Project Fingerpaint

SRD-0.1

Software Requirements Document

Authors:
Tessa Belder (0739377)
Lasse Blaauwbroek (0749928)
Thom Castermans (0739808)
Roel van Happen (0751614)
Benjamin van der Hoeven (0758975)
Femke Jansen (0741948)
Hugo Snel (0657700)

Junior Management:
Simon Burg
Areti Paziourou
Luc de Smet

Senior Management:
Mark van den Brand, MF 7.096
Lou Somers, MF 7.145

Technical Advisor:
Ion Barosan, MF 7.082

Customer:
Patrick Anderson, GEM-Z 4.137

Eindhoven - June 23, 2013

Abstract

This document describes the Software Requirements of FINGERPAINT. This project is part of the Software Engineering Project (2IP35) and is one of the assignments at Eindhoven University of Technology. The Software Requirements Document (SRD) is based on the ESA standard for software development, as set by the European Space Agency (ESA) [1]. This document presents a specification of solutions for the requirements described in the URD [2].

Contents

Document Status Sheet

Document Status Overview

General

Document title:	Software Requirements Document
Identification:	SRD-0.1
Author:	Tessa Belder, Thom Castermans, Benjamin van der Hoeven, Roel van Happen, Femke Jansen
Document status:	Internally approved

Document History

Version	*Date*	*Author*	*Reason of change*
0.0	21-May-2013	Tessa Belder, Benjamin van der Hoeven, Roel van Happen, Femke Jansen, Thom Castermans	Initial version.
0.1	23-May-2013	Roel van Happen,Thom Castermans	Revision after feedback from the technical advisor.

Document Change Records Since Previous Issue

General

Date:	21-May-2013
Document title:	Software Requirements Document
Identification:	SRD-0.1

Changes

Page	*Paragraph*	*Reason to change*
10	2.7	updated the figure reference so it correctly refers to figure 2.1.
22	3.1.7	Rephrased SQR80-84 for better clarity.
7	2.5	Rephrasing section 2.5.
8	2.7	Rephrased tier to instance for client machines.
9	2.7.2	Rephrased that the HTTP server is responsible for communicating simulations rather than executing them.
13-14	3.1.6	Moved and rephrased the first subsubsection (Set options for a mixing protocol) to the define mixing protocol subsubsection.
13	3.1.6	Added a new bit of text for drawing at the beginning of the subsection.
25	3.1.8	Rephrased the first two sentences of the 'Requirements regarding the presentation of results' subsubsection.

Chapter 1

Introduction

This chapter lists general information about this document.

1.1 Purpose

This document provides a translation of all the user requirements listed in section 3 of the URD [2]. Although the URD describes the wishes of the client, the goal of the SRD is to represent the developers' view of what the FINGERPAINT application must be able to do. Note that the software requirements listed in this document are implementation-independent: that is, the requirements describe *what* FINGERPAINT must do, but not *how* the requirements will be realized. The requirements are modelled in a logical model, which provides a simplified view of the content and behaviour of the application.

1.2 Scope

FINGERPAINT is an application designed and developed by Group Fingerpaint for prof. dr. ir. P.D. Anderson. The application provides a cross-platform tool to visualise fluid mixing. Users can define the initial concentration distribution, as well as manipulate the mixing protocol. The resulting fluid distribution can be stored by the user on their device, for later reference.

1.3 List of definitions and abbreviations

1.3.1 Definitions

Client Prof.dr.ir. P.D. Anderson.

Firefox A web browser developed by Mozilla.

Google Chrome A web browser developed by Google.

Internet Explorer A web browser developed by Microsoft.

iOS A mobile operating system developed by Apple.

iOS Safari A web browser developed by Apple designed for devices running iOS.

iPhone A line of smartphones developed by Apple.

iPad A line of tablet computers developed by Apple.

Opera A web browser developed by Opera Software.

Safari A web browser developed by Apple.

System administrator A person who is employed to maintain and operate a computer system and/or network. After the SEP project has been completed, this person will be responsible for maintaining the FINGERPAINT application.

1.3.2 Abbreviations

2IP35	The Software Engineering Project
ADD	Architectural Design Document
CM	Configuration Manager
GUI	Graphical User Interface
SEP	Software Engineering Project
SR	Software Requirements
SRD	Software Requirements Document
TU/e	Eindhoven University of Technology
URD	User Requirements Document

1.4 List of references

[1] ESA, *ESA Software Engineering Standards*. ESA, March 1995.

[2] Group Fingerpaint, "User requirements document," *SEP*, 2013.

[3] prof.dr.ir. P.D. Anderson, "prof.dr.ir. P.D. Anderson's homepage." `http://www.mate.tue.nl/mate/showemp.php/19`. [Online; accessed 1-May-2013].

[4] Group Fingerpaint, "Architectural design document," *SEP*, 2013.

[5] COLEY consulting, "Moscow prioritisation." `http://www.coleyconsulting.co.uk/moscow.htm`. [Online; accessed 24-April-2013].

1.5 Overview

The remainder of this document describe the software requirements in more detail. Chapter 2 gives a general description of:

- relation to current projects (2.1);
- relation to predecessor and successor projects (2.2);
- function and purpose (2.3);
- environment (2.4);

- relation to other systems (2.5);
- general constraints (2.6);
- model description (2.7).

Chapter 3 gives a detailed description of the functional requirements of the system in 3.1 and a list of non-functional requirements is given in 3.2. The requirements traceability matrix is described in chapter 4.

Chapter 2

General Description

In this chapter we discuss the relation of this project to the "outside world": if there are any related projects running currently and if there were related projects in the past. Then, the purpose of the FINGERPAINT application and the environment in which it operates are discussed. After that, its relation to other systems is covered. Finally, some general constraints are described and a description of the logical model is given.

2.1 Relation to current projects

No other current projects are related to FINGERPAINT.

2.2 Relation to predecessor and successor projects

FINGERPAINT has multiple predecessor projects. These projects resulted in multiple Matlab[1] tools that are available on the client's web page [3]. FINGERPAINT will combine some of the functionality of these tools into a mobile web application. FINGERPAINT will be developed in such a way that the client can easily extend the application with new mixers. When the development of FINGERPAINT is complete, FINGERPAINT is no longer responsible for the application after the final deliverable produced in the SEP project. This means that the client may change, add or remove the application's functionality.

2.3 Function and purpose

FINGERPAINT is an application that serves as an educational tool for anyone who wants to gain a deeper understanding of the process of mixing in general, and in particular for students at the TU/e. By interacting with the application, users can quickly and easily find out what the effects of a certain mixer and mixing protocol on an initial distribution are. The user can thus obtain a better understanding about the way this mixer functions. FINGERPAINT may also be used as a quick and convenient way to observe whether a mixing protocol renders good or bad mixing results.

[1]`http://www.mathworks.nl/products/matlab/`

2.4 Environment

Fingerpaint is a web application that is developed primarily for use on mobile devices. This means the application will mostly be accessed through web browsers on smartphones and tablets. It is expected that the application will mostly be used on iPhones and iPads. Therefore, Fingerpaint must support iOS Safari version 6.0 and above. Furthermore, Fingerpaint should support Firefox version 20 and above, and Google Chrome version 26 and above. Lastly, if time permits, Fingerpaint could also support Internet Explorer version 10 and above, Opera version 12.1 and above, and Safari version 6.0 and above.

To support the significant share of smartphones and tablets that run on Android, Fingerpaint should run on devices running on Android version 4.0 and higher. Lastly, if time permits, Fingerpaint could also run on devices running on Windows 8.

The hardware used by the users must be able to run at least one of the supported operating systems and browsers. Also, the application obviously works better on screens that have a diagonal of at least about 4 inches and a resolution of at least about 540x960 pixels - the larger the screen, the easier it is to draw on it (up to a certain maximum, about the size of a desktop monitor with a diagonal of 20 inches). This is because the user draws with their finger and since fingers have a certain size, the screen should be large enough to comfortably draw and see what has been drawn at the same time. On the other hand, when the screen is too big, it costs too much effort to fill up large parts of the mixer.

The application must support the following screen resolutions:

- “Phone portrait”: 540x960 pixels;
- “Phone landscape”: 960x540 pixels;
- “Tablet portrait”: 800x1280 pixels;
- “Tablet landscape”: 1280x800 pixels;
- “Desktop”: 1600x900 pixels.

2.5 Relation to other systems

The Fingerpaint application is an independent system. However, just like every other web application, it is dependent in some way on the browser. That is, the correct (according to the HTML standard[2]) rendering of the web page is done by the browser. Of course, the application will be tested in multiple browsers and built in such a way that it displays correctly in as many browsers as possible.

2.6 General constraints

The user interface should be suitable for mobile devices, so it will be easy to share the visualised results with other people, and to quickly try out new ideas for mixers wherever the user may be.

[2] `http://www.whatwg.org/`

We assume that the server can compute the displacement of fluids reasonably fast[3], so the mixing run can be executed quickly. When the new concentration distribution has been computed, this concentration distribution is sent back to the client device along with a metric to indicate the performance of the mixer. The results are then visualised on the client's device.

As we do not want to be locked to one specific type of device, we have chosen to design a cross-platform solution. While this means that desktop PCs should also be able to run the application, we do not actively support such devices. We will instead concentrate on mobile devices.

It should be possible to save mixing runs on the client device for later reference. For each saved run, we store the initial distribution, the mixer and protocol used, the resulting fluid distribution and the resulting performance metric.

2.7 Model description

On a very high level, the architecture of the Fingerpaint application can be divided into different tiers communicating through communication channels. A graphical representation of the relation between tiers and channels can be found in figure 2.1.

2.7.1 Client tier

Of the Client tier, there are an arbitrary number of instances. These instances are the physical machines of the users of the Fingerpaint application (phones, tablets, laptops, desktops).

Client Browser and Client Persistence

Each Client instance runs a Client Browser (one of the browsers as specified in section 2.4) and has a persistent storage facility (Client Persistence). The Client Browser can use this facility to store data that is specific to the user and does not need to be stored in a central location. The Client Browser provides the user with a Graphical User Interface. All interactions of the user with the application are performed through this GUI.

2.7.2 Application Server tier

The Application Server is a physical machine maintained by the system administrator of the application. It is used to distribute the application and provide services to the application.

HTTP Server

The application is distributed on demand to the Client instances using HTTP by the HTTP server. The HTTP server is a piece of software the responds to requests by either serving a file from a static collection of content or responding with a dynamically build response. The former is used to serve the actual application, while the latter is used when the client requests a simulation, which goes through the HTTP server as well.

[3]Refer to chapter 6 of the ADD [4] for exact requirements.

Application Service

Whenever centralised data or a simulation is required by the application running on a Client, this is handled by the Application Service in response to a request from the HTTP server. This service is responsible for communicating any simulations and their results and also for serving data from the persistent storage. Data from the persistent storage can be a list of mixers for example.

2.7.3 Application Persistence tier

The Application Service may use a global persistent storage facility in order to store data that needs to be available to all Clients. This storage facility is provided in the Application Persistence tier and is communicated to through the Application Persistence Communication channel. This tier may be on a different physical machine than the Application Server, but in practice, it will likely run on the same hardware.

2.7.4 Simulator Server

The simulations that need to be done for the application run on a dedicated machine although in practice, this will likely be the same physical machine as the Application Server.

Simulator Service and Fortran Module

Whenever a Client wishes to run a simulation, it interfaces with the Simulator Service indirectly through the HTTP server, Application Service and Simulator Service Communication channel. The Simulator Service uses an existing Fortran Module to calculate the result of a simulation.

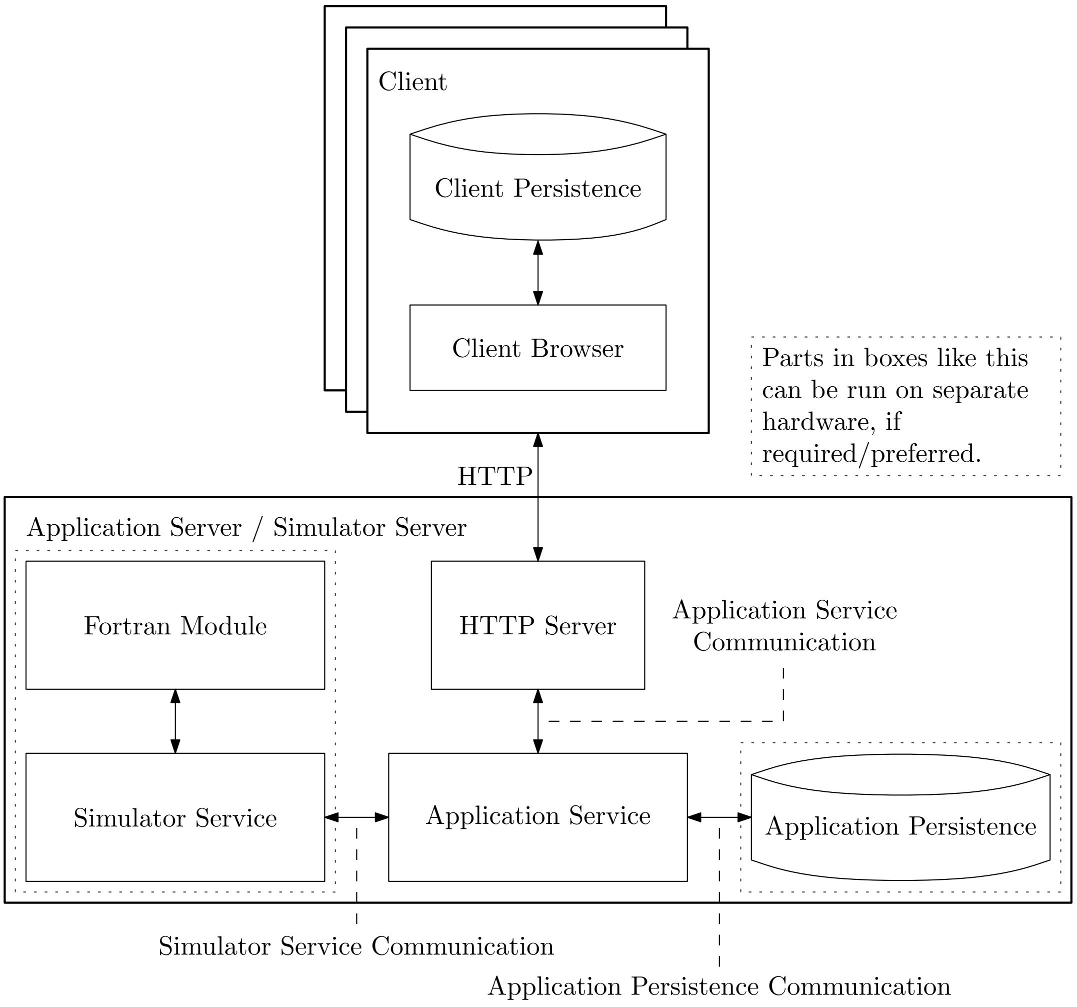
Client
Client Persistence
Client Browser
Parts in boxes like this can be run on separate hardware, if required/preferred.
HTTP
Application Server / Simulator Server
Fortran Module
HTTP Server
Application Service Communication
Simulator Service
Application Service
Application Persistence
Simulator Service Communication
Application Persistence Communication

Figure 2.1: The different tiers of the system

Chapter 3

Specific requirements

This chapter lists all specific software requirements of the application to be developed, both functional and non-functional requirements. The requirements are categorised according to the interface they belong to, as described in section 2.7.

Each requirement has a specific priority, based on the MoSCoW model [5]:

- *must have*; requirements with this priority are essential for the product, and must be implemented.
- *should have*; requirements with this priority are not essential for the product to work. However, they are nearly as important as the *must have*'s and are therefore expected to be implemented.
- *could have*; requirements with this priority are a nice addition to the product, and may be implemented, if time and budget allow this.
- *won't have*; requirements with this priority will not be implemented in this version of the product, but may be nice to implement in future versions.

Only those user requirements from the URD [2] with a priority higher than *won't have* will be translated to software requirements in this chapter.

In some of the requirements in this section, the term "fast enough" is mentioned: for a precise definition, refer to chapter 6 of the ADD [4].

Note that as Fingerpaint application is developed using GWT, some of the software requirements listed in this chapter are described in terms of specific GWT widgets or panels[1].

3.1 Functional requirements

This section lists the functional requirements for the Fingerpaint application.

[1]The website for the GWT widgetlist can be found at `https://developers.google.com/web-toolkit/doc/latest/RefWidgetGallery`.

3.1.1 HTTP Server

SRQ1 *must have*
The HTTP Server must be able to serve files to the Client Browser.

3.1.2 Application Service

SRQ2 *must have*
The Application Service must be able to retrieve all the available mixers from the Application Persistence.

SRQ3 *must have*
The Application Service must be able to send all the available mixers to the HTTP server which in turn should send it to the Client Browser.

SRQ4 *must have*
The Application Service must allow for adding new mixers to the Application Persistence.

SRQ5 *must have*
The Application Service must allow for removing existing mixers from the Application Persistence.

SRQ6 *must have*
The Application Service must be able to provide all available geometries to the Client Browser (via the HTTP server).

3.1.3 Application Persistence Communication

SRQ7 *must have*
The Application Persistence Communication must be able to access all the data from the Application Persistence.

SRQ8 *must have*
The Application Persistence Communication must be fast enough to serve the data from the Application Persistence to the Application Service.

3.1.4 Application Persistence

SRQ9 *must have*
The Application Persistence must deliver all saved mixers to the Application Service.

SRQ10 *must have*
The Application Persistence must save new mixers handed by the Application Service.

SRQ11 *must have*
The Application Persistence must remove existing mixers handed by the Application Service.

SRQ12 *must have*
The System Administrator is able to add new mixer types for a geometry.

SRQ13 *must have*
The System Administrator is able to remove mixer types for a geometry.

3.1.5 Application Service Communication

SRQ14 *must have*
The Application Service Communication must be able to access all the functionality from the Application Service.

SRQ15 *must have*
The Application Service Communication must be fast enough to provide the functionality from the Application Service to the Client Browser.

3.1.6 Client Browser

Defining an initial concentration distribution

The user can select a circle or square-shaped drawing tool to draw with. He can draw on the canvas by swiping with his finger or drawing the mouse.In addition, the user can reset the drawn distribution to completely white by using a `reset` button.

SRQ16 *must have*
The user can select a colour (black or white) to paint with for the initial concentration distribution, via a toggle button.

SRQ17 *must have*
The user can define an initial concentration distribution with the selected colour on the canvas, by drawing with their finger.

SRQ18 *must have*
After selecting the circle-shaped drawing tool in SRQ34, the user can draw with this tool on the canvas.

SRQ19 *must have*
The user can reset the current concentration distribution to a completely white concentration distribution, by clicking on the “Reset” button.

SRQ20 *should have*
After selecting the square-shaped drawing tool in SRQ35, the user can draw this tool on the canvas.

Select a geometry, mixer and initial concentration distribution

The user can select a rectangular mixer geometry. To this end, a cell browser is available that lists all the available geometries in the first column. This first column will contain several geometries, including `rectangle`, `square`, `circle` or `Journal Bearing`. After clicking on a certain geometry, the available mixers for this geometry are displayed in the second column of the cell browser. The user can now select a mixer of choice and then, the third column in the cell browser lists all the available starting concentration distributions for the selected mixer. There are three types of starting concentration distributions that the user can choose from. The first type is a `blank` concentration distribution, which simply means that the user will be presented with a clean canvas when this option is selected. The second type is a `load` option, so the user can load previously saved concentration distributions that are stored on their device. The third type of concentration distribution is a `predefined` distribution, meaning that the user can choose from a few predefined distributions that are already present in the application. In case of the second and third option, there will be a third column in the cell browser that lists all the available concentration distributions that can be loaded. When the user clicks on an item in the third column, the required distribution is immediately loaded on an appropriate canvas. For the first option, a blank canvas is displayed on the screen when the user clicks on `blank`.
This translates to the following software requirements:

SRQ21 *must have*
The mixing interface contains a cell browser with four columns, to select an initial mixer, geometry and options for the initial concentration distribution.

SRQ22 *must have*
The user can select `rectangle` as a geometry, via the first column in the cell browser.

SRQ23 *must have*
After having selected a geometry, the user can select a mixer that fits the selected geometry. The mixer can be selected via a second column in the cell browser.

SRQ24 *should have*
The user can select `square` as a geometry, via the first column in the cell browser.

SRQ25 *could have*
The user can select `circle` as a geometry, via the first column in the cell browser.

SRQ26 *could have*
The user can select `Journal Bearing` as a geometry, via the first column in the cell browser.

SRQ27 *must have*
The user can select a blank canvas after choosing a geometry and mixer, by choosing the `blank` option in the third column in the cell browser.

SRQ28 *should have*
The user can view a list of all previously saved concentration distributions, by choosing the `load` option in the third column in the cell browser.

SRQ29 *should have*
After selecting the `load` option from SRQ28, the user can select a previously saved distribution by clicking on one distribution in the fourth column in the cell browser.

SRQ30 *could have*
The user can view a list of all predefined concentration distributions, by choosing the `predefined` option in the third column in the cell browser.

SRQ31 *could have*
After selecting the `predefined` option from SRQ30, the user can select a predefined initial concentration distribution by clicking on one distribution in the fourth column in the cell browser.

Change the drawing tool

The user can change the drawing tool that is used to paint on the canvas. The current shape and size of the drawing tool is displayed on the mixing interface, as a button with an image. When the user clicks on this button, a pop-up menu appears with one horizontal panel with two cells. The left cell contains a vertical panel with multiple cells; each cell contains a button and corresponds to a different shape for the drawing tool (circle or square, for instance). The right cell contains a slider to adjust the size of the drawing tool. The user can change the settings for the drawing tool and close the pop-up menu by clicking on the button again.

SRQ32 *must have*
The mixing interface contains a button that displays the current shape of the drawing tool.

SRQ33 *should have*
The button from SRQ32 also displays the current size of the drawing tool.

SRQ34 *must have*
After clicking the button from SRQ32, the user can select a circle-shaped drawing tool via a pop-up panel.

SRQ35 *should have*
After clicking the button from SRQ32, the user can select a square-shaped drawing tool via a pop-up panel.

SRQ36 *should have*
The user can adjust the size of the drawing tool, using a slider in the popup panel from SRQ34 and SRQ35

.

Define mixing protocol for specific geometries

A mixing protocol consists of movements of the geometry (if applicable), and how long these movements are executed. Different movement types are possible for each geometry. A step of a mixing protocol has a certain duration (the step size). This duration D can be any value that can be created by combining the possible base values for this duration , namely 4, 2, 1, 0.5 and 0.25. Such a value D can be entered in a spinner near the drawing canvas. If an invalid value is entered – i.e. a value that cannot be created by combining the mentioned values – this value is rounded up or down to the nearest valid value, whichever is closest. For example, 4.2 is rounded up to 4.25, while 4.1 is rounded down to 4. Each movement is performed for D time units, which can be set using SRQ37.

Depending on the geometry, a step has additional parameters signifying which part moves and how:

For the rectangular or square geometries, a wall movement is defined by either T or B, denoting a movement of the top or bottom wall, respectively. These walls are visible above and below the canvas, respectively, and the user can define the wall movement and direction by swiping over the screen. Looking directly at the screen, a T movement indicates that the top wall should move to the right, and a B movement indicates that the bottom wall should move to the left. This way, the walls normally move in a clockwise direction. Each of these wall movements can be combined with a '$-$' sign, to indicate that the direction of movement should be counter-clockwise. This means $-T$, for example, indicates that the top wall should move to the *left*.

The circular geometry does not support movements, but the mixer that is placed inside the circle does support some form of movement. The user can specify how far he/she wants to move the mixer inside the geometry, and this serves as the protocol.

For the *Journal bearing* geometry, movements are defined by rotating the outer or inner circle. Swiping to the right near the top of the outer circle means the outer circle rotates clockwise for D time units, and swiping to the right near the top of the inner circle means the inner circle rotates clockwise for D time units. Here, too, a '-' sign implies the rotation is executed in the opposite direction.

To indicate how many times the protocol created using the above items is executed, the user can enter a number in a (different) spinner. We use a spinner because the number it contains can easily be incremented or decremented by using the arrow keys, and a value that is not near the original value can easily be entered using its text box.

SRQ37 *must have*
The user can enter a valid step size in a text box.

SRQ38 *must have*
The user can indicate via a spinner how many times the protocol should be applied. This value must be a positive integer.

SRQ39 *must have*
The *Step* class contains a movement and a duration for this movement.

SRQ40 *must have*
The *Protocol* class contains a list of *Step*s.

SRQ41 *must have*
After defining a mixer, a mixing protocol and an initial distribution, the user can press the `Mix` button to execute the actual mixing. This button is greyed out (it cannot be pressed) if no mixing protocol has been defined.

SRQ42 *must have*
The user can reset the protocol, which erases all *Step*s, effectively starting over with defining the protocol.

Visualising the results

The concentration distribution is visualised on the client's screen, using the canvas that was originally used to draw the initial concentration distribution. The performance result is visualised in a graph, which contains all previous performance results for this particular run.

SRQ43 *must have*
Concentration distributions can be drawn on the client's canvas.

SRQ44 *must have*
Performance results can be plotted in a graph.

Execute mixing runs

The user can choose to execute a single step of a mixing protocol, by defining a single *Step*. This is done by choosing one wall movement and its duration. This invokes SRQ41, SRQ99, SRQ100, SRQ106, SRQ101 and SRQ43.

SRQ45 *must have*
After selecting a geometry, a mixer, an initial distribution and a single *Step*, the user can execute a single step, if the "Define protocol" check box has not been checked.

3.1.7 ClientPersistence

SRQ46 *should have*
The user can save an initial concentration distribution locally on their device, using a button and popup panel.

Managing mixing protocols

To manage mixing protocols, the user can save and load the mixing protocols they have created. It is also possible to load a predefined mixing protocol.

SRQ47 *should have*
The user can save an initial concentration distribution locally on their device, using a button and popup panel.

SRQ48 *should have*
The user can click the `Save` button to save the mixing protocol to their device. The desired name for the protocol can then be specified in the pop-up that appears.

SRQ49 *should have*
The user can press the `Remove` button to remove the saved protocol from the device.

SRQ50 *should have*
After the user has selected the geometry and the mixer using SRQ22, SRQ24, SRQ25 or SRQ26 and SRQ23, they can choose to load a previously saved mixing protocol. This opens a menu with the names of all applicable saved protocols. Pressing the name of one such protocol loads it.

SRQ51 *could have*
After SRQ22, SRQ24, SRQ25 or SRQ26 and SRQ23, the user can load a predefined mixing protocol. This opens a menu containing all suitable predefined mixing protocols for the selected geometry. Pressing the name of one such protocol selects it, which means it will be used once the mixing is executed.

Mixing runs

SRQ52 *must have*
After the mixing has been executed, the user can save the results on their device by pressing the `Save` button. This opens a pop-up using which the user can enter a name for the results.

SRQ53 *must have*
In the menu with previously saved mixing runs, the user can press the `Remove` button to remove this saved run from storage.

Save an initial concentration distribution

SRQ54 *should have*
The user can save an initial concentration distribution locally on their device, using a `Save Distribution` button.

SRQ55 *should have*
After clicking the save button from SRQ54, the user can specify a name for the concentration distribution.

SRQ56 *should have*
After specifying a name in SRQ55, the user can save the distribution with this name using the `Save` button.

SRQ57 *should have*
When the save from SRQ56 was successful, the user can click the `OK` button to go back to the mixing interface.

SRQ58 *should have*
If the specified name from SRQ55 is already taken, the user can click the `Overwrite` button to overwrite the previous distribution with the same name.

SRQ59 *should have*
The user can cancel the saving process, by clicking the `Cancel` button; the user will now return to the mixing interface.

Load previously saved concentration distribution

SRQ60 *should have*
The mixing interface has a `Load` button that allows the user to load previously saved concentration distributions.

SRQ61 *should have*
After clicking the `Load` button from SRQ60, the `Load distribution` interface appears.

SRQ62 *should have*
The `Load distribution` interface contains a cell browser with two rows.

SRQ63 *should have*
The first row of the cell browser from SRQ62 contains a `Load` option, to load previously saved distributions.

SRQ64 *should have*
The second row of the cell browser from SRQ63 displays all previously saved concentration distributions for the selected option of SRQ63.

SRQ65 *should have*
The user can load a specific concentration distribution by tapping on it; the mixing interface is now shown, with the selected distribution displayed on the canvas.

SRQ66 *should have*
The user can press the `OK` button in the message from SRQ75 to return to the `Load distribution` interface.

SRQ67 *could have*
The first row of the cell browser from SRQ62 contains a `Predefined` option, to load predefined concentration distributions.

SRQ68 *could have*
The second row of the cell browser from SRQ62 displays all predefined concentration distributions for the selected option of SRQ67.

Remove previously saved concentration distributions

SRQ69 *should have*
The mixing interface has a `Remove saved distributions` button to remove previously saved distributions.

SRQ70 *should have*
After clicking the `Remove saved distributions` button, the history interface is shown, with a cell list that provides all the previously saved concentration distributions.

SRQ71 *should have*
Each item from in the cell list of SRQ70 contains a check box that can be selected or deselected, to indicate which distributions should be removed.

SRQ72 *should have*
After selecting at least one item from SRQ71, the user can press the `Remove saved distributions` button and a *Are you sure?* message is shown.

SRQ73 *should have*
The user can remove all selected items from SRQ71 by clicking the `Yes` button in the dialogue from SRQ72.

SRQ74 *should have*
The user can cancel the removal process by clicking the `No` button in the dialogue from SRQ72.

SRQ75 *should have*
If no distributions have been saved yet, a *No saved distributions* message is shown.

SRQ76 *should have*
The user can press the `OK` button in the message from SRQ75 to return to the history interface.

SRQ77 *should have*
If the user has not given access rights to the application, a *Insufficient access rights* error message is shown.

SRQ78 *should have*
The user can press the `OK` button in the message from SRQ77 to return to the history interface.

SRQ79 *should have*
After pressing the button from requirements SRQ73 or SRQ74, the user returns to the history interface.

Saving and exporting results

The results window can be opened by choosing `Results` from the main interface. This window contains a `save` button, an `export` button and a `load results` button. If the `export` button is pressed, a pop-up with several available export options is shown: `export performance`, `export picture` and `export animation`. `export animation` is only available if the `animate mixing` checkbox was checked in the mixing interface during the run. The user can then name the file. If during saving the specified name for `saving/exporting` is already in use, the application returns a `name already in use` message. Additionally, a `load performance` button is available in the results window. The user can use this button to open a popup. In this popup they can select the files they want to load. The button `load selected` can now be pressed and now loads all the selected files. If no files are selected, the button named `cancel` simply cancels without loading. An alternative way to cancel loading is to click somewhere not on the pop-up.

SRQ80 *should have*
The `export performance` option can be used to open the `export` menu to `export` an image of the current performance graph.

SRQ81 *could have*
The `export animation` button can be used to open the `export` menu to `export` an animation of the mixing procedure.

SRQ82 *should have*
In the export menu, the user can specify a name for the exported file.

SRQ83 *should have*
If the user clicks on the save button in the `export` menu, the file is stored at the location where the browser stores files, with the name specified in SRQ82.

SRQ84 *should have*
The user should be able to cancel loading files by clicking outside of the loading popup.

SRQ85 *should have*
If there are no access rights to store the exported/saved data at the selected location when SRQ83 is executed, the application returns a `no access rights` message.

SRQ86 *should have*
If there is no memory space to store `exported/saved` data when SRQ83 is executed, the application returns an `out of memory` message.

SRQ87 *must have*
If a file exists with the same name as the one entered in SRQ82, the application returns a `File exists` message.

SRQ88 *should have*
The popups opened by SRQ87 and SRQ86 can be closed with an `ok` button.

SRQ89 *should have*
The user can overwrite the old file in SRQ87 with an `overwrite` button if a saved file already exists.

SRQ90 *should have*
The user can cancel overwriting the old file in SRQ87 with a `cancel` button if a saved file already exists.

Loading results

In the results window a `load results` button is available. This button opens a loading window in which a `results` file is opened. Multiple results can be selected.

SRQ91 *should have*
The `load results` button can be used to open the load menu for a `results` file.

SRQ92 *should have*
In the load menu one or more `results` files can be selected to be loaded.

SRQ93 *should have*
In the load menu, the `load` button can be used to load the files selected in SRQ92.

SRQ94 *should have*
When SRQ93 is executed, the end mixture and the performance are loaded from files selected in SRQ92.

SRQ95 *should have*
If multiple files are selected in the loading menu, they are all depicted in a single performance graph.

SRQ96 *should have*
If in SRQ94 multiple files are selected, the user is able to browse through the result mixtures.

SRQ97 *should have*
In the load menu, the `cancel` button can be used to exit the loading menu.

Language selection

In the menu bar a flag of the currently selected language is visible. When clicked, a popup appears containing several language options, each symbolised by the flag of their country. These flags can be clicked to change the language to that specific language. Standard language is English. The preferred language should be stored on the client device.

SRQ98 *could have*
The user can choose a different language for the application.

3.1.8 Simulator Service Communication

To execute a mixing protocol, the following SRQs are executed in this order: SRQ41, SRQ99, SRQ100, SRQ106, SRQ101 and SRQ43.

SRQ99 *must have*
The parameters (initial concentration distribution, geometry, mixing protocol, mixer and number of protocol applications) can be sent to the server through the Simulator Service Communication channel.

SRQ100 *must have*
The server is able to parse the parameters that it receives and interpret those.

SRQ101 *must have*
The server can send the resulting distribution and performance statistics to the Client Browser via the HTTP server.

Requirements regarding the presentation of results

The user is able to get the distribution that resulted from executing his mixing run. In additon, the mixing performance is given after each mixing step. He is also able to save and export these results. To this end a results window is available. In the results window, the final distribution is shown. There is also a `show statistics` button. When pressed, the performance of the mixing run is shown in a graph. In this new menu the `show statistics` button changes to `show result`. When this button is pressed, the image switches back to the default results window. The results window is reached by pressing the `start mixing` button in the mixing interface. The mixing interface is the window where the user can define his mixing protocol. In the mixing interface there is a `animate mixing` checkbox. If this checkbox was checked when the `start mixing` button is pressed, an animation of the mixing procedure is shown in the results window.

SRQ102 *must have*
In the results window the result of the user's mixing protocol is shown. This interface can be reached by pressing the `start mixing` button.

SRQ103 *must have*
The user can use the `show performance` button to view the performance graph of the mixing run.

SRQ104 *must have*
The user can use the `show result` button to switch back to the standard results window and view the result distribution of the mixing run.

SRQ105 *could have*
An animation of the mixing run is shown in the results window if the `animate mixing` checkbox was checked.

3.1.9 Simulator Service

SRQ106 *must have*
The server executes the mixing run using the parameters received via the HTTP server. These parameters are passed as input to the FORTRAN implementation.

Information exchange

The Simulator Service must be able to send and receive matrix files from the Client Browser and to the Fortran Module. These matrices specify the characteristics of the mixers.

SRQ107 *must have*
The Simulator Service must be able to receive protocol information from the Client Browser.

SRQ108 *must have*
The Simulator Service must be able to pass (references to) matrix files to the Fortran Module, which the Fotran Module can use for simulations.

SRQ109 *must have*
The Simulator Service must be able to pass protocol information to the Fortran Module.

SRQ110 *must have*
The Simulator Service must, together with the Application Service, make sure that the listed mixers in the Application Persistence are known to the Fortran Module, for each geometry.

3.1.10 Fortran Module

SRQ111 *must have*
The Fortran Module must accept input (matrix, protocol information) from the Simulator Service.

SRQ112 *must have*
The Fortran Module must provide output (vectors) to the Simulator Service.

3.2 Non-functional requirements

3.2.1 Performance

SRQ113 *must have*
Waiting time between submitting input and receiving output is around 5 seconds on average.

SRQ114 *should have*
Waiting time between submitting input and receiving output is around 3 seconds on average.

SRQ115 *could have*
Waiting time between submitting input and receiving output is around 1 second on average.

3.2.2 Interface

SRQ116 *must have*
An English interface is available.

SRQ117 *should have*
A Dutch interface is available.

3.2.3 Operational

get more requirements

3.2.4 Resource

get more requirements

3.2.5 Verification and testing

get more requirements

3.2.6 Portability

SRQ118 *must have*
The application runs on iOS Safari version 6.0 and higher.

SRQ119 *should have*
The application runs on Firefox version 20 and higher.

SRQ120 *should have*
The application runs on Google Chrome version 26 and higher.

SRQ121 *could have*
The application runs on Internet Explorer version 10 and higher.

SRQ122 *could have*
The application runs on Safari version 6.0 and higher.

SRQ123 *must have*
The application runs on devices running on iOS version 6 and higher.

SRQ124 *should have*
The application runs on devices running on Android version 4.0 and higher.

SRQ125 *could have*
The application runs on devices running on Windows 8.

3.2.7 Maintainability

get more requirements

3.2.8 Reliability

get more requirements

3.2.9 Security

get more requirements

3.2.10 Safety

get more requirements

3.2.11 Documentation

get more requirements

3.2.12 Extensibility

get more requirements

Chapter 4

Requirements traceability matrix

4.1 URD to SRD

The following matrix lists which SRQs implement the CPRs from the URD.

CPR	SRD				
1	21, 22	2	23	3	24
4	25	5	26	6	16, 17, 27
7	18, 32, 34	8	19	9	35, 20
10	33, 36	11	46, 47, 57, 58, 59, 54, 55, 56	12	69, 76, 78, 79, 70, 71, 72, 73, 74, 75, 77
13	28, 60, 27, 29, 61, 64 29, 64, 65, 66, 62, 63, 75	14	30, 67, 68	15	
16		17	39, 40	18	45
19	42	20	37	21	38
22	48	23	49	24	50
25	40	26	40	27	40
28	51	29	41, 43, 99, 100, 101	30	52, 91, 92, 93, 94, 97
31	53	32	102, 44	33	82, 83, 84, 85, 86, 88, 89, 90
34	103, 104	35	80	36	95, 96
37	80, 82, 83, 84, 85, 86, 88, 89, 90	38	105	39	81, 82, 85, 86, 88, 89, 90
40	116	41	117		

The following matrix lists which SRQs implement the CNRs from the URD.

CNR	SRD				
1	118	2	119	3	120
4	121	5	122	6	
7	123	8	124	9	125
10	113	11	114	12	115
13	4				

4.2 SRD to URD

The following matrix lists all CPRs from the URD implemented by each SRQ.

SRD	CPR		
1		2	3
4		5	6
7		8	9
10		11	12
13		14	15
16		17	18
19		20	21
22		23	24
25		26	27
28		29	30
31		32	33
34		35	36
37		38	39
40		41	42
43		44	45
46		47	48
49		50	51
52		53	54
55		56	57
58		59	60
61		62	63
64		65	66
67		68	69
70		71	72
73		74	75
76		77	78
79		80	81
82		83	84
85		86	87
88		89	90
91		92	93
94		95	96
97		98	99
100		101	102
103		104	105
106		107	108
109		110	111
112		113	114

The following matrix lists all CNRs from the URD implemented by each SRQ.

SRD	CPR		
1		2	3
4		5	6
7		8	9
10		11	12
13		14	15
16		17	18
19		20	21
22		23	24
25		26	27
28		29	30
31		32	33
34		35	36
37		38	39
40		41	42
43		44	45
46		47	48
49		50	51
52		53	54
55		56	57
58		59	60
61		62	63
64		65	66
67		68	69
70		71	72
73		74	75
76		77	78
79		80	81
82		83	84
85		86	87
88		89	90
91		92	93
94		95	96
97		98	99
100		101	102
103		104	105
106		107	108
109		110	111
112		113	114