applications has been a popular research topic in recent years. This chapter will give an overview of some of the research done in this area.

2.1 RIAs compared to traditional web pages

Farrell and Nezlek (2007) investigated the advantages and limitations of rich internet applications compared to traditional web pages and categorized RIAs into three distinct types. The advantages they found are that RIAs do not require a page refresh for every user action, which avoids redundant data being passed between the client and server and improves the user experience. Furthermore, web developers are able to create more complex user interfaces that closely mimick their desktop counterparts.

The limitations they found are that RIAs are built on a set of technologies that have a history of potential incompatibilities between browsers and platforms. This means that an application may not work as expected in one or several browsers, which may inadvertently deny users of unsupported browsers access to a service. They also found that because of the increased complexity of RIAs, larger-scale development and deployment is hampered by lack of a coherent architectural style for applications and libraries. They suggest that a style derived from similiar processes used in the creation of desktop applications should be used when developing and deploying RIAs to handle the increased complexity.

They argue that the biggest limitation of RIAs is accessibility. Because of the lack of a unified standard, users may not be aware of how application data updates are occurring, so they may not be able to properly identify or locate them. This is especially the case with updates occurring on a different section of a page than the region where the user is currently interacting with the application. One way to overcome this type of problem is through an explanation of the interaction to the user. Additionally, the provision of a page refresh as in tradiation web pages is also an acceptable addition to enhance accessibility.

Finally, they pointed out how RIAs can be organized into three distinct types as a function of how they are developed or deployed. The first type, plug-in based, involves creating the application on a dedicated platform and then deploying it as either an embedded solution or a standalone application launched from the browser. Examples of this are Flash and Java applications which can be embedded into a web page. The advantages of these types of applications are that they allow developers to have a stable and understood platform for development, and provides assurances that it will run in the same way across multiple platforms, as long as the correct plug-in is available.

The second and by far the most well known type of RIAs are referred to as script-based RIA. These applications make use of standard web technologies such as HTML, CSS, DOM and JavaScript. They use HTML and CSS to style and present the user interface, JS to make asynchronous requests to the server and use DOM scripting to perform on-the-fly rendering. The advantages of this approach are that it gives the application a minimal approach and requires no software to be pre-installed before the application can be used. The disadvantage is that these technologies are typically inconsistent between different browsers and developers must therefore be extra careful to ensure that all users receive the same quality of service.

The third and final type of RIA they discussed is browser-based RIAs that incorporate a user interface language which allows developers to specify the needed elements and their interactions in a declarative format. An example of such a format is the XUL language created by the Mozilla Foundation. The advantages of this approach is the formats are based existing XML standards and are typically platform independent.

2.2 RIAs compared to native applications

Charland and Leroux (2011) discussed the strengths and weaknesses of developing native applications for mobile phones compared to developing rich internet applications. They found that because the programming languages and tools are different for each phone when developing applications natively, developers are required to have a wide varity of different skills. It is also time consuming and mundane to develop the same application multiple times. Conversely, when developing RIAs the developers are typically only required to know HTML, JavaScript and CSS and the code only has to be written once.

However, they found that RIAs have limited support for the software and hardware commonly found on mobile phones, such as touch screens and GPS trackers. They also require the phone to be connected to the internet at all times while running the application. Native applications can access the software and hardware on phones and can be run even when the device has no internet access. The design of native applications also follows a uniform standard and uses user interface controls tailored to the specific device. RIAs often look out of place when compared to native applications.

They also evaluated the latency and execution time of both types of applications and found that RIAs were always slower to launch and slower to respond to user commands. Their conclusion was that the RIA approach has not yet reached the same level of performance as native applications, but the gap between the two is getting smaller, and with time RIAs will become indistinguishable from native experiences.

2.3 The future of HTML

In their book, **Lubbers et al. (2011)** examined what the future of web applications could be. They predicted that within a few years there will be close to zero users browsing the web with legacy browsers that do not implement the HTML5 standard such as Internet Explorer 6. This will allow web developers to make use of HTML5 features such as peer-to-peer video communication between users, support for multimedia streaming in the browser and asynchronous communication between a browser client and a server.

In the more distant future, browsers will be able to display 3D graphics by interacting with a computer's graphical processing unit. This will allow resource intensive games that previously had to be downloaded as native code and installed to be played directly in the browser.

They also discussed how browsers in the future will be able to interact with hardware on phones and other handheld devices. HTML5 is offering APIs for detecting touch events and gestures, the user's location and changes in a device's orientation. This will allow web developers to create even more complex web applications incorporating the user's actions in real life.

Chapter 3

Prototype Specification

In order to evaluate the current state of rich internet application development, it was decided that it would be necessary to develop a realistic web-based application with complex functionality. In cooperation with NAROM, a design document was created for a web-based simulation application called Emissions. This application will be used by students during high school science classes to learn space related concepts. This chapter will describe the design and requirements of Emissions.

3.1 Design

The design of Emissions is closely related to a similar system NAROM commissioned last year, Spaceship Aurora, where students are split into groups and must cooperate with a mission commander to complete a mission. During the mission, several challenges arise that must be solved by the students.

A problem with Spaceship Aurora is that because everyone participating in a mission has be physically present at the same location, only a small number of schools located nearby NAROM's facilities are able to use the system as a teaching device. Emissions will attempt to solve this problem by displaying all mission related information in a web browser and having the students and teacher communicate with the mission commander via an integrated video chat.

The scenario used in Emission is called *Under a Solar Storm* and involves four teams of four students each that are assisting an astronaut in repairing a broken satellite in space. While repairing the satellite, an unexpected solar flare breaks out that increases the radiation in the area to critical levels and panicking the astronaut. The students are responsible for the astronaut's survival and must continuously monitor her condition and give her instructions accordingly.

The first of the student teams is called *the science team* and is responsible for calculating the average radiation level inside the suit of the astronaut. If they decide that the astronaut is exposed to too much radiation to safely complete the mission, they must notify another team called *the security team*.

The second team, called *the astronaut team*, is responsible for monitoring the astronaut's breaths per minute and heart rate. If either of these values increase above the recommended maximum, the team must advise the astronaut to calm down.

The security team must monitor how much oxygen is remaining in the astronaut's oxygen tank and the amount of carbon dioxide inside the astronaut's suit. They must then use this information in addition to the information received from the science and astronaut teams to evaluate if it is safe to continue the mission. If they decide that the radiation values are too high, the astronaut is too stressed, or there is not enough oxygen to complete the satellite repairs, they can tell the astronaut to abort the mission and come back to earth.

The fourth and final of these teams, *the communication team*, is responsible for keeping an open communication channel between earth and the astronaut. This is done by continuously monitoring the current signal strength and switching to a different communication satellite if the signal drops below a threshold where communication becomes difficult. This team is also the only team that will communicate directly with the astronaut and mission commander. If the other teams have any information they wish to relay to either of these people, they must do so through the communication team.

In order to succeed, the students must do well both on a scientific and on a social level. The scientific element involves reading numbers on a graph and doing simple calculations to determine if the astronaut is in any real danger while the humane aspect involves calming the astronaut and telling her to continue working when there is no perceived danger, and if necessary, telling her to retreat back to spacecraft without panicking her.

Combining the soft and hard sciences in a realistic manner is not something that is currently done in the Norwegian education system. If done correctly, NAROM believes that this type of education will motivate students not currently interested in science to take their science classes more seriously as well as teach our next generation to be more compassionate and understanding towards others.

To achieve realism, both the mission commander and the astronaut role were originally going to be played by live actors appearing on a big screen in the classroom while the students were working. The actors would react to events happening in the simulation such as radiation levels increasing or oxygen levels dropping. The astronaut would also respond to and carry out commands given by the students. Unfortunately, this idea was eventually dropped due to budget constraints.

The role of mission commander will be filled by a person from Andøya Rocket Range who will introduce the students to the mission and report mission progress along the way. This person must also be able to step out of character during the mission to help out with technical difficulties or give the students hints on how to proceed if they are stuck. The mission commander can either be physically present in the classroom with the students or communicate with the students by using a video chat.

The astronaut role will be filled by a person appearing on a number of pre-recorded video clips playing in a loop on a computer in the classroom. Each video clip represents one of several states that the astronaut can be in. Examples of states are calm, nervous, panicked, and working. The current state of the astronaut is controlled by the mission commander.

The realism of the simulation hinges on NAROM's ability to provide good video clips and the mission commander's ability to switch between the clips during the mission. It would be preferable that the role of the astronaut is filled by a live actor as creating pre-recorded video for every imaginable scenario is not possible, but it is understandable that they are not willing to spend the extra resources. The current role distribution only requires a single person from ARR to be involved in each simulation which is the absolute minimum.

3.2 Requirements

NAROM created an 8-page document (appendix A) describing in detail all the tasks that the students must complete during a mission. This document was used as a basis for writing the final requirement specification below. The keyword *must* means that the definition is an absolute requirement of the specification, whereas *must not* means that the definition is an absolute prohibition of the specification. The keyword *should* means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course, whereas *should not* means there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label (Bradner, 1997).

1. The science team
   - 1.1. The science team must be able to monitor the astronaut's current radiation exposure.
   - 1.2. The science team must be able to evaluate the severity of the current radiation exposure by calculating the average by hand and inputting it into a field.
     - 1.2.1. If the value entered is above 65 mSv/h, the user must be warned that the astronaut is exposed to dangerously high amounts of radiation.
   - 1.3. The science team must be able to initiate video calls with the security team.
2. The astronaut team
   - 2.1. The astronaut team must be able to monitor the respiration rate of the astronaut.
   - 2.2. The astronaut team must be able to evaluate the severity of the respiration rate by calculating the average oxygen used per minute in percent of the total amount of oxygen in the astronaut's tank and inputting it into a field.
     - 2.2.1. If the value entered is above 1.2%, the user must be warned that the astronaut is using too much oxygen.

2.3. The astronaut team must be able to monitor the heart rate of the astronaut.

2.4. The astronaut must be able to evaluate the severity of the heart rate by calculating the average beats per minute and inputting it into a field.

  2.4.1. If the value entered is above 80, the user must be warned that the astronaut's heart rate is abnormally high.

2.5. The astronaut team must be able to initiate video calls with the security team.

3. The security team

3.1. The security team must be able to monitor the amount of carbon dioxide that the astronaut is breathing in.

3.2. By pressing a button, the security team must be able to command the astronaut to change the carbon dioxide filter.

  3.2.1. Changing the carbon dioxide filter must reset the carbon dioxide amount that the astronaut is breathing into its minimum level.

  3.2.2. Changing the carbon dioxide filter can only be done once per mission.

3.3. The security team must be able to monitor the oxygen remaining in the astronaut's tank relative to the mission time remaining. This should be done by displaying two graphs on a chart with oxygen remaining on the Y-axis and time remaining on the X-axis.

3.4. The security team must be able to initiate video calls with the communication team and the mission commander.

3.5. The security team must be able to receive video calls from the science team, the astronaut team, the communication team and the mission commander.

4. The communication team

4.1. The communication team must be able to view a pre-recorded video of the astronaut when connected to an active satellite.

4.2. Active satellites must periodically become unavailable during the mission.

4.3. There must exist at least two satellites where at least one of them is always active.

4.4. The communication team must be able to connect to an active communication satellite by entering a frequency corresponding to a satellite's frequency into a text box.

4.4.1. Connecting to a communication satellite must take at least two seconds.

4.4.2. While connecting to a new satellite, the astronaut video clip must be unavailable.

4.5. The communication team must be able to initiate video calls with the security team and the mission commander.

4.6. The communication team must be able to receive video calls from the security team and the mission commander.

5. The mission commander

5.1. The mission commander must be able to start a new mission with a mission time defined in minutes.

5.2. The mission commander must be able to change the remaining mission time during a mission.

5.3. The mission commander must be able to change the astronaut video clip viewed by the communication team.

5.4. The mission commander must be able to initiate video calls with the security team and the communication team.

5.5. The mission commander must be able to receive video calls from the security team and the communication team.

6. Video calls

6.1. When a video call is initiated between two teams, the callee must be given the option to either accept the incoming call or hang up. Hanging up must immediately cancel the call and accepting must start the call.

6.2. A team can only participate in one concurrent call.

6.3. A user must be able to hang up a video call at any time.

6.4. If the internet browser of a user participating in a video call crashes, the system must automatically hang up the video call.

6.5. If a team accepts an incoming video call while currently active in another video call, the system must automatically hang up the previous video call before starting the new one.

6.6. The system must support concurrent video calls between different teams, up to a theoretical maximum of two concurrent calls.

A few changes were done to do the original specification in cooperation with NAROM before creating the above requirement specification. Originally the science team had to track the total amount of radiation accumulated in the astronaut's body as well as the current amount of radiation in the atmosphere. It was decided to combine these two variables into a single variable to avoid confusing the user. The user was also supposed to calculate the radiation average by taking exactly four samples over thirty seconds, but it was decided to let the users figure out for themselves how many samples to take and how often to take them when calculating averages. This gives the users more freedom which will probably make the task more fun.

3.3 Use case diagram

An UML use case is an object-oriented modeling construct that is used to define the behavior of a system. Interactions between the user and the system are described through a prototypical course of actions along with a possible set of alternative courses of action (Sengupta and Bhattacharya, 2008). It was decided to model the system requirements as use cases because they are typically easy to understand for people with little technical background, which made it easier to discuss the high level requirements with NAROM.

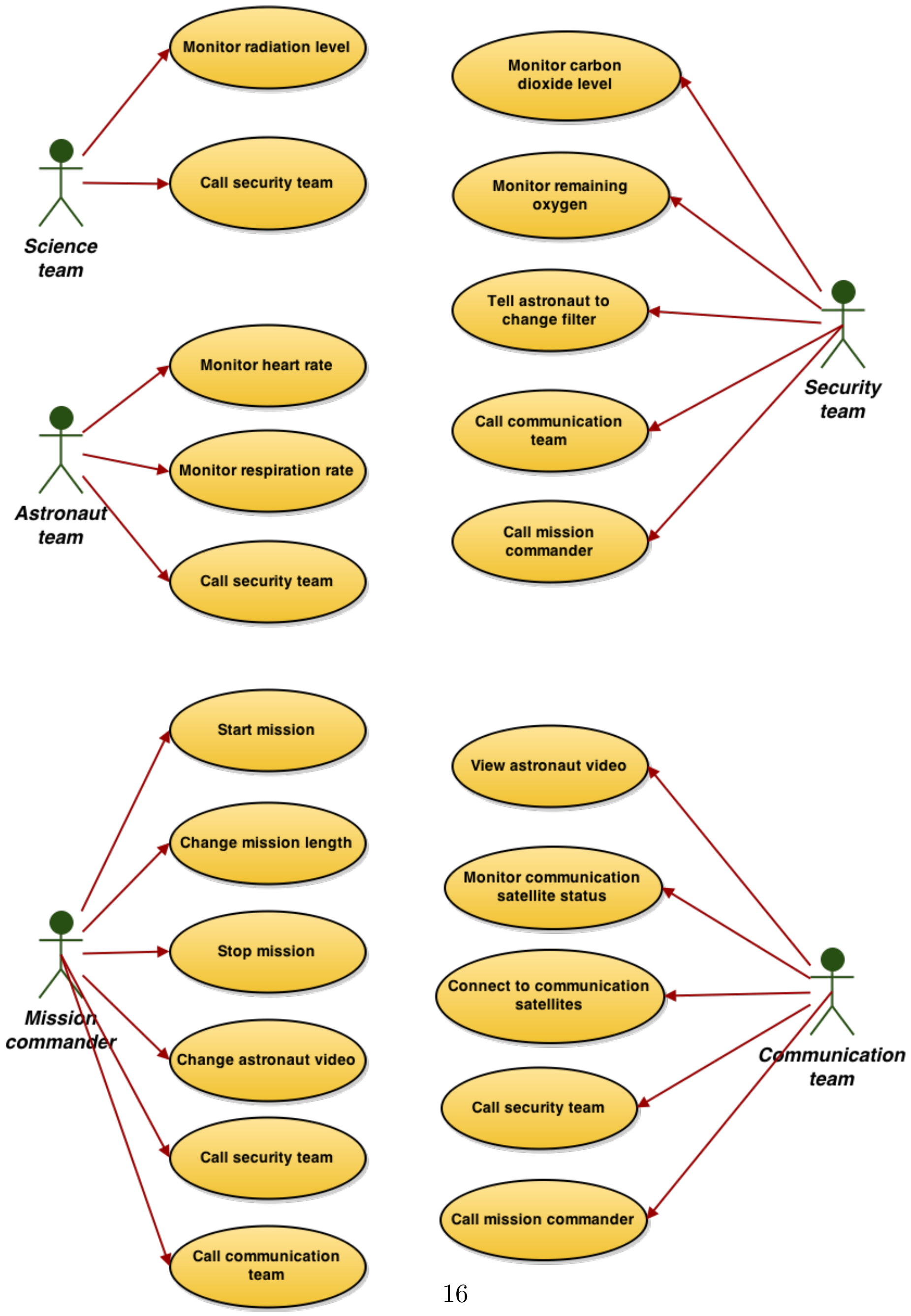


Figure 3.1: The functional requirements modelled as an UML use case diagram

3.4 Non-functional requirements

Non-functional requirements are requirements that are not specifically concerned with the functionality of a system. They place restrictions on the product being developed and the development process, and they specify external constraints that the product must meet (Sommerville and Kotonya, 1998).

3.4.1 Documentation

Because the prototype will be expanded upon by other developers to create a fully functional product, it is important the system is well documented to efficiently introduce new developers to the system so that they can start developing as fast as possible. This thesis and comments in the source code itself will serve as the documentation.

3.4.2 Maintainability

Maintainability is the capability of the software product to be modified. Modifications may include corrections, improvements or adaptations of the software to changes in environment, and in requirements and functional specification (ISO/IEC 9126, 2001).

Because a similar system has never been created before, Emissions will likely go through multiple iterations to figure out the optimal settings. It is therefore important that the system can be modified to fit different scenarios. NAROM has also requested that they must be able to modify certain parameters of the simulation themselves without the need for developer assistance.

3.4.3 Portability

Portability refers to the practice of guaranteeing that a software program will behave similarly when run on two different computers (Brown, 1979). In the context of rich internet applications, portability refers to an application's ability to behave as intended when run in different web browser implementations.

Because the application will run on the devices of students whose preferred choice of web browser is varied and unknown, it is important that Emissions is able to function on a wide variety of browser implementations.

3.4.4 Reliability

Reliability is a software's ability to withstand and recover from component, or environmental, failure (ISO/IEC 9126, 2001). Because sessions in which the system will be used are short and planned out weeks in advance, it is important that the system is reliable enough to be used to completion without delays or failure.

3.4.5 Usability

Usability refers to the ease of use of a given function. The ability to learn how to use a system (learnability) is also a major subcharacteristic of usability (ISO/IEC 9126, 2001). It is important that the system is intuitive enough to offer a short learning curve for eighth graders that have never used a similar system before. Because the prototype developed for this thesis will focus on the technical details as opposed to the system design and look and feel of the final product, its usability will not be evaluated.

Chapter 4

Technology Evaluation

This chapter will give an overview of the technology that was considered for creating the Emission prototype.

4.1 Skype

An important system requirement is that the mission commander must be able to communicate with the students using a microphone and a web camera. This can be done with the popular peer-to-peer VoIP application *Skype* developed by KaZaa in 2003 that allows its users to place voice calls and send messages to other users of Skype clients (Baset and Schulzrinne, 2004). However, this application is not web-based and does not offer a web-based plugin and can therefore not be smoothly integrated into a rich internet application.

An alternative was discussed where the required video chat functionality of Emissions would be delegated to Skype and not integrated into the web application itself. Delegating the video chat functionality to a third party application would ensure that the video chat is always available and would save server costs related to sending audio and video data between clients. It would also reduce complexity during development as implementing video chat functionality from scratch is not a trivial task.

However, this was ultimately decided against as assisting each group of students participating in a mission with installing and setting up Skype would probably take up an unacceptable amount of time. There are also issues with simultaneously participating in a Skype call and viewing a web page on a handheld device.

4.2 Adobe Flash

Adobe Flash started out as a drawing application for pen computers in the early 1990s. Today it is a popular software platform used for creating animated movies, games and rich internet applications, and remains a popular choice for web developers because of its large installed user base. It is most commonly used for serving streaming video and advertisements on web pages.

An advantage of using Flash is that it is a well-known technology that is well-supported in all popular browsers available on a desktop computer provided that the user downloads the Adobe Flash Player plug-in. This plug-in will most likely already be installed on a user's computer because it is required to view the popular video streaming website Youtube.com.

In addition, using Flash guarantees that the application will behave similarly across multiple platforms. This reduces the need to test the application thoroughly on multiple different platforms and makes it future-proof, meaning that maintenance will be easier. Implementing video chat functionality is also relatively straight forward when using Flash, as you can rely on the functions already implemented by the Flash Media Server (Sanders, 2008).

A drawback of using Flash is that it is not supported on Apple devices such as iPads, iPods and iPhones. From the inception of their iPhone, Apple decided to not include Flash Player in their iOS operating system because it is a proprietary software with many security vulnerabilities and the number one reason Apple Macs crash (Jobs, 2010).

4.3 Java

Java is a software platform released in 1996 that like Adobe Flash promises "write once, run anywhere" capabilities, meaning that software written in Java should run on any platform without requiring recompilation or additional code.

However, an implementation of the Java Virtual Machine is not available for a number of handheld devices, among them the Apple iPhone, Apple iPad and Windows Phone (Bloice et al., 2009). This means that a web application written in Java cannot be accessed when using a handheld device, which is a big drawback.

Many Norwegians will be familiar with Java from paying bills online as all Norwegian online banks until recently required their users to log in using a Java applet. Frequent problems with updating the Java Runtime Environment and articles published in major newspapers regarding security vulnerabilities in Java has given the software platform somewhat of a bad reputation in Norway, reaching a climax when Aftenposten, one of the largest newspapers in Norway, in 2013 recommended all their readers to deactivate Java in their browsers (Nilsen, 2013).

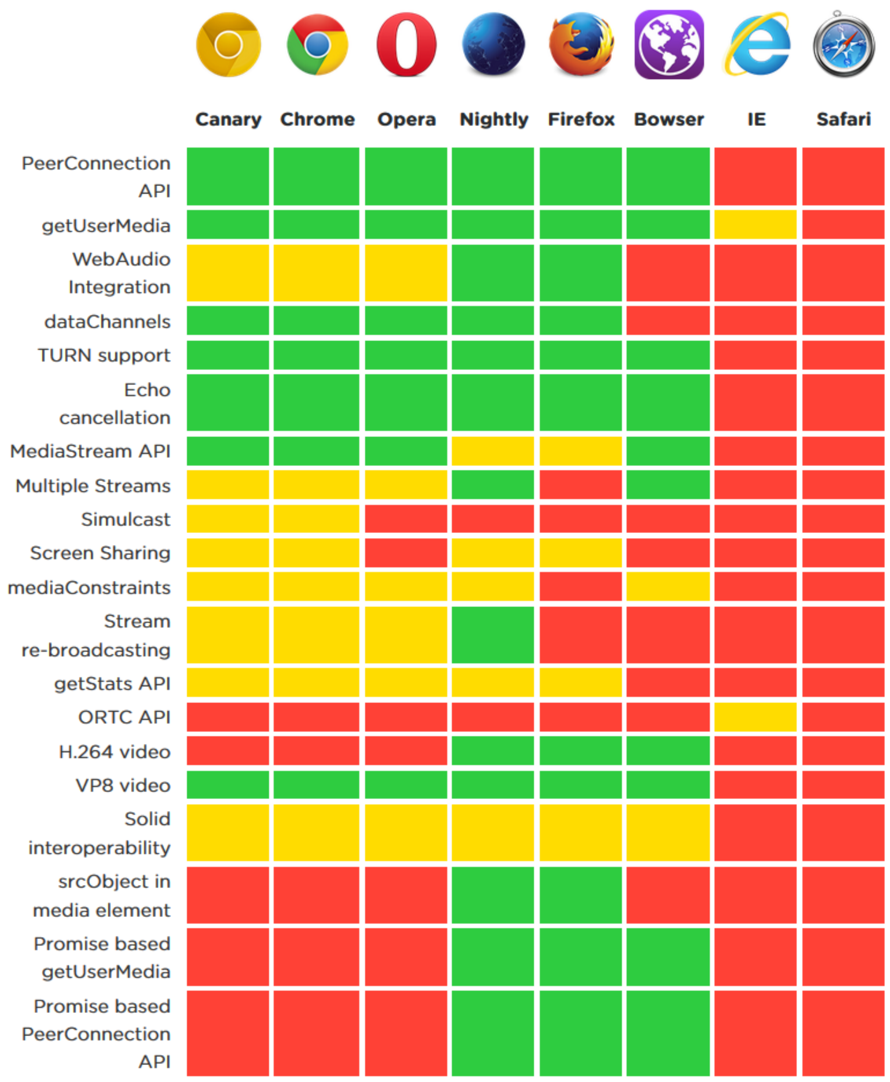
4.4 HTML5

HTML is the markup language used to structure web pages. The fifth revision of the language, called HTML5, was finalized in October 2014 and is considered a very recent technology. One of the main objectives of this revision was to allow web developers to create interactive and multimedia-rich web applications without having to rely on third party plug-ins such as Flash and Java. Browser makers have been quick to implement the most important features of HTML5 and users quick to update. According to jwplayer.com, 98% of the internet's population was using a browser that implements the HTML5 video element in March 2015.

4.5 WebRTC

WebRTC, short for web real time communication, is an API part of the new HTML5 standard defined by W3C that enables browser-based, peer-to-peer voice communication, video chat and file sharing (Bergkvist et al., 2012). The main advantage of using WebRTC technology is user convenience as it requires no downloads, plug-ins or installs to be used. Another advantage is that the multimedia streams are not distributed by a central server, but by the peers themselves. This means that hosting a WebRTC call requires little to no bandwidth, which makes it cheap. The fact that the stream does not need to go through a central server also means that the path between peers is shorter, which reduces audio and video latency.

The drawback of using WebRTC is that it is a new technology in the beta stage that is not fully implemented by any browser. Recent versions of Google Chrome and Mozilla Firefox have implemented enough of the API to make it usable for video conferencing, but Internet Explorer and Safari currently do not support any features of WebRTC. The figure below displays what features of WebRTC are being supported by which browsers. A green square means the the feature is fully implemented and working, a yellow square means that the feature is available to some degree while a red square means that it completely unsupported.



	Canary	Chrome	Opera	Nightly	Firefox	Bowser	IE	Safari
PeerConnection API	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥
getUserMedia	🟩	🟩	🟩	🟩	🟩	🟩	🟨	🟥
WebAudio Integration	🟨	🟨	🟨	🟩	🟩	🟥	🟥	🟥
dataChannels	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟥
TURN support	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥
Echo cancellation	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥
MediaStream API	🟩	🟩	🟩	🟨	🟨	🟩	🟥	🟥
Multiple Streams	🟨	🟨	🟨	🟩	🟥	🟩	🟥	🟥
Simulcast	🟨	🟨	🟥	🟥	🟥	🟥	🟥	🟥
Screen Sharing	🟨	🟨	🟥	🟨	🟨	🟥	🟥	🟥
mediaConstraints	🟨	🟨	🟨	🟨	🟥	🟨	🟥	🟥
Stream re-broadcasting	🟨	🟨	🟨	🟩	🟥	🟥	🟥	🟥
getStats API	🟨	🟨	🟨	🟨	🟨	🟥	🟥	🟥
ORTC API	🟥	🟥	🟥	🟥	🟥	🟥	🟨	🟥
H.264 video	🟥	🟥	🟥	🟩	🟩	🟩	🟥	🟥
VP8 video	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥
Solid interoperability	🟨	🟨	🟨	🟨	🟨	🟨	🟥	🟥
srcObject in media element	🟥	🟥	🟥	🟩	🟩	🟥	🟥	🟥
Promise based getUserMedia	🟥	🟥	🟥	🟩	🟩	🟩	🟥	🟥
Promise based PeerConnection API	🟥	🟥	🟥	🟩	🟩	🟩	🟥	🟥

Figure 4.1: Current WebRTC support by browser as of April 2015 as reported by www.iswebrtcreadyyet.com

4.6 Node.js

Because the application is web-based, a centralized server is required to be online during a simulation to serve the web content to the clients. The server must also remember the current state of the simulation in case a user closes the browser window in the middle of the simulation and wants to get back in. The server will also function as a WebRTC signalling server. Although video and audio is sent directly between clients when using WebRTC, a centralized signalling server is still required to introduce the peers that will be communicating to each other.

Node.js is a popular server framework invented in 2009 by Ryan Dahl. An advantage of using this framework is that the server-side and client-side logic of the application will be written in the same language, specifically JavaScript. This makes the server-side code more accessible for front-end web developers, which gives NAROM more flexibility when finding someone to finish or modify the system.

JavaScript was until recently only used for client-side user interaction within the browser and considered unsuited for server-side usage because of its inherent tardiness of being a non-compiled language. Node.js remedies this by using the Google initiated, open source V8 engine to compile JavaScript into highly optimized server-side machine code (Dahl, 2012). This makes Node.js consistently outperform both the traditional Apache framework and the more recent Ruby on Rails framework in high concurrency, low data throughput environments (McCune, 2011).

4.7 Summary

It was decided that the prototype will be created as a script-based RIA, meaning that only standard web technologies such as HTML, CSS, DOM and JavaScript will be used. This approach was chosen because it requires no software such as browser plug-ins to be pre-installed before the application can be used. This is important because installing software takes time, a resource there will be little of when a mission must be started and finished within the short duration of class.

Another advantage is that limiting the implementation to standard web technologies will allow users of handheld devices such as iPhones and certain Android phones that do not support plug-in based web applications to use the system. The drawback of a script-based approach is that the application is likely to behave inconsistently in different browsers, which means that it will be important to test the application thoroughly in all popular browsers.

WebRTC, an API part of the recent HTML5 standard, will be used to enable video calls between the students and the mission commander. The advantage of using this technology is that all data is transferred peer-to-peer, which eliminates the need for an expensive server to distribute the multimedia streams. The drawback of WebRTC is that it is not supported by all browsers, which means that fallback solutions may have to be provided.

All server-side logic will be implementing using the Node.js framework. This framework was chosen because it outperforms similar server frameworks and is easier to learn for developers only familiar with JavaScript. The drawback of using this framework is that it is a relatively new framework. This means that it has undergone less testing than older frameworks, which may indicate that it less stable. It also means that the number of available libraries is fewer and that the API is more likely to change.

Chapter 5

Implementation

This chapter will explain how the prototype was implemented.

5.1 The server

The server implementation can be found in server.js. The server has four responsibilities:

1. Serve static web content such as HTML and JavaScript.
2. Initialize WebRTC connections between peers.
3. Keep track of simulation time and notify connected clients when timed events occur.
4. Send the current state of the simulation to clients connecting after the simulation has started.

Serving static content is done by using Express.js, a Node.js web application framework that simplifies creating a web server. The line below is all the code required to make the files in the "public" folder available to a user visiting the web site hosting the Node.js server.

```
app.use(express.static(__dirname + '/public'));
```

Messages are sent between clients by using the socket.io library that enables realtime, bi-directional messages to be sent between client and server. For example, to initiate a WebRTC call, a client sends a "call" message to the server along with another string describing the recipient, for example "security" if the intended callee is the security team. The security team client then responds with a WebRTC signal containing all the necessary information to set up the call, which the server simply relays to the caller. This initialization procedure will be described in more detail in the chapter covering the WebRTC implementation.

The server recognizes many other message types. When it receives a "start mission" message along with an integer describing the mission time in milliseconds, the server initializes all simulation-related variables such as radiation level and oxygen remaining to their default values and sends a "mission started" message to all clients along with the mission length.

Some simulation variables are defined on the server as ranges. As an example, the astronaut's heart rate can be either "high", which is between 90 and 120 beats per minute, "medium", which is between 60 and 80 beats per minute, or "low", which is between 40 and 60 beats per minute. It is up to the clients to decide on the precise value at any given time, as long as the value lies within the range dictated by the server.

This system was created because in order for the simulation to be realistic, variables such as the heart rate and radiation levels cannot be static for long periods of time. In fact, they should change on a second-to-second basis. Initially, this was achieved by deciding the exact value on the server-side and then distributing it to the clients. However, this turned out to create a lot of unnecessary server-to-client traffic and caused the server to require more than the expected amount of processing power. Because it is important for NAROM that the server is cheap to run, it now sends ranges instead, which reduces both the bandwidth and processing power requirements to negligible amounts.

At predetermined points in a mission, timed events will occur that change the current range of simulation variables. The server is responsible for notifying the clients when an event occurs. This is done by the "updateRanges" function that periodically compares the current mission time to time of the scheduled events and if an event is due, sends a message to all clients describing the event type and which variables have changed.

It was requested by NAROM that they must be able to change the timing and effects of events themselves without developer assistance to make tweaks to the simulation based on class size and experience. It is therefore important that all events are defined in a human-readable way. JavaScript Object Notation (JSON) structures were chosen to represent events. JSON is a lightweight, text-based, language-independent data interchange format derived from the JavaScript standard. It defines a small set of formatting rules for a portable representation of structured data (Crockford, 2006). This format is easy to read and modify for people without programming experience and can be natively parsed by JavaScript without the need to write a custom parser.

All time-based events are defined in the events.json file. An event is defined as a tuplet where the first element is an integer that represents when the event should occur in seconds and the second element describes the effect of the event. All events are contained within another array that describes the event type. For example, the array describing all events related to the heart rate variable looks like this in JSON:

```
"heartRateEvents": [
        [0, "low"],
        [40, "high"],
        [60, "normal"],
        [90, "high"],
        [130, "low"]
],
```

This means that at 0 seconds into the simulation (i.e. as soon as the simulation begins), the astronaut's heart rate should be low. Then at 40 seconds into the simulation the heart rate should change from low to high and so forth. The events.js also contains another type of structure that defines the ranges of a variable. For example, the structure defining the exact values for the heart rate levels looks like this in JSON:

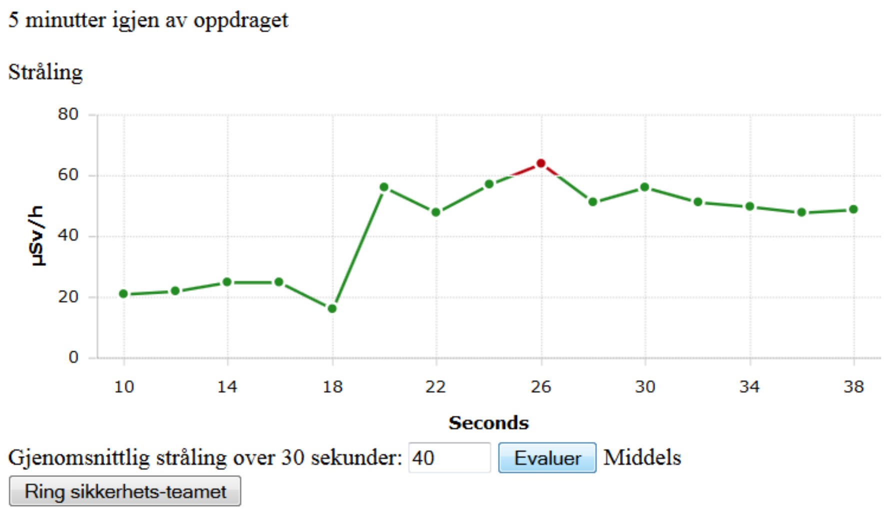
```
"heartRateLevels": {
        "low": [40, 60],
        "normal": [60, 80],
        "high": [90, 120]
},
```

When the server starts up, the events.json file is read and the events within it are congregated into a single array sorted by the time they occur. This makes it easy to only run the event loop when it is known that an event will occur instead of periodically polling and checking for events, which saves processor cycles on the server side.

The server's final responsibility is to provide a way for clients connecting after the simulation has started to receive the current simulation state. The server keeps track of the current simulation state in an array called "ranges". Clients can access this array by sending a "get ranges" message to the server.

When the mission time expires or when a "stop mission" message is sent to the server, the server emits a "mission stopped" message to all clients, marks all events as incomplete and resets all variables to their default values.

5.2 The science team screen


5 minutter igjen av oppdraget

Stråling

Gjenomsnittlig stråling over 30 sekunder: 40 Evaluer Middels

Ring sikkerhets-teamet

Figure 5.1: The science team interface

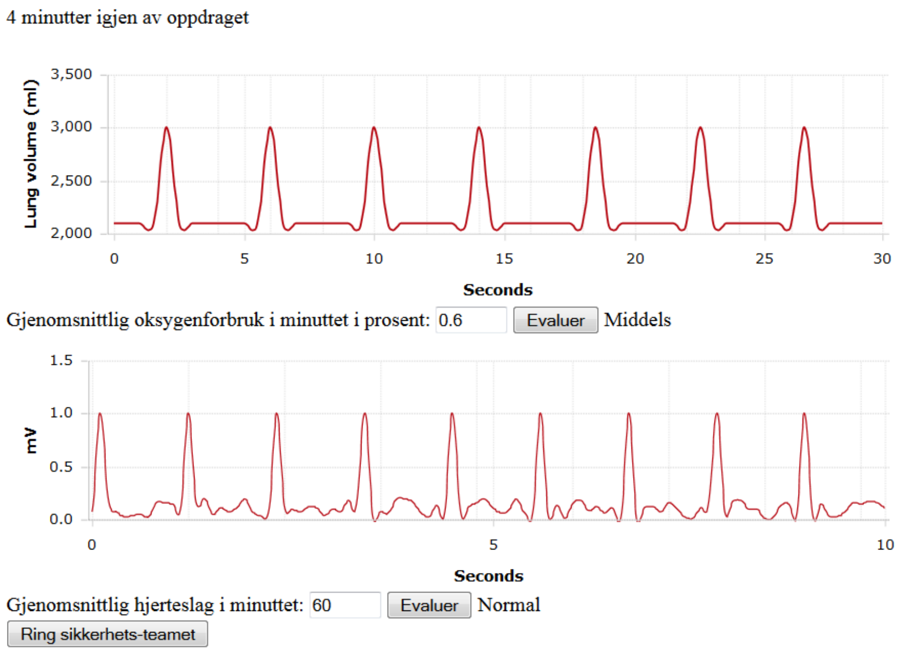
The interface that the science team will use contains a mission timer, a chart displaying the current radiation, an input field where the average radiation can be evaluated by the system and a button to call the security team. The relevant code can be found in the science.html and science.js files.

The animated chart is created using AmCharts, a charting library for JavaScript. It is assumed that the developers that will create the graphical user interface for Emissions will want replace this graph implementation with their own custom graph library, so the AmCharts library has been made easy to replace. All the radiation samples are inserted into a standard JavaScript array, so the only code that references the library directly is contained within the AmCharts.ready listener function.

When the client receives the "mission started" message from the server, the chart is displayed and the event loop is started. The event loop inserts a new radiation sample into the chart every 2 seconds. The sample value is a random integer within the range dictated by the server. The server can change the current range by sending the “change radiation” along with a tuple containing the new minimum and maximum values.

When a number is entered into the evaluation field and the button is pressed, a string is displayed that describes the radiation severity. For example, an average radiation of 40 is considered medium. The thresholds for the various radiation severities are not hardcoded, but received from the server by sending the “get levels” message. The values returned are read by the server from the events.json file.

5.3 The astronaut team screen


4 minutter igjen av oppdraget

Gjenomsnittlig oksygenforbruk i minuttet i prosent: 0.6 Evaluer Middels

Gjenomsnittlig hjerteslag i minuttet: 60 Evaluer Normal

Figure 5.2: The astronaut team interface

The interface that the astronaut team will use contains a mission timer, a chart displaying the astronaut's respiration rate, a chart displaying the astronaut's heart rate, input fields where the average respiration and heart rates can be evaluated by the system and a button to call the security team. The relevant code can be found in the astronaut.html and astronaut.js files.

It is important for NAROM that the charts used in the system are similar to charts used in the real world. Hence, instead of simply plotting the heart rates and respiration rates on a chart, they wanted the students to be able to figure out the respiration rate and heart rate from looking at a pneumograph and a electrocardiograph (ECG) respectively.

A pneumograph is an instrument commonly used in the medical profession to measure a patient's respiratory movements (Kraman, 1988). The instrument plots a patient's current lung volume on a chart. Since the patient's lung volume increases each time he breathes in, a student can approximate the number of respirations per minute by counting the number of graph peaks in ten seconds and multiplying it by six.

The pneumograph used by the system displays the astronaut's respiratory movements over 30 seconds. The chart contains two samples for each second, meaning that the maximum numbers of respirations per minute the system supports is 120. Adults with a respiration rate of more than 24 breaths per minute are likely to be critically ill (Cretikos et al., 2008), so this limit should be reasonable.

The chart is created by first figuring out the number of respirations per minute by picking a random integer within the range decided by the server and then creating a corresponding number of peaks. For example, if the respiration rate is decided to be 14, the value of every 14th sample in the chart is increased.

An electrocardiogram (ECG) is a test used by medical professionals to record and analyze the electrical currents produced by the human heart muscle (Stuchl, 1988). These tests are done by devices called electrocardiographs that model the electrical currents as a graph on a sheet of paper or on a computer screen. The resulting graph contains much information related to the state of the heart muscle that can be used to diagnose a disturbed heart rhythm among other things, but can also be used without training to figure out a person's heart rate by counting the peaks.

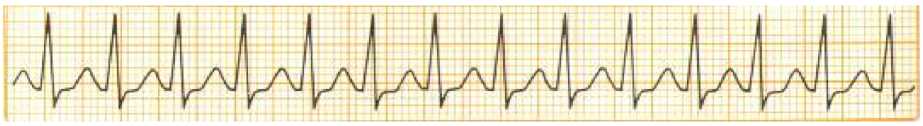


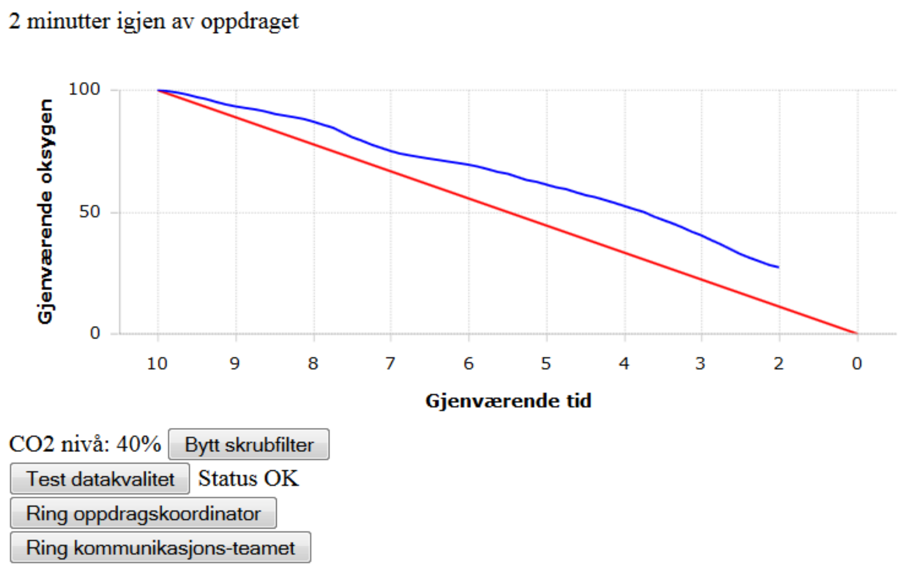
Figure 5.3: ECG of a real patient taken from Northern Arizona University

Mimicking an ECG proved more difficult than mimicking a pneumograph because an ECG contains more information than just the heart rate. After experimenting with different techniques, a somewhat convincing graph was created by filling the space between the peaks with randomly generated samples of low values. Because the ECG will only be used to figure out the astronaut's heart rate, the actual value of the samples in between the peaks do not matter.

The graph contains ten samples every second which allows a maximum heart rate of 600, far above the maximum ever recorded. Initially each heart rate sample was created on demand, however this turned out to require more processing power than my web browser allowed because of the random number generation. This was solved by pre-rendering 200 samples and inserting them into a buffer array. The event loop takes one sample from the buffer ten times per second to create an animated graph. The graph loops after 20 seconds, but this was not noticeable in tests. A new buffer of samples is generated when the heart rate is changed by the server.

The students are expected to evaluate the heart rate by inputting the average value into the text box. This average value is calculated by counting the number of peaks on the ECG and multiplying the result by six. The average respiratory rate must however be inputted as average percentage of the available oxygen consumed per minute. To arrive at this value, the students must divide the number of respirations per minute by 50, because 50 respirations amounts to 1% oxygen usage.

5.4 The security team screen


2 minutter igjen av oppdraget

CO2 nivå: 40%

Status OK

Figure 5.4: The security team interface

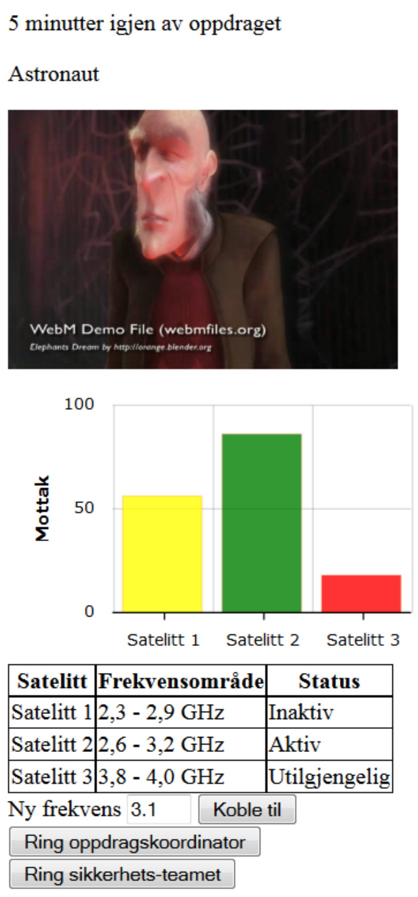
The interface that the security team will use contains a mission timer, a chart with a blue graph displaying the oxygen remaining in the astronaut's oxygen tank together with a red linear graph displaying mission time remaining for comparison, a string displaying how much CO2 the astronaut is currently breathing, a button to tell the astronaut to change the CO2 scrub filter, a button to test the satellite reception and buttons to call the mission commander and communication team.

The graph is updated once a minute to show how much oxygen remains. If the blue line dips below the red line, the astronaut will not have enough oxygen to complete the mission unless the oxygen usage is reduced. The remaining oxygen is calculated by subtracting a random number within the oxygen usage range from the current oxygen storage.

The CO2 scrub filter should be changed when the CO2 level is above 25%. Changing the filter resets the CO2 level to zero, but can only be done once per mission, so timing is important. The rate of which the CO2 level increase is static and solely based on mission time. At the end of a mission, the CO2 level will be 50% if the filter has not been changed. If the filter is changed exactly halfway through the mission, the CO2 level will never increase above 25%.

The buttons for testing the satellite reception and data quality are hidden until the mission commander reports that the astronaut has successfully repaired the satellite. The reception test will always succeed and the data quality test will always fail on the first try, but succeed when performed a second time.

5.5 The communication team screen


5 minutter igjen av oppdraget

Astronaut

Satelitt	Frekvensområde	Status
Satelitt 1	2,3 - 2,9 GHz	Inaktiv
Satelitt 2	2,6 - 3,2 GHz	Aktiv
Satelitt 3	3,8 - 4,0 GHz	Utilgjengelig

Ny frekvens 3.1

Figure 5.5: The communication team interface

The interface that the communication team will use contains a mission timer, a video feed displaying the astronaut, a chart displaying the reception of the communication satellites, a table displaying the frequency of each communication satellite, an input field and button to connect to another satellite and buttons to call the mission commander and security team. The relevant code can be found in the communication.html and communication.js files.

The video feed is a HTML5 video element that can play video files encoded in the WebM format. The video clip automatically loops and multimedia controls such as pausing and seeking are hidden to give the impression that it is a live feed. The currently playing video clip is decided by the mission commander.

If the communication team is not connected to a receptive communication satellite, the video feed will be blacked out. By default, the team will be connected to satellite number 1, but the reception quality of all the satellites will change every two seconds to a random percentage within a range dictated by the server. If the reception quality of a satellite drops below 35%, the team will be automatically disconnected from that satellite and must connect to a new one.

To connect to a new satellite, the team must look up the frequency range of a receptive satellite in the table and input a value within that range. Connecting to a new satellite takes five seconds. When the connection is established, the video element reactivates.

5.6 The mission commander screen

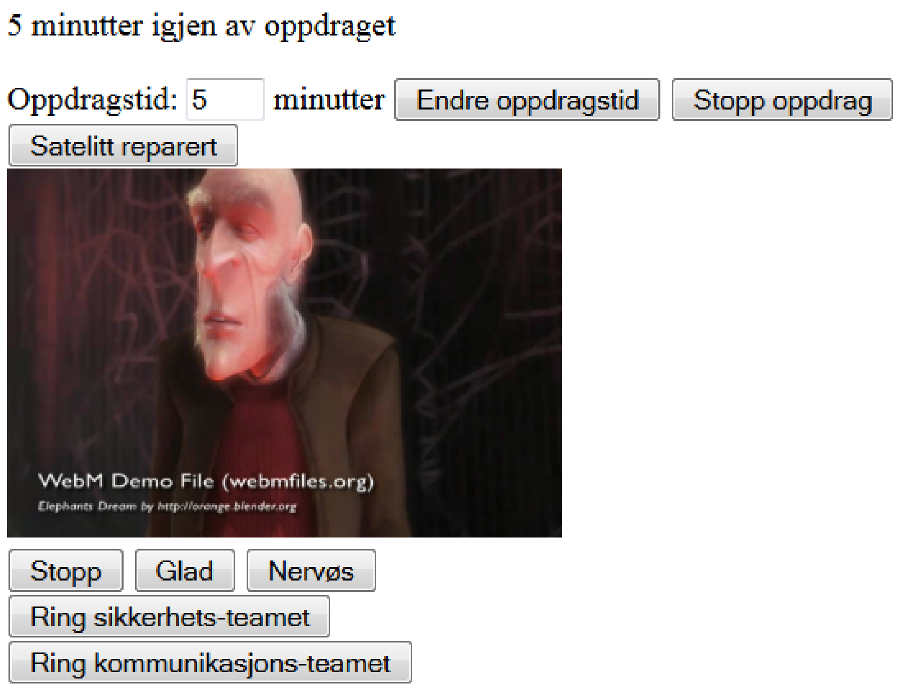

5 minutter igjen av oppdraget
Oppdragstid: 5 minutter
Endre oppdragstid
Stopp oppdrag
Satelitt reparert
WebM Demo File (webmfiles.org)
Elephants Dream by http://orange.blender.org
Stopp
Glad
Nervøs
Ring sikkerhets-teamet
Ring kommunikasjons-teamet

Figure 5.6: The mission commander interface

The mission commander interface is meant to be used by the person hosting the simulation. This will typically be an adult located at Andøya Rocket Range. The interface contains buttons for starting and stopping the mission, changing the mission length and marking the mission as completed. It also displays the astronaut video clip currently playing on the communication team client and contains buttons for changing or stopping the video. It is also possible for the mission commander to start a video call with either the security team or the communication team. The mission commander will most likely always be in an active video call with the communication team because he will not usually be physically present at the location where the simulation is taking place.

When the “start mission” button is pressed, a “start mission” message is sent to the server that will notify all clients that a new mission has started. When the mission commander client receives confirmation by the server that the mission was started, the “start mission” button is replaced by a “stop mission” button and the “change mission time” and “mission complete” buttons are added to the interface.

The intention of the “satellite repaired” button is to notify the clients that the main goal of the mission has been completed, which is to repair the satellite. This notification is only used by security team client which will be able to test the satellite signal. The mission will automatically stop when the mission time is zero, but the mission commander can also stop the mission prematurely by pressing the “stop mission” button.

When the astronaut video is changed by pressing one of the buttons below the video, a “change video” message is sent to the server along with an URL pointing to a WebM file. This signal will be relayed by the server to the communication team client which will immediately start streaming from the specified URL.

5.7 WebRTC

To hide the complexity involved in initializing WebRTC connections a wrapper function called “RTC.connect” was created and can be found in WebRTC.js file. This function takes five inputs: The ID of the caller and callee teams, a socket object connected to the server, and the HTML5 video elements that will display the local and remote video feeds. The function returns an object that represents a WebRTC connection. This object has five functions: Call, answer, disconnect, onCallStarted and onCalledEnded.

The call function sends a call message to the callee client by using the open server socket. When the callee receives this message, the user can either hang up or answer. If he decides to answer, the “RTC.connect” function will be called on the callee client followed by the answer function being called on the resulting connection object.

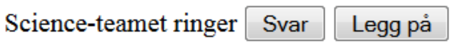


Figure 5.7: An incoming call as displayed on the bottom of the security team screen

The answer function first requests access to the user’s web camera and microphone with the “navigator.getUserMedia” function. Then, it creates a connection offer specified using the Session Description Protocol (SDP), a protocol intended for describing multimedia session metadata such as transport addresses and media options (Handley et al., 2006). This offer is sent back to the caller client, which automatically responds with an answer message also specified using SDP. The initialization phase is then complete and the video call starts. When either user closes the browser window or clicks the “hang up” button, the disconnect function is called that closes the peer connection and fires the onCallEnded listener on both clients which hides the video elements.

If either client is behind a firewall using network address translation (NAT), initializing a WebRTC connection between two peers requires more work. NAT functions by dividing a network of computers into stub domains. Every computer within a domain will appear to have the same IP address to computers outside the domain, although this is not really the case (Srisuresh and Egevang, 2001). This system is commonly used for security reasons and to get around the limited number of IPv4 addresses as IP addresses within a stub domain can be reused.

The problem with NAT in relation to WebRTC is that a peer-to-peer connection cannot be established without knowing the private IP address of the callee's computer. WebRTC can get around this by using STUN servers. Session Traversal Utilities for NAT (STUN) is a protocol that provides a means for an endpoint to determine the IP address and port allocated by a NAT that corresponds to its private IP address and port (Rosenberg et al., 2008). A STUN server takes a request containing a public IP address and responds with a private IP.

In some cases, STUN is unable to find the private IP address of a peer making it impossible for that peer to communicate directly with other peers. In these situations, WebRTC can use an intermediate TURN server acting as a communication relay. Traversal Using Relays around NAT (TURN) is a protocol that allows the host to control the operation of the relay and to exchange packets with its peers using the relay (Mahy et al., 2010). A TURN server should only be used as an absolute last resort as all the peer-to-peer benefits of WebRTC are lost.

Most of the work related to using STUN and TURN is handled by the browser, but the developer must still provide the address of at least one STUN server and one TURN server if he wishes his WebRTC implementation to make use of these features. To ensure that the servers are available when needed, developers using WebRTC are encouraged to host their own STUN and TURN servers. Since this was out of the scope of this thesis, the prototype uses a STUN server provided for free by Google and a TURN server provided for free by Viagénie. There is no guarantee that these servers will be operational when needed.

Chapter 6

Evaluation

This chapter will present the results from the evaluation of the Emissions prototype.

6.1 Functional requirements

The functional requirements of the prototype were evaluated by manually testing the availability of each requirement. It was found that the prototype implements all of the functional requirements correctly.

6.2 Non-functional requirements

Evaluating the non-functional requirements was more difficult as their evaluation is usually subjective, for example can two developers disagree on the quality of a piece of source code.

6.2.1 Portability

The prototype was run in different web browsers commonly found in schools to evaluate its portability. The results are shown in the table below.

Web Browser	**Operating System**	**Hardware**	**Result**
Internet Explorer 10	Windows 8	Dell laptop	Fail
Google Chrome 42	Windows 8	Dell laptop	Pass
Mozilla Firefox 39	Windows 8	Dell laptop	Pass
Safari 8	OS X Yosemite	iMac	Fail
Google Chrome 42	OS X Yosemite	iMac	Pass
Mobile Safari 5	iOS 5.1	iPhone 4S	Fail
Mobile Safari 8	iOS 8.3	iPad Air	Fail
Google Chrome 33	Android 4.4	Galaxy S5	Pass
Google Chrome 38	Android 5	Nexus 9	Pass

Table 6.1: Prototype portability

The prototype functioned correctly in all browsers except for video calling and the astronaut video, which only worked on Google Chrome and Mozilla Firefox on laptop computers and Google Chrome on the Android phone. Because video calling is an essential part of the application, the prototype cannot be considered portable enough to be used effortlessly in a school setting.

6.2.2 Maintainability

The documentation and maintainability of the prototype were evaluated by a fellow master student. After reading a short document describing the architecture of the prototype he was able to make modifications to the functionality of the application and add new functionality in a reasonable amount of time. He felt that the source code was well documented and easy to follow.

6.2.3 Reliability

Evaluating the reliability of an application is difficult as it typically requires many testers using the application for long periods of time. The more testers, the more accurate the evaluation. The reliability of the prototype was tested by using the application in one hour sessions three times. The application showed no signs of instability during testing, but because of the lack of testing the prototype cannot yet be considered completely reliable.

Chapter 7

Conclusion and Further Work

7.1 Conclusion

The goal of this thesis was to investigate and evaluate the technology that exists today for creating rich internet applications and build a prototype for Emissions, a web-based simulation tool for learning that will be used by Norwegian eighth graders to learn about space-related activities. A requirement specification and use case diagram for Emissions was created in cooperation with NAROM. The specification highlighted the importance of making the application maintainable, reliable and portable.

A technology evaluation was done that covered Flash, Java, HTML5 and Node.js, the most popular tools that exist today for creating rich internet applications. HTML5 and Node.js were selected for the implementation of Emissions due to their portability, ease of implementation and user convenience. The client-side of Emissions was created in HTML5 and WebRTC while the server-side was created in Node.js.

During evaluation it was found that the prototype successfully implements the requirement specification and fulfills the non-functional goals of maintainability, documentation and reliability, but cannot be considered portable enough to be used in a classroom where the hardware of students is unknown, but is known to include devices such as phones, tablets and laptops.

The lack of portability is mostly due to the video calling functionality which is implementing by using WebRTC, a part of the HTML5 standard that is still a work in progress and only functionally implemented in browsers and devices created by Google or Mozilla.

As of today, this issue cannot be resolved as no technology exists that can implement cross-platform video chatting functionality in a browser. A fallback solution based on Adobe Flash could potentially enable video chatting in Internet Explorer and Safari browsers, but browsers that support neither Flash or WebRTC, such as Mobile Safari on recent iPhones, offer no way to enable video chat support.

Additionally, a literature review found that RIAs cannot access hardware of handheld devices such as touch screens, GPS trackers and microphones, and were always slower to launch and slower to respond to user commands than native applications. Another limitation is that the device must always be connected to the internet while a RIA is running.

Because of these limitations the technology that exists today cannot be considered sufficient for creating cross-platform rich internet application with the same functionality, performance and user convenience as native applications.

However, it cannot be ignored that internet applications developed today have access to a lot more functionality than those developed ten years ago. With features announced for the next iteration of the HTML standard such as 3D rendering in the browser and better access to hardware, it is certainly conceivable that web applications will become indistinguishable from native applications within a few years time and that the development of native applications will become a thing of the past for most development houses.

7.2 Further work

Much work must be done to the Emissions prototype before it can be used in a real scenario.

- **Create a graphical user interface** - Because the goal of the Emissions prototype was to implements its functional requirements to evaluate the limitations of rich internet applications, the overall design as well as the look and feel of the application was ignored. Before the application can be used in a real scenario, a user interface must be designed and integrated with the functionality of the prototype.
- **Evaluate usability** - After the user interface is created, usability tests should be performed to uncover faults in the design. The user interface should then be changed according to the results.
- **Include missing video clips** - The prototype is currently using placeholder video files when displaying the astronaut. More realistic video clips of an astronaut should be created and integrated into the application.
- **Improve portability** - The multimedia and video chatting functionalities of the application only work when the prototype is accessed with a Mozilla Firefox or Google Chrome browser. In order for the multimedia players to work in other browsers, the video files must be transcoded to the mp4 format and provided as a fallback in case the web browser is incapable of decoding WebM-encoded video files.

  To make video calling work in browsers that do not support WebRTC, a fallback solution relying on Adobe Flash or a downloadable WebRTC plug-in must be provided. Providing video chat support for devices that support neither WebRTC or Flash, such as the Apple iPad, will not be possible unless a native application is created.
- **Work on deployment** - The application requires an operational Node.js server to function. Steps have been taken during the implementation of the prototype to ensure that the usage of server bandwidth is minimal, so an expensive solution should not be required.
- **Provide reliable STUN and TURN** - The prototype relies on free STUN and TURN servers provided by Google and Viagénie in order for video calls to function if the users are behind firewalls. These servers may cease to provide their services for free at any time, which will cause the

video chat functionality to fail in some cases. To be absolutely certain that video chatting will function in all cases, private STUN and TURN servers should be either rented or provided by the system administrator.

- **More testing -** Very little testing was done to evaluate some aspects of the prototype. The reliability and usability of the prototype should be evaluated by performing many usability tests in realistic scenarios with eight graders and a mission commander from NAROM.

Bibliography

Baset, S. A. and Schulzrinne, H. (2004). An analysis of the skype peer-to-peer internet telephony protocol. *arXiv preprint cs/0412017*.

Bergkvist, A., Burnett, D. C., Jennings, C., and Narayanan, A. (2012). Webrtc 1.0: Real-time communication between browsers. *Working draft, W3C*, 91.

Bloice, M. D., Wotawa, F., and Holzinger, A. (2009). Java's alternatives and the limitations of java when writing cross-platform applications for mobile devices in the medical domain. In *Information Technology Interfaces, 2009. ITI'09. Proceedings of the ITI 2009 31st International Conference on*, pages 47–54. IEEE.

Bradner, S. (1997). Key words for use in rfcs to indicate requirement levels.

Brown, P. J. (1979). *Software portability.* CUP Archive.

Busch, M. and Koch, N. (2009). Rich internet applications. *UML-Based web engineering, Technical Report*, 902.

Charland, A. and Leroux, B. (2011). Mobile application development: web vs. native. *Communications of the ACM*, 54(5):49–53.

Cretikos, M. A., Bellomo, R., Hillman, K., Chen, J., Finfer, S., and Flabouris, A. (2008). Respiratory rate: the neglected vital sign. *Medical Journal of Australia*, 188(11):657.

Crockford, D. (2006). The application/json media type for javascript object notation (json).

Dahl, R. (2012). Node. js: Evented i/o for v8 javascript.

Farrell, J. and Nezlek, G. S. (2007). Rich internet applications the next stage of application development. In *Information Technology Interfaces, 2007. ITI 2007. 29th International Conference on*, pages 413–418. IEEE.

Handley, M., Perkins, C., and Jacobson, V. (2006). Sdp: session description protocol.

ISO/IEC 9126 (2001). Software engineering – product quality. Standard, International Organization for Standardization, Geneva, CH.

Jobs, S. (2010). Thoughts on flash. `https://www.apple.com/hotnews/thoughts-on-flash/`. Online; accessed 25-May-2015.

Kraman, S. S. (1988). Simple capsule pneumograph. US Patent 4,732,159.

Lubbers, P., Albers, B., Salim, F., and Pye, T. (2011). *Pro HTML5 programming*. Springer.

Mahy, R., Matthews, P., and Rosenberg, J. (2010). Traversal using relays around nat (turn): Relay extensions to session traversal utilities for nat (stun). *Internet Request for Comments*.

McCune, R. R. (2011). Node. js paradigms and benchmarks. *STRIEGEL, GRAD OS F*, 11.

Nilsen, J. (2013). Du er nødt til å deaktivere java. `http://www.aftenposten.no/digital_old/nyheter/-Du-er-nodt-til-a-deaktivere-Java-7089466.html`. Online; accessed 25-May-2015.

Rosenberg, J., Mahy, R., Matthews, P., and Wing, D. (2008). Session traversal utilities for nat (stun). Technical report, RFC 5389 (Proposed Standard).

Sanders, W. (2008). *Learning flash media server 3.* " O'Reilly Media, Inc.".

Sengupta, S. and Bhattacharya, S. (2008). Formalization of uml diagrams and their consistency verification: Az notation based approach. In *Proceedings of the 1st India software engineering conference*, pages 151–152. ACM.

Sommerville, I. and Kotonya, G. (1998). *Requirements engineering: processes and techniques.* John Wiley & Sons, Inc.

Srisuresh, P. and Egevang, K. (2001). Traditional ip network address translator (traditional nat).

Stuchl, R. J. (1988). Ekg monitoring system. US Patent 4,754,762.

Appendix A

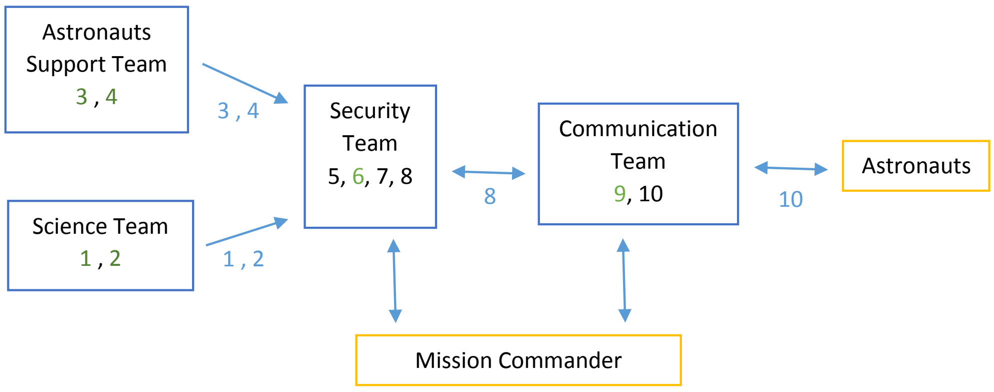
Below is the 8-page document received from NAROM describing in detail all the tasks that the students must complete during a mission when using the Emissions application. The requirement specification used for creating the prototype is based on this document.

E-Mission: "Under a Solar Storm"

Information diagram, groups and definition of activities

Information diagram:

The next graph shows the flux of information between the different groups. It also contains the activities each group does. These are specified with a number inside the group box. The activities description can be found below. The number is also indicated over a flux information arrow if an activity generates a result that must be communicated to another group. An activity that partially requires external tasks, that is, a calculation external to the GUI, is indicated in green.



Groups:

Each group consists in up to 4 members. The definition of the activities includes who is doing what. However, this is still possible to do small adjustments in the future. In general, member 1 is designed as the one who takes control of the computer, that is, who uses the GUI. Members 2 and 3 will have extra assignments, like calculation of values using external calculators to be entered afterwards in the computer. Member 4 is in general designed to be the one who assist member 1, and communicate values to other groups.

Definition of activities:

In this section it is described the activities for each group. It is described what the GUIs must contain, the complementary activities in the group (without using GUIs), the distribution of the activities and how often the members of the group must proceed with these activities. The external tasks, that is, the complementary tasks done without using the computers are described in green. Finally, between brackets, after naming the activity, it is shown a number that is used as reference in the graphs.

Science team (ScT)

- Monitor of the radiation level (1):

  - Action: monitor the radiation level. Includes lecture of the sampling of values periodically, to do the average of these value, include the result again in the GUI. Action to be done each 5 minutes.

  - How:
    - The member of the team responsible of the computer (member 1) must proceed the initialization of the procedure of taking samples of radiation level. A button of initialization should be used. After this action, a monitor for taking 4 samples values during 30 s is used. As guide take a look to the pag 48 of the "Spaceship Aurora Screen Reference"

    - Member 2 of the team proceed reading the values and making the average with an external calculator. They inform back this result. External task: using a calculator calculate the average value for the radiation using as input the 4 values that appears in the GUI.

    - Member 1 introduce this value. A new graph appears showing the radiation status: a bar is shown in green if it is ok, orange if it could be dangerous or red if it is too high.

    - Member 4 must indicate to SeT if the radiation level is orange: in this case the situation can be dangerous if it does not improve. He/she must inform immediately in case of red, or three oranges in a raw because then the situation is very dangerous for the mission.

- Accumulation of radiation in the body (2):

  - Action: monitor the amount of radiation that it is accumulated in the body of the astronaut.

  - How:
    - External task: Member 3, using external calculator, proceeds adding the radiation accumulation level each time the radiation level is measured (each 5 minutes). Green values from the radiation level do not count, orange values will count as 15, red levels as 50. This is an accumulated value. The radiation level in the body is serious if it reaches 50, and very serious if it reaches 75.

    - Member 4 must indicate to ScT if the radiation level in the body is serious (over 50) and, specially, if it is very serious (over 75). This last case might demand to stop the

<u>Astronauts support team (AST)</u>

- Monitor breath rate (3):
  - Action: monitor the breath rate and, based on that, have an estimation of the oxygen left. Action to be done each 5 minutes
  - How:
    - GUI must have an initialization button for start recording Steigen respiration rate and a graph displaying the respiration output rhythm (kind of medical type, see as example figure 1 below) during 20 seconds.
    - Member 1 must initialize and record the respiration rate. When it is finished (after 15 seconds), he/she counts the number of inspirations.
    - <u>External task:</u> Based on the number of inspirations in 15 seconds, member 2 must calculate (external calculator) the extrapolation to 60 seconds because this is the value to input, and say the result to member 1
    - <u>External task:</u> Based on result given by member 2 (number of inspirations in 60 s), member 3 calculates (external calculator) the amount of oxygen consumed in the last minute for this respiration rate. To be done we know that in a normal situation (respiration rate equal 25 times in a minute) the amount of oxygen consumed is 0.5% of the tank.
    - Member 1 set this latest value in the GUI and a graphics updating the amount of oxygen left is displayed (in percent). It is also shown the percent consumed in the last minute with the latest respiration rate. If that is over 0.75% it must be shown in red (alert)
    - If the respiration rate is high and, therefore, the consumption of oxygen goes over 0.75%, member 4 must inform asap to SeT . Steigen must be informed to calm down..

- Monitor hearth rate (4):
  - Action: monitor the breath rate and, based on that, give indications to SeT. Action to be checked every 10 minutes
  - How:
    - GUI must show an initialization button for start recording Steigen hearth rate

- A graph (cardiogram) is shown during 10 seconds (see, as example, graph 1 below). Member 1 must indicate the number of pulsations in this 10 seconds

- External task: Member 3 must calculate the amount of pulsations in 60 seconds with the calculator.

- Member 1 set the result that will appear in green if it is below 90, in orange if it is below 120, and in red if it is over 120.

- Member 4 must indicate SeT to ask Steigen to relax, or even stop for a moment the work, if the pulsations are over 120.

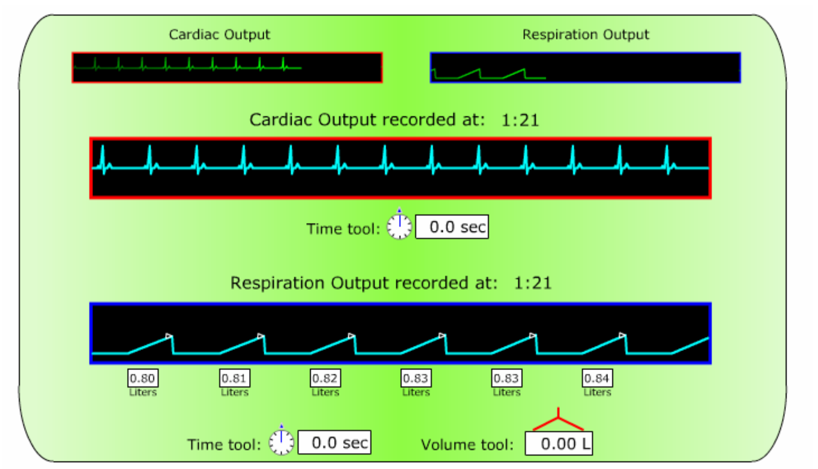


Figure1: Example of graphs showing the hearth rate and respiration rate.

Security team (SeT)

- Carbon dioxide level (5):
  - Action: monitor the amount of Carbon dioxide level and ask to the astronaut to change scrubber (filter) if it is necessary. This action will be necessary to do only one time during the mission.
  - How:
    - GUI must have a monitor of the carbon dioxide level and a monitor showing the scrubber (filter) selected. During the mission the carbon dioxide level increase over security limits. This can be indicated with an alarm and red color. Based on pag 42 of the “Spaceship Aurora Screen Reference”.

    - Member 1 asks member 4 to indicate the problem to CT to inform Steigen to change scrubber (filter). This change is updated in GUI.
    - Member 1 must monitor if levels drops

- Monitor oxygen left in tank (6):
  - Action: to monitor the oxygen in the tank of the astronaut; to estimate time left if the respiration rate increases too much.
  - How:
    - GUI display a graphic showing the oxygen level (in percent). Every 10 minutes, member 1 must read the value and set it in a input field in the GUI “Current oxygen level”.
    - After entering this value, a graph is updated displaying Oxygen remaining vs mission time. The value must follow a line for being sure that everything is ok (see figure 2 below as example). If the new value is very much below this line the consumption of oxygen goes very fast. This graph is an estimation for a normal respiration rate.
    - GUI also presents a number with the total percent of oxygen left. For a normal respiration rate the average of consumption is around 0.5% of the tank per minute.
    - SeT will be informed by AST if the respiration rate goes over 0.75% consumption of the tank/minute. External task: Member 2 will proceed to calculate (external calculator) the time left if the astronaut goes on breathing at this rate. For this he/she knows from the graph the remaining time (total time minus the time already spent in the mission) in case of consuming 0.5% per minute.
    - GUI has a field to set in this value. The graph showing Oxygen remaining vs mission time will be update with a new line for this higher consumption level. A value for the mission time left for this case is indicated as well.

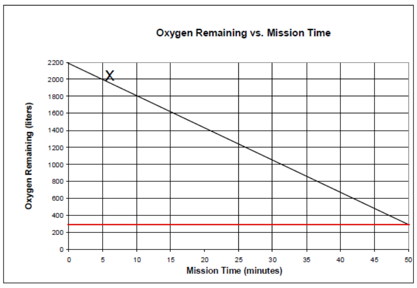


Figure 2: Example graph indicating Oxygen remaining vs Mission time

- Check Communication status with Tyr-V (7):
    - Action: checking the status of the communication with Tyr-V (satellite astronaut Alexandra Steigen is repairing). This is done after Alexandra confirms that she has finished the job.
    - How:
        - Checking starts when Steigen confirms that she has replaced the transmitter. GUI has a button showing for example “Test Tyr5 data reception”. During 10 seconds or so it is displayed a sequence of “1s” and “0s” arriving, for example a box with 11000101000110101…. After this 10 seconds an “OK” status is displayed.
        - After completing the data reception test, member 1 proceeds checking the data quality. GUI contains a similar button for this purpose and graph indicating process status. The first time it is checked the output is fail. The second time is ok.
        - During the first checking, when the quality process fails, SeT must inform MC who repeats and confirms the action
        - A button showing “Send to safe mode” is also present but deactivated. At the end of the mission, as soon as the quality procedure is ok, the button turns to activated, and member 1 must sent the satellite to safe mode asap.

- Receiving and analyzing information status decide how to proceed (8):
    - Action: SeT will receive information status from AST and ScT (1, 2, 3, 4), and together with their own results (5, 6, 7) will decide what to do. That will means mainly to indicate to the CT what information to say to the astronaut. In parallel, they consult with the MC the decisions.
    - How:

        - Information from ScT: member 4 of the ScT will come to inform if radiation level is too high, dangerous, or if the radiation already accumulated in the body is dangerous. In all cases, previous consulting MC, a recommendation is done and Alexandra must be informed. The script will indicate the type of recommendation that must be done (MC will drive a little the situation) during the different stages of the mission.

        - Information from AST: member 4 of the AST will come to inform if respiration rate is high or the hearth rate is high. In the first case SeT must proceed calculating the time left with the new respiration rate. SeT inform MC and decide if it is necessary to inform Alexandra to calm down, stop working for a moment, or even cancel the EVA. The script will indicate the type of recommendation that must be done (MC will drive a little the situation) during the different stages of the mission.

        - Additional information as output from tasks 5, 6, 7 can demand to inform the astronaut.

        - Member 4 indicates to CT specifically and precisely what to transmit to the astronaut.

<u>Communication team (CT)</u>

- Keep communication with the Spaceship Aurora and Astronauts (9)

    - Action: Keep an active open channel for communication with the spaceship aurora and the astronauts. Change communication satellite if it is necessary. This action must be done each 10 minutes

    - How:
        - GUI must give indication on the status of the communication link. For example, an icon showing a satellite with different colors depending on the communication status is suggested: green > ok, orange > weak signal, red > no signal. The signal will be lost when the communication satellite is out of range.

        - In case of weak signal (orange), member 1 must select a new satellite from a list that is shown in the GUI. The selected satellite must be on-line and in-range. An example is shown in page 41 of "Spaceship Aurora Screen Reference"

        - Same procedure if the communication goes to red, but in this case asap, because all connections with the mission is lost in the different groups.

        - When the new satellite is selected, GUI will display the possible frequencies to stablish the connection. It will be an interval different for each satellite, for example between 2.3 and 2.9 GHz. External task: member 2 must select the best frequency for communication, and this is the one in the middle of the interval. Note: in general, the satellite will use S channel that is between 2 and 4 GHz. Member 2 said the result back to member. GUI has an field to set the channel and button to start communication.

- Communication with astronaut (10):
    - Action: Inform the astronaut about the recommendations from Earth (8). Receive astronaut's status and news.
    - How:
        - GUI must have an area dedicated for videoconference when contacting the astronaut. A button for starting communication and other for stopping communication are present. Member 1 click the starting communication button to contact the astronaut, give the indications, ask for feedback, and close the communications.
        - A button normally in grey indicating "EVA incoming call" will be flashing in red and with sound if the astronaut wants to communicate. Again, member 1 proceed as in the previous point.
        - Member 2 and 3 will receive indications from SeT (member 4) with the information to communicate to the astronaut. They can also get assist from the MC. The information to say would be due to different circumstances and are concrete down in the script. MC will help to drive the situation.