TU/e

Project FINGERPAINT

ADD-0.1

Architectural Design 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 contains descriptions of the architecture of the FINGERPAINT application. This project is part of the Software Engineering Project (2IP35) and is one of the assignments at Eindhoven University of Technology. The Architectural Design Document (ADD) is based on the ESA standard for software development, as set by the European Space Agency (ESA) [1].

Contents

Document Status Sheet

Document Status Overview

General

Document title:	Architectural Design Document
Identification:	ADD-0.1
Author:	Tessa Belder, Thom Castermans, Femke Jansen
Document status:	Externally approved

Document History

Version	*Date*	*Author*	*Reason of change*
0.0	11-Jun-2013	Tessa Belder, Thom Castermans, Femke Jansen	Revised chapters 4, 5 and 7 with respect to adding/removing mixers.
0.1	13-Jun-2013	Tessa Belder, Thom Castermans, Femke Jansen	Revised version following from feedback of team managers.

Document Change Records Since Previous Issue

General

Date:	13-Jun-2013
Document title:	Architectural Design Document
Identification:	ADD-0.1

Changes

Page	*Paragraph*	*Reason to change*
-	-	Processed feedback on version 0.0 from team managers.

Chapter 1

Introduction

This chapter lists general information about this document.

1.1 Purpose

The Architectural Design Document (ADD) describes the basic design of the FINGERPAINT application that is being developed by Group Fingerpaint. First of all, it describes how the application is divided into different components. Then, for each component, it describes the dependencies to other components, and finally the relation to external interfaces.

1.2 Scope

The FINGERPAINT application is an application designed and developed by Group Fingerpaint for prof.dr.ir. P.D. Anderson, at the Eindhoven University of Technology. The application 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.

1.3 List of definitions and abbreviations

1.3.1 Definitions

Broadband connection A broad range of technologies, all of which provide higher data rate access to the Internet.

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

Fortran: A general-purpose, imperative programming language.

Google Web Toolkit: An open source set of tools that allows for the creation and maintenance of JavaScript applications in Java.

Mixing step: Part of a mixing protocol. A mixing step involves a step size and possibly a movement of a wall with a given direction.

Subordinate: A child component.

Wall: A moveable part of the application that can be used to define the direction of the mixing step to be executed. The wall is compatible with the selected geometry. For instance, when a rectangular geometry is used, there are two rectangular walls (top and bottom) that can be moved to the left or right.

1.3.2 Abbreviations

2IP35	The Software Engineering Project
ADD	Architectural Design Document
ADSL	Asymmetric Digital Subscriber Line
GUI	Graphical User Interface
GWT	Google Web Toolkit
JNI	Java Native Interface
Mb	Megabit
SEP	Software Engineering Project
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, "Software requirements document," *SEP*, 2013.

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

[4] Stuart Knightley, "Homepage of the JSZip library." `http://stuk.github.io/jszip/`. [Online; accessed 13-June-2013].

1.5 Overview

The remaining chapters describe the architectural design of FINGERPAINT in more detail. Chapter 2 gives a system overview. Chapter 3 describes the system context. The relationship with external components is explained in detail in this chapter. Chapter 4 covers the sytem design. The name and reference of the used design method are given. In chapter 5 all components are described in detail. For each component, its type, purpose, function, subordinates, dependencies to other components, interfaces, resources needed, internal processing and data are described. Chapter 6 gives an overview of all resources needed to build, operate and maintain the application. Chapter 7 contains a traceability matrix. This matrix shows how each software requirement of the SRD [2] is linked to the components described in the ADD.

Chapter 2

System overview

A description of the FINGERPAINT application can be found in the URD [3]. Furthermore, the SRD [2] includes descriptions of the background of FINGERPAINT and the environment it operates in. FINGERPAINT is a web-based information system. It does not interact with any other information systems in its environment. Figure 2.1 shows the environment of FINGERPAINT.

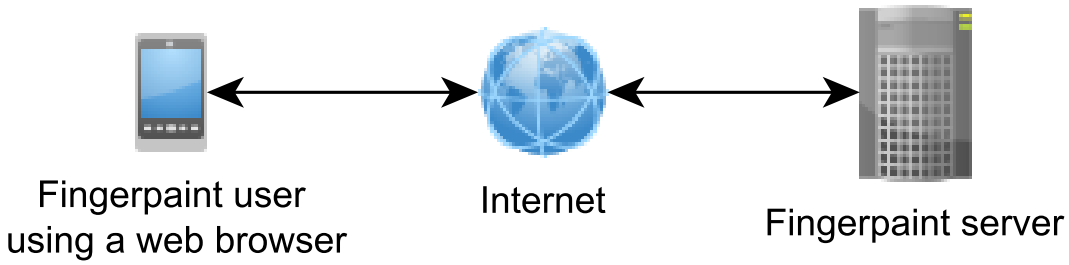

Figure 2.1: The environment of FINGERPAINT

Chapter 3

System context

In this chapter, the context of the system is discussed. This means any external components that are used by the FINGERPAINT application. The FINGERPAINT application uses one external component: a Fortran server. This component is described is section 3.1. For communication with this server data compression is used, which is described in section 3.2.

3.1 Fortran server

Any computations that need to be done by the FINGERPAINT application are offloaded to the Fortran module as described in section 2.7.4 of the SRD [2]. Communication with this module is done by calling its procedure in Fortran. This procedure simulates one mixing step and uses several parameters. Some of these parameters describe the length of other parameters, this is needed to ensure that the application only uses its own memory. These parameters are: `len_geometry`, `len_matrix`, `len_concentration_vector` and `len_step_name`. The other parameters are described in the table below:

`geometry`	This is a string that identifies which geometry is used. The idea is that this string is a descriptive string, that can also be used in the GUI. For example, this string can be "Rectangle 400x240". There is a set that contains all the different geometries, to make sure that these strings are consistent throughout the application.
`matrix`	This is a string that identifies which matrix is used, from the matrices available for the selected geometry. The string should, just like the above string, be descriptive. There is a set that contains all the different mixers, to make sure that these strings are consistent throughout the application.
`concentration_vector`	A vector that describes the concentration distribution on the canvas: simply an array of values between 0.0 and 1.0. This value will be changed by the function.
`step_size`	Size of the mixing step that is to be simulated.

`step_name`	Describes the type of step that should be simulated (e.g. for a rectangular geometry it contains which of the walls is moved and in which direction).
`segregration`	A value that defines how well a mixture is separated, a lower segregation means the mixture is well-mixed. This is not an input parameter, but a pointer to a value that will be set by this function: an output parameter.

3.2 Data compression

As described in the previous section, all parameters needed to perform the calculations need to be uploaded from the client's device to the Fortran server. This poses a possible problem, however, as the `len_concentration_vector` parameter contains such a large amount of data that uploading it with a speed of 1 to 2 Mb/s takes longer than the time requirements mentioned in the URD [3]. Since this is the average upload speed for mobile and ADSL broadband connections, and the FINGERPAINT application will mainly be used from mobile devices, it is a priority that it also functions correctly and within a timely matter at these speeds. Therefore, the data of the `len_concentration_vector` parameter is compressed before uploading it to the Fortran server, which decreases the time needed to upload it.

For this purpose, the open source JavaScript Libray JSZip [4] is used on the client-side of the application to compress the data. On the server-side Java functions are used to decompress the data again before sending it to the Fortran module to perform the actual calculations. Whenever the server sends data to the client, this data is automatically compressed, if the client's device supports this functionality.

Chapter 4

System design

This chapter describes the technical aspects of the design of Fingerpaint.

4.1 Design method

Fingerpaint is implemented as a web application using the Java software development framework Google Web Toolkit (GWT). Section 4.2 defines the components of Fingerpaint and their dependencies.

4.2 Decomposition description

The decomposition of Fingerpaint into components is based on the requirements of the URD [3] and the SRD [2].

Section 4.2.1 lists the components, which are further described in chapter 5. Section 4.2.2 illustrates the dependencies between the components. The identified components are strongly based on the tiers listed in section 2.7 of the SRD [2]. An overview of these tiers is given in Figure 2.1 of the SRD [2].

4.2.1 List of components

The following components are identied:

- Server
  - Application Persistence
  - HTTP Server
  - Fortran Module
  - Application Service
  - Simulator Service

- Client
  - Client Browser
    - Layout
    - Application State
  - Client Persistence

4.2.2 Dependencies diagram

Figure 4.1 illustrates the dependencies between the components of FINGERPAINT. Arrows indicate a 'depends on' relation between components. This relationship means that the component on the start of the arrow needs the functionality of the component on the end of the arrow to fulfill its task. A double arrow indicates a bidirectional relationship, so the components on both sides of the arrow need eachother.

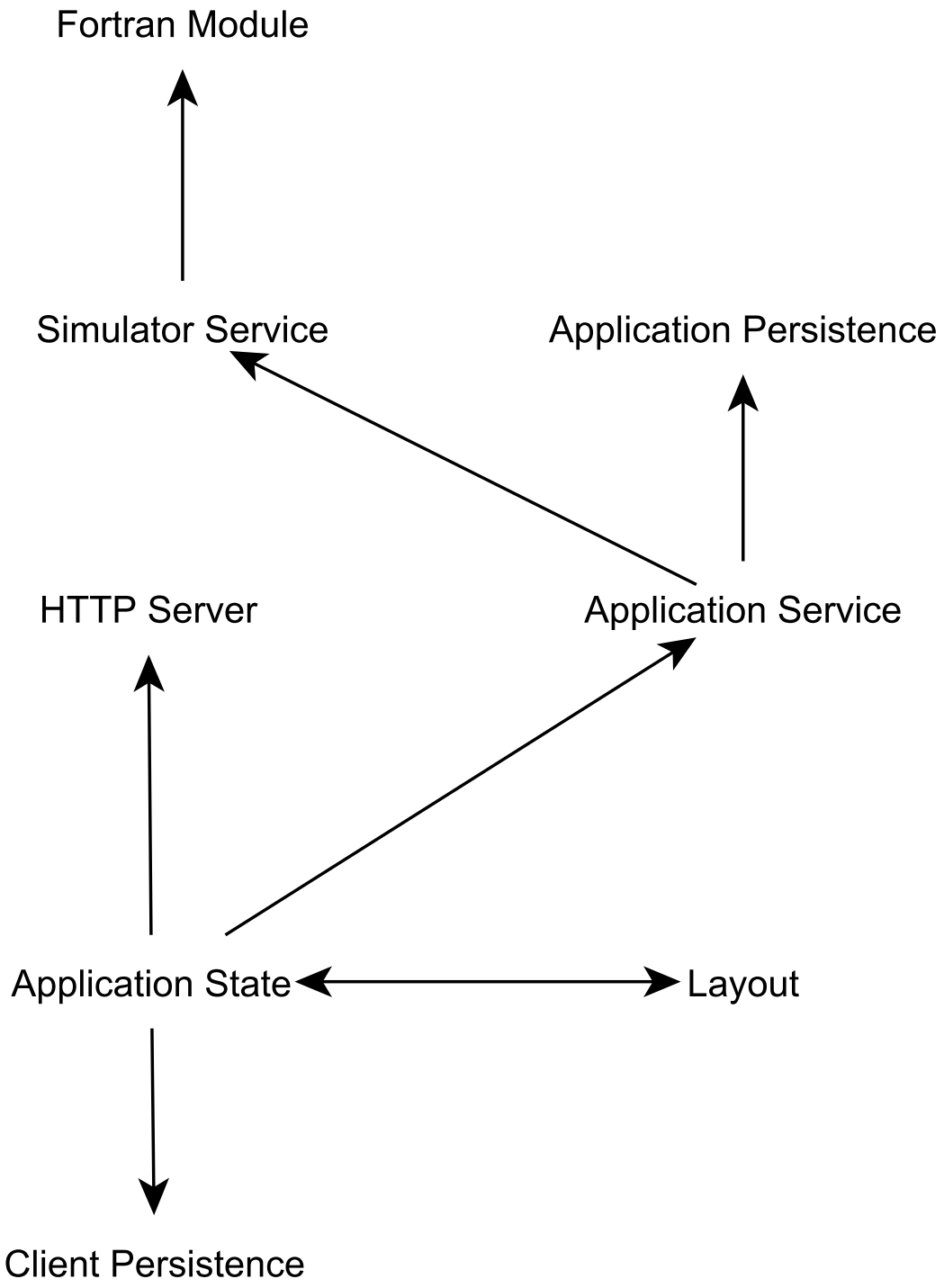

Figure 4.1: Components of FINGERPAINT and their dependencies

Chapter 5

Component description

In this chapter we describe all of the components that were identified in chapter 4 in detail. For every component, we give an identifier and the type, purpose, function, subordinates, dependencies and interfaces of the component. Furthermore, the internal control flow and internal data flow are described.

5.1 Server

5.1.1 Fortran Module

Component Identifier

Server.FortranModule

Type

Program.

Purpose

Provides functionality for the following software requirements:
SRQ43, SRQ83, SRQ84, SRQ86, SRQ87, SRQ88, SRQ89

Function

The Fortran Module calculates how the mixing distribution changes as a mixing step is applied.

Subordinates

The Fortran Module does not have any known child components. This is because the module was given to Group Fingerpaint as a black box.

Dependencies

The Fortran Module does not have dependencies in relation to other components. This is because it uses its own data, along with the parameters for computations and only interfaces with its parent module (the Simulation Service).

Interfaces

The Fortran Module has a single interface with the Simulation Service. This interface is the procedure call of the module in Fortran. This call receives a concentration distribution, a geometry, mixer and a mixing step as arguments and returns a concentration distribution and the segregation factor. For more detail regarding this interface, refer to chapter 3.

References

The description of the specific requirements, mentioned in the "Purpose"-section, can be found in the SRD [2].

Processing

As the implementation of the Fortran Module is beyond the scope of project FINGERPAINT, the internal flow is irrelevant.

Data

As the implementation of the Fortran Module is beyond the scope of project FINGERPAINT, the internal data is irrelevant.

5.1.2 Simulator Service

Component Identifier

Server.SimulatorService

Type

Program.

Purpose

Provides functionality for the following software requirements:
SRQ43, SRQ81, SRQ82, SRQ83, SRQ84, SRQ85, SRQ86

Function

The purpose of the Simulator Service is to simulate a mixing run on the server and compute how the concentration distribution changes when such the mixing protocol from this run is applied.

Subordinates

The Simulator Service is composed of multiple Java classes, located in two packages. This is because one part is located on the client side, while another part resides on the server (calling the Fortran module, that runs on the server). The structure of these packages is shown below:

- nl.tue.fingerpaint.server.simulator
  - NativeCommunicator.java
  - SimulatorServiceImpl.java
- nl.tue.fingerpaint.client.simulator
  - Simulation.java
  - SimulationResult.java
  - SimulatorService.java
  - SimulatorServiceAsync.java

Dependencies

The Simulator Service depends on the Fortran Module for technical calculations.

Interfaces

The Simulator Service uses the Fortran Module's interface to communicate with it. That interface is described in subsection 5.1.1, more specifically the section "Interfaces". This communication is done via C code, which in turn calls the Fortran function.

The Simulator Service also receives information regarding mixing runs that have to be executed from the Application Service, and it sends the resulting distribution back to the Application Service. This communication is done by the Simulator Service Communication, which uses JNI.

References

The description of the specific requirements, mentioned in the "Purpose"-section, can be found in the SRD [2].

Processing

For processing requests, the Simulator Service keeps each request separate. For each request, the protocol is split up in single steps, which are separately sent to the Fortran Module.

Data

The Simulator Service has no internal data other than the protocol and distribution defined by the request. This data is modified as each step of the protocol is processed by the Fortran Module and finally returned when all steps are analysed. Note that the concentration distribution may be duplicated a number of times, when intermediate results are requested.

5.1.3 Application Persistence

Component Identifier

Server.ApplicationPersistence

Type

Database.

Purpose

Provides functionality for the following software requirements:
SRQ2, SRQ3, SRQ4, SRQ5, SRQ6, SRQ7, SRQ8, SRQ9, SRQ10, SRQ11, SRQ12, SRQ13, SRQ46

Function

This component is the database that contains the different mixers and their information. Mixers can be retrieved from this component for use within the Fortran Module. All the communication to execute these functions is done by the Application Persistence Communication, which is described in section 2.7.3 of the SRD [2].

Subordinates

The Application Persistence does not have any child components.

Dependencies

The Application Persistence component does not depend on any other components.

Interfaces

This component is accessed through the Application Service component. It receives requests to return certain mixers from the Application Service and sends the requested data to the Application Service. The Application Persistence Communication, which consists of Java calls, is used for this purpose.

References

The description of the specific requirements, mentioned in the "Purpose"-section, can be found in the SRD [2].

Processing

No processing of data is done in this component.

Data

This component is a database containing data that represents the different mixers.

5.1.4 HTTP Server

Component Identifier

Server.HTTPServer

Type

Program.

Purpose

Provides functionality for the following software requirements:
SRQ1, SRQ43, SRQ85

Function

The HTTP server is a piece of software that responds to requests by serving a file from a static collection of content. This is used to serve the application on the client side.

Subordinates

The HTTP Server does not have any known child components. This is because this component was not built by Group Fingerpaint.

Dependencies

The HTTP Server component does not depend on any other components.

Interfaces

This component is accessed through the Application State Component, which sends requests for web pages to the HTTP Server. The HTTP server responds to these request by serving the requested file(s). All this communication is done through HTTP.

References

The description of the specific requirements, mentioned in the "Purpose"-section, can be found in the SRD [2].

Processing

As the implementation of the HTTP server component is beyond the scope of project Fingerpaint, the internal flow is irrelevant.

Data

The HTTP server component contains a static collection of content files, that can be served to the client side of the application.

5.1.5 Application Service

Component Identifier

Server.ApplicationService

Type

Program.

Purpose

Provides functionality for the following software requirements:
SRQ2, SRQ3, SRQ4, SRQ5, SRQ6, SRQ7, SRQ10, SRQ11, SRQ12, SRQ13, SRQ14, SRQ15, SRQ43, SRQ81, SRQ82, SRQ85

Function

Whenever centralized data or other communication is required by the application running on a Client, this is done through the Application Service using the Application Service Communication channel.

Subordinates

This component is composed of three packages, which contain Java classes. The structure of the packages is shown below:

- nl.tue.fingerpaint.client.serverdata
  - ServerDataCache.java
  - ServerDataService.java
  - ServerDataServiceAsync.java
- nl.tue.fingerpaint.server
  - ServerDataServiceImpl.java
- nl.tue.fingerpaint.shared
  - ServerDataResult.java

Dependencies

The Application Service depends on the Application Persistence component when the Application State component makes a request for data from the Application Persistence. Furthermore, the Application Service depends on the Simulator Service component when the Application State sends a mixing run to be executed on the Fortran Module.

Interfaces

The Application Service has an interface with the Application Persistence component. This communication is done through a Java call. The Application Service sends requests to the Application persistence to return a mixer. It receives data belonging to mixers from the Application Persistence.

The Application Service also has an interface with the Simulator Service component. This communication is done through JNI. The application service sends information belonging to a mixing run to the Simulator Service. It receives the result of the calculated mixing run from the Simulator Service.

References

For an in-depth description of the Application Service and the functionality described in the "Purpose"-section, see the SRD [2].

Processing

Basically, the only thing the Application Service does is passing data through from the Application State to the Simulation Service or Application Persistence, and vice versa. This would imply that no processing is done. This is not entirely true however, as some data is compressed before sending it to the client. Thus, the Application Service may format the data before sending it to the client or the back end; see section 3.2 for more information.

Data

The Application Service does not have any internal data. The only data residing in this component is the data passing through with requests.

5.2 Client

5.2.1 Layout

Component Identifier

Client.Layout

Type

GUI.

Purpose

Provides functionality for the following software requirements:

SRQ16, SRQ17, SRQ18, SRQ19, SRQ20, SRQ21, SRQ22, SRQ23, SRQ25, SRQ24, SRQ26, SRQ27, SRQ28, SRQ29, SRQ30, SRQ31, SRQ32, SRQ33, SRQ34, SRQ35, SRQ36, SRQ37, SRQ38, SRQ39, SRQ40, SRQ41, SRQ42, SRQ43, SRQ44, SRQ45, SRQ47, SRQ48, SRQ49, SRQ50, SRQ51, SRQ52, SRQ53, SRQ54, SRQ55, SRQ57, SRQ58, SRQ59, SRQ60, SRQ61, SRQ62, SRQ63, SRQ64, SRQ65, SRQ66, SRQ67, SRQ68, SRQ70, SRQ71, SRQ72, SRQ73, SRQ74, SRQ75, SRQ76, SRQ77, SRQ78, SRQ79, SRQ**??**, SRQ80, SRQ93, SRQ94, SRQ95, SRQ96, SRQ97, SRQ98, SRQ99, SRQ100, SRQ101, SRQ102

Function

This component is responsible for the GUI layout.

Subordinates

The layout is generated by GWT, which means that there are no files that actually contain the layout. However, there are files that control the output and which are parsed/used by GWT to generate the actual layout. These files are listed below:

- fingerpaint.nocache.js *generated by GWT: will load the application*
- fingerpaint.css *used by GWT to generate inline CSS*
- Fingerpaint.html *base HTML, populated dynamically with JavaScript*

Dependencies

The layout component depends on the Application State component. If the data inside this component changes, the layout component has to be updated accordingly.

Interfaces

This component does not have any interfaces. It is part of the Client Browser, and any interfacing with this component is done through the Application State component.

References

The description of the specific requirements, mentioned in the "Purpose"-section, can be found in the SRD [2].

Processing

In this component data regarding information that was input by the user is sent to the Application State component. The data is processed there. This component also receives data from the Application State component regarding the contents of the GUI. The Layout component uses this data to update the GUI.

Data

This component does not keep any internal data.

5.2.2 Client Persistence

Component Identifier

Client.ClientPersistence

Type

The Client Persistence is a set of files stored on the system of the client.

Purpose

Provides functionality for the following software requirements:
SRQ47, SRQ48, SRQ49, SRQ50, SRQ51, SRQ52, SRQ53, SRQ56, SRQ62, SRQ75, SRQ76

Function

The Client Persistence is responsible for storing results and user-defined protocols and distributions.

Subordinates

The Client Persistence does not have any child components.

Dependencies

The Client Persistence component does not depend on any other components.

Interfaces

The Application State uses the Client Persistence interface for storing and retrieving files. This communication is done through a Java-call. This component is accessed through the Application State component. It receives requests to add, remove or return distributions or protocols from the Application State. If it receives a request to return a file, it will send the requested data to the Application State. The communication between these two components consists of Java calls.

References

The description of the specific requirements, mentioned in the "Purpose"-section, can be found in the SRD [2].

Processing

No processing of data is done in this component.

Data

The data stored in the Client Persistence is a collection of files representing mixing protocols and concentration distributions.

5.2.3 Application State

Component Identifier

Client.ApplicationState

Type

Program

Purpose

Provides functionality for the following requirements:
SRQ47, SRQ48, SRQ49, SRQ50, SRQ51, SRQ52, SRQ53, SRQ54, SRQ55, SRQ56, SRQ62

Function

The Client Browser provides the user with a Graphical User Interface. All interactions of the user with the application are performed through this GUI. Whenever a user action on the GUI makes any change to the state of the application this change is stored in the Application State. When any information stored in the Application State needs to be displayed on the GUI it is accessed as well. For saving a file locally on the device the Application State is needed to communicate and forward this to the Client Persistent component to store it there. Finally, it also communicates mixing runs to the Application Service component to forward it to the Fortran Module which does the computation.

Subordinates

In a similar manner as the layout component, as described in subsection 5.2.1, the components regarding the application state are parsed by GWT. For this purpose, several classes from the following packages are used:

- nl.tue.fingerpaint.client.model
- nl.tue.fingerpaint.client.serverdata
- nl.tue.fingerpaint.client.simulator
- nl.tue.fingerpaint.client.storage

Dependencies

For saving a file locally on the device the Application State state depends on the Client Persistence to make this happen. To make any changes to the state of the application it depends on the Layout to issue the parameters for the new value. To retrieve the files with web content that need to be displayed, it depends on the HTTP Server. Finally, when the user wants to execute a mixing step/run, and the Layout communicates this to the Application State, it depends on the Application Service component to forward it to the Fortran Module which does the computation.

Interfaces

The Application State can send information that needs to be stored locally on the device to the Client Persistence. It also sends information about a mixing run that has to be executed to the Application Service, and receives information regarding the end results of a mixing run from the Application Service. All this communication is done through Java calls. Finally, it also receives files with web content from the HTTP Server, which is done through HTTP.

References

For an in-depth description of the Application Service and the functionality described in the "Purpose"-section, see the SRD [2].

Processing

The Application State converts a mixing run call to a JSON format to send to the Application Service. When files need to be saved it converts the data to String format.

Data

The data stored in the Application State are the variables entered by the user in the Layout component.

Chapter 6

Feasibility and resource estimates

This chapter gives an estimation of the computer resources which are needed to develop and operate Fingerpaint. Also, when all of the resource requirements in this chapter are met, the software requirements SRQ9, SRQ15 from the SRD [2] are also met.

The requirements for the development of Fingerpaint are:

CPU	$\geq$ 1.0 GHz x86 or equivalent
Memory	$\geq$ 4 GB RAM (or at least enough disk space to swap to)
Hard disk	$\geq$ 500 MB free on hard disk
Operating System	Windows 7 or higher, or some up-to-date Linux distribution
Software	Java 1.7 or higher, Google Web Toolkit SDK 2.5.1, iOS Safari version 6.0 or higher, Firefox version 20 or higher, Google Chrome version 26 or higher, Internet Explorer version 10 or higher and Safari version 6.0 or higher.

The requirements for the server needed to develop the Fingerpaint application are:

CPU	$\geq$ 1.0 GHz x86 or equivalent
Memory	$\geq$ 2 GB RAM
Hard disk	$\geq$ 200 MB free on hard disk
Operating System	Windows 7 or higher, or some up-to-date Linux distribution
Software	Java 1.7 or higher, Google Web Toolkit SDK 2.5.1.

The requirements for the operating the Fingerpaint application on the server side are:

CPU	$\geq$ 1.0 GHz x86 or equivalent
Memory	$\geq$ 2 GB RAM
Hard disk	$\geq$ 200 MB free on hard disk
Operating System	Windows 7 or higher, or some up-to-date Linux distribution
Software	Java 1.7 or higher, Google Web Toolkit SDK 2.5.1.

The requirements for operating the FINGERPAINT application on a client device are:

CPU	$\geq$ 1.0 GHz x86 or equivalent
Memory	$\geq$ 256 MB RAM
Operating System	Any supporting the software
Software	iOS Safari version 6.0 or higher, Firefox version 20 or higher, Google Chrome version 26 or higher, Internet Explorer version 10 or higher or Safari version 6.0 or higher.

Chapter 7

Requirements Traceability Matrix

In this chapter, we link SRQs, described in the SRD [2], to the components described in this document. We also do this the other way round, namely listing which SRQs are implemented in a component.

7.1 SRD to ADD

The following matrix lists which components are linked to the SRQs in the SRD.

SRQ	Component(s)	SRQ	Component(s)
1	Server.HTTPServer	2	Server.ApplicationService
3	Server.ApplicationService	4	Server.ApplicationService
5	Server.ApplicationService	6	Server.ApplicationService
7	Server.ApplicationService	8	Server.ApplicationPersistence, Server.ApplicationService
9	Server.ApplicationPersistence, Server.ApplicationService	10	Server.ApplicationPersistence
11	Server.ApplicationPersistence	12	Server.ApplicationPersistence
13	Server.ApplicationPersistence	14	Server.ApplicationService
15	Server.ApplicationService	16	Client.Layout
17	Client.Layout	18	Client.Layout
19	Client.Layout	20	Client.Layout
21	Client.Layout	22	Client.Layout
23	Client.Layout	24	Client.Layout
25	Client.Layout	26	Client.Layout
27	Client.Layout	28	Client.Layout
29	Client.Layout	30	Client.Layout
31	Client.Layout	32	Client.ApplicationState, Client.Layout
33	Client.ApplicationState, Client.ClientPersistence, Client.Layout	34	Client.ApplicationState, Client.Layout

SRQ	Component(s)	SRQ	Component(s)
35	Client.ApplicationState, Client.ClientPersistence, Client.Layout	36	Client.ApplicationState, Client.Layout
37	Client.Layout	38	Client.Layout
39	Client.Layout	40	Client.Layout
41	Client.Layout	42	Client.Layout
43	Client.ApplicationState, Server.ApplicationService, Server.FortranModule, Server.SimulatorService	44	Client.ApplicationState, Client.Layout
45	Client.ApplicationState, Client.Layout	46	Client.ClientPersistence
47	Client.Layout	48	Client.Layout
49	Client.Layout	50	Client.ApplicationState, Client.ClientPersistence, Client.Layout
51	Client.ApplicationState, Client.ClientPersistence, Client.Layout	52	Client.ApplicationState, Client.ClientPersistence, Client.Layout
53	Client.ApplicationState, Client.ClientPersistence, Client.Layout	54	Client.Layout
55	Client.Layout	56	Client.ApplicationState, Client.ClientPersistence, Client.Layout
57	Client.Layout	58	Client.ApplicationState, Client.Layout
59	Client.Layout	60	Client.ApplicationState, Client.ClientPersistence, Client.Layout
61	Client.Layout	62	Client.ApplicationState, Client.ClientPersistence
63	Client.Layout	64	Client.ApplicationState, Client.ClientPersistence, Client.Layout
65	Client.Layout	66	Client.Layout
67	Client.Layout	68	Client.Layout
70	Client.Layout	71	Client.Layout
72	Client.ApplicationState	73	Client.ApplicationState, Client.ClientPersistence, Client.Layout
74	Client.Layout	75	Client.ApplicationState, Client.ClientPersistence, Client.Layout

SRQ	Component(s)	SRQ	Component(s)
76	Client.ApplicationState, Client.Layout	77	Client.ApplicationState, Client.Layout
78	Client.Layout	79	Client.Layout
??	Client.Layout	80	Client.Layout
81	Server.ApplicationService, Server.SimulatorService	82	Server.ApplicationService, Server.SimulatorService
83	Server.SimulatorService	84	Server.SimulatorService
85	Client.ApplicationState, Server.ApplicationService, Server.SimulatorService	86	Server.FortranModule, Server.SimulatorService
87	Server.FortranModule	88	Server.FortranModule
89	Server.FortranModule	90	Client.ApplicationState, Client.Layout, Server.ApplicationPersistence, Server.ApplicationService, Server.FortranModule, Server.SimulatorService
91	Client.ApplicationState, Client.Layout, Server.ApplicationPersistence, Server.ApplicationService, Server.FortranModule, Server.SimulatorService	92	Client.ApplicationState, Client.Layout, Server.ApplicationPersistence, Server.ApplicationService, Server.FortranModule, Server.SimulatorService
93	Client.ApplicationState, Client.ClientPersistence, Client.Layout	94	Client.ApplicationState, Client.ClientPersistence, Client.Layout
95	Client.ApplicationState, Client.ClientPersistence, Client.Layout	96	Client.ApplicationState, Client.ClientPersistence, Client.Layout
97	Client.ApplicationState, Client.ClientPersistence, Client.Layout	98	Client.ApplicationState, Client.ClientPersistence, Client.Layout
99	Client.ApplicationState, Client.ClientPersistence, Client.Layout	100	Client.ApplicationState, Client.ClientPersistence, Client.Layout
101	Client.ApplicationState, Client.ClientPersistence, Client.Layout	102	Client.ApplicationState, Client.ClientPersistence, Client.Layout

7.2 ADD to SRD

The following matrix lists which SRQs are implemented by each component.

Component	**SRQ(s)**
Client.ApplicationState	32, 33, 34, 35, 36, 43, 44, 45, 50, 51, 52, 53, 56, 58, 60, 62, 64, 72, 73, 75, 76, 77, 85, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102
Client.ClientPersistence	33, 35, 46, 50, 51, 52, 53, 56, 60, 62, 64, 73, 75, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102
Client.Layout	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, 44, 45, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 63, 64, 65, 66, 67, 68, 70, 71, 73, 74, 75, 76, 77, 78, 79, **??**, 80, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102
Server.ApplicationPersistence	8, 9, 10, 11, 12, 13, 90, 91, 92
Server.ApplicationService	2, 3, 4, 5, 6, 7, 8, 9, 14, 15, 43, 81, 82, 85, 90, 91, 92
Server.FortranModule	43, 86, 87, 88, 89, 90, 91, 92
Server.HTTPServer	1
Server.SimulatorService	43, 81, 82, 83, 84, 85, 86, 90, 91, 92