HEIDELBERG UNIVERSITY HEILBRONN UNIVERSITY

MASTER'S THESIS

Design and implementation of a dedicated planning and implementation software system using a driving simulator to support occupational therapy intervention and data collection in the context of studies

Author:
Hendrik GRAUPNER

Supervisors:
Prof. Dr. Diana SCHMIDT
Dr. Matthew CRISLER

January 30, 2014

Correspondence E-Mail Address:
hendrik.graupner@alumni.uni-heidelberg.de

Abstract

The aim of this master's thesis is the design and implementation of a dedicated software system, for planning and implementation of occupational therapy intervention and research studies, in a driving simulator environment.

In the first part, the concept based on user requirements is presented. It consists of architectural patterns and guidelines with the main focus on utility and application security. The result of this part is the design of a web application which supports integration in a clinical as well as a research environment.

The second part presents the reference implementation of the previously introduced concept. It was developed under a case study in a research facility which hosts a driving simulator. A close cooperation and the influence the researcher's experience led into a product which provides advanced usability for the target users.

In conclusion, the thesis validated the concept indirectly under a testing phase of the reference implementation. It provides the base for a follow-up project to refine the software product and extend the concept to different fields of application.

Keywords: therapy planning, study planning, data collection, driving simulator, web application, java

Contents

List of Figures

List of Tables

List of Code Listings

List of Acronyms

API	Application Programming Interface
CDS	Clinical Driving Simulator
CU-ICAR	Clemson University International Center for Automotive Engineering
CS	Control Software
CSV	Comma-separated Values
DAO	Data Access Object
DTO	Data Transfer Object
GWT	Google Web Toolkit
HHS	United States Department of Health and Human Services
HIPAA	Health Insurance Portability and Accountability Act
HTML	Hypertext Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
Java EE	Java Platform, Enterprise Edition
JAX-RS	Java API for RESTful Web Services
JAXB	Java Architecture for XML Binding
JDBC	Java Database Connectivity
JPA	Java Persistence API
JSNI	JavaScript Native Interface

JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
LTS	Long-Term Support
MVC	Model View Controller
MVP	Model View Presenter
NIST	National Institute of Standards and Technology
NSA	National Security Agency
OWASP	Open Web Application Security Project
POJO	Plain Old Java Object
SHA	Secure Hash Algorithm
SOA	Service-oriented Architecture
SOP	Same Origin Policy
SQL	Structured Query Language
SS	Simulation Software
RAM	Random-Access Memory
REST	Representational State Transfer
RPC	Remote Procedure Call
TDF	Task Definition File
TLS	Transport Layer Security
UI	User Interface
UML	Unified Modeling Language
URL	Uniform Resource Locator
XML	Extensible Markup Language
XSRF	Cross-Site Request Forging
XSS	Cross-Site Scripting

Glossary

client
: A client is the person who attends a plan. In the case of therapy a synonymous term is *patient* and in study terminology it is a *participant*.

criterion
: In this context a criterion is the configuration of a measurement to be a task implementation's termination condition. E.g. the measurement errors would be a criterion if the goal of an implementation session was achieving a *total error count of less than five* in one trial.

implementor
: An implementor is the person who supervises the implementation session with a client.

measure
: Measures are available data fields of a specific task (e.g. *mean speed in mph*).

measurement
: A measurement is the actual value of a measure at implementation time. E.g. a measurement of *mean speed* could be *54 mph*.

parameter
: Parameters are configuration options of tasks. E.g. a task could provide the choice between *easy*, *medium*, and *hard* for the parameter *difficulty*.

plan
: A plan is an assemblage of tasks one or a set of clients are supposed to attend. Another, more clinical, term used is *protocol*.

property
: Properties are used to specify subtasks more detailed. E.g. a task could

come with three subtasks which contain the same measures. To have a better way of distinguish them each could have a property called *target speed* with the values *35 mph*, *45 mph*, and *55 mph*.

task

A task is something a client is asked to do during therapy or a study. It can be a driving simulator scenario, a physical exercise (e.g. push-ups) etc.

1

Introduction

This chapter introduces the topic of this thesis. The reader learns what the background and the importance of the work that is presented in the following chapters, relevant terms, and the document's top level structure.

1.1 Background

In medicine one of the most important and at the same time challenging processes is the elaborate planning of procedures involving patients. The outcome is a protocol which consists of instructions for particular sessions. Those protocols form the base of effective and efficient medical care in clinical routine and high quality data in medical research. Because of the versatile nature of those environments this process of planning happens under many different circumstances. However, the goal of assembling the optimal arrangement of medical measures remains the same.

The processes of medical therapy and study planning are difficult and time-consuming. This is especially true if the protocol involves a high number of participants and complex exercises. In real life it is often the case that this planning process is not as computer-aided as it could be. The reasons are that software solutions are usually hard to integrate into processes and technical infrastructures and/or scare off potential users by requiring a lot of effort to familiarize. This causes the remaining of traditional approaches of paper based protocols or the use of suboptimal software products (e.g. text editing or table handling software).

A software solution that is capable of replacing the traditional approaches can help saving a clinician's as well as a researcher's most valuable resource: time. In addition it brings the advantages of modern information technology like re-utilization of previous work and improved digital data collection and handling. The latter is particularly useful if the process involves a data producing device that is capable of communicating with other computers. This thesis specializes in the Clinical Driving Simulators (CDSs), one particular type of such devices.

1.2 Motivation

Simulators have a growing importance in driving related occupational therapy, skill assessment, and research. "Driving simulators offer a safe alternative to road testing, and because they can be made readily available in laboratory settings, they allow for more careful control of clinical parameters than is possible with epidemiological study designs." [1] They are mainly installed in clinical and research environments.

The advantage of a research setup is the ability of data post processing. After a simulator session, stored data files can be retrieved from the simulator's hard disk and prepared for further analysis. In a clinical working process the data is required right when it occurs. This natural lack of flexibility is not yet fully compensated. It requires a data retrieval strategy and the according tools that are capable of fetching data in real time and providing it in a processable format immediately.

A real life want is the motivation for this thesis. The issues described are not exclusively true for this particular scenario but for other clinical data producing systems, too.

1.3 Case Study

This thesis is practically oriented and aims to develop a solution for a real life want. The cooperation with the human factors research division of Clemson University International Center for Automotive Engineering (CU-ICAR)[1] provides access to the field of application.

[1] `http://www.cuicar.com/`

One of the researchers' tasks is the development and testing of new scenarios on the DriveSafety CDS-250 (see figure 1.1) hosted there. The CDS is a complex device that represents the cockpit of a car, simulates interactive operations of the car in a variety of driving environments, and also supports other interactive clinical activities. It includes multiple computers to run the simulation and fetch occurring data.

Figure 1.1: The DriveSafety CDS-250 main components are a car's cockpit, three monitors, and computers in its base. An additional computer device (e.g. a laptop) is used to control the system.

The increasing complexity of studies involving the driving simulator generated the need for a computer-aided solution to support study planning and implementation processes. This thesis uses the CDS-250 laboratory at CU-ICAR as an exemplary case study for a potential application of its concepts. In return the reference implementation, the Protocol Manager, is customized to fit the particular needs and expectations of this research division. In addition this work profits from the experience of the interdisciplinary team at the junction point of psychology, engineering, computer science, and medicine.

1.4 Objective

This thesis' objective is the design and proof of a software concept that supports the planning and data collection processes in clinical therapy and medical research, involving a driving simulator. To achieve that, the core functionality is support of the work of clinicians and researchers in a way that increases time efficiency and assists error avoidance in complex scenarios. Even though the environment of interest involves a driving simulator the solution requires to be flexible enough to not be restricted to this device to support a variety of different setups and different data sources in a particular setup. The reference implementation provides an example and proof of the concept developed in cooperation with end users.

1.5 Document Structure

This thesis consists of the following chapters subsequent to the introduction.

Chapter 2 - Basics introduces terms and describes the current procedure of the case study. If the reader is not familiar with the case study's setup, it is recommended to begin reading with this chapter.

Chapter 3 - Concept introduces the software concepts. It shows the process of designing the system and the resulting architecture. This concept tries to solve the existing issues and forms the base for the reference implementation.

Chapter 4 - Implementation presents the implementation of the Protocol Manager. This chapter's goal is to impart the structures of the software and explain key components using code examples. Another focus is explaining user guidance using User Interface (UI) screenshots.

Chapter 5 - Conclusion critically discusses the results in terms of achieving of the goals. Finally, an outlook concludes this thesis by giving suggestions for followup projects.

2

Basics

This chapter describes pre conditional knowledge for the thesis. An introduction of terms used through the whole document is a complement to the glossary on page VII. The chapter's main part is a description of the case study's current driving simulator procedure, to impart basic knowledge of the environment and understand the importance of the following work.

2.1 Terms

This document provides content at the junction between medicine and computer science. Since each of those disciplines have their own terminology, it is necessary to clarify the meaning of terms.

This thesis' main subject is the planning of clinical therapy and medical research protocols. The term *plan* is used synonymous with protocol. While from the code point of view "a *planner* creates a plan", a clinician might say: "a therapist creates a protocol".

A person who attends a therapy session is typically a patient or, in the case of a research study, a participant. The term that describes both is *client*. This client (person) should not be confused with the client (software) that is a common part of a web application design pattern in software development.

The exercises a client executes during an implementation session of a plan are called *tasks*. A task can be of physical nature (e.g. push-ups) or involve a device that measures certain target values (e.g. heart rate while running on a treadmill for five minutes). As these examples illustrate, tasks typically produce data that

is stored for later use (e.g. analysis). A set of measurements is called implementation *result*. Results are typically assembled plan-wise or client-wise.

2.2 Current CDS Procedure

The DriveSafety CDS-250 was developed for a clinical operating environment to assist occupational therapy and assess driving skills after treatment. A secondary use is the performance of driving related research studies. The following describes the CDS procedure for research studies at CU-ICAR.

An important part of the preparation of simulator sessions is the creation of plans. A plan describes the procedure of therapy, assessment or a study, involving a collection of clients. The creation requires expertise, experience, and knowledge of the client's as well as the simulator's capabilities. In addition clinicians always bear a high responsibility for patients' health and well-being. For those reasons plan creation is typically the work of a lead therapist.

	A	B	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	A
1	Hand Control Comparison Criteria															
2																
3		Criteria	20	21	22	23	24	25	26	27	28	29	30	31	32	3
4			HC20_T1	HC21_T1	HC22_T1	HC23_T1	HC24_T1	HC25_T1	HC26_T1	HC27_T1	HC56_T1	HC29_T1	HC55_T1	HC31_T1	HC32_T1	HC3
5	ReactionTimerSteering	Fastest Avg. Time (s)	0.592	0.508	0.492	0.567	0.542	0.596	0.517	0.521	0.479	0.554	0.488	0.517	0.846	0.
6		max-()	0.254	0.338	0.354	0.279	0.304	0.250	0.329	0.325	0.367	0.292	0.358	0.329	0.000	0
7	z-score		-0.728	0.424	0.643	-0.385	-0.043	-0.783	0.300	0.245	0.821	-0.207	0.696	0.300	-4.211	0
8	GasAndBrakeEasy	Fastest Time (s)	44	43	42	41	44	43	46	45	47	42	49	43	44	
9		max-()	5	6	7	8	5	6	3	4	2	7	0	6	5	
10	z-score		0.234	0.453	0.671	0.889	0.234	0.453	-0.202	0.016	-0.420	0.671	-0.856	0.453	0.234	0
11		Number of Bonks	0	0	0	0	0	0	0	0	0	1	0	0	0	
12		max-()	1	1	1	1	1	1	1	1	1	0	1	1	1	
13	z-score															
14	GasAndBrakeMedium	Fastest Time (s)	45	47	46	45	48	47	49	48	49	44	49	44	45	
15		max-()	10	8	9	10	7	8	6	7	6	11	6	11	10	
16	z-score		0.684	0.008	0.346	0.684	-0.329	0.008	-0.667	-0.329	-0.667	1.021	-0.667	1.021	0.684	1
17		Number of Bonks	0	2	0	0	0	0	0	0	1	0	1	0	0	
18		max-()	2	0	2	2	2	2	2	2	1	2	1	2	2	
19	z-score															
20	GasAndBrakeHard	Fastest Time (s)	48	51	48	47	53	52	54	51	52	46	60	48	49	
21		max-()	18	15	18	19	13	14	12	15	14	20	6	18	17	
22	z-score		0.696	0.356	0.696	1.076	-0.005	0.176	-0.185	0.356	0.176	1.256	-1.265	0.696	0.716	0
23		Number of Bonks	0	2	0	0	0	0	0	1	2	2	1	1	0	
24		max-()	4	2	4	4	4	4	4	3	2	2	3	3	4	
25	z-score															
26	ReactionTimerStoplight	Fastest Avg. Time (s)	0.717	0.675	0.475	0.575	0.546	0.575	0.529	0.658	0.504	0.546	0.571	0.567	0.633	0

Figure 2.1: The data collection sheet consists of a matrix of clients (columns) and target measures (rows). This example shows side-by-side comparison to determine clients' relative performances.

In the case study at CU-ICAR, the procedure currently involves a combination of

paper and digital text documents. While the actual plan is a list of tasks supplemented with instructions, data collection happens in Microsoft Excel sheets. Figure 2.1 (page 6) shows an exemplary section of a realistic data collection sheet which forms a matrix of clients (columns) and target measures (rows). As the numbers of columns and rows increase, data collection becomes more complex. In addition the current degree of complexity is a result from the past development of data collection requirements. In early studies all clients followed the same procedure, whereas today's studies require more complicated methods (e.g. side-by-side comparison to other clients' data).

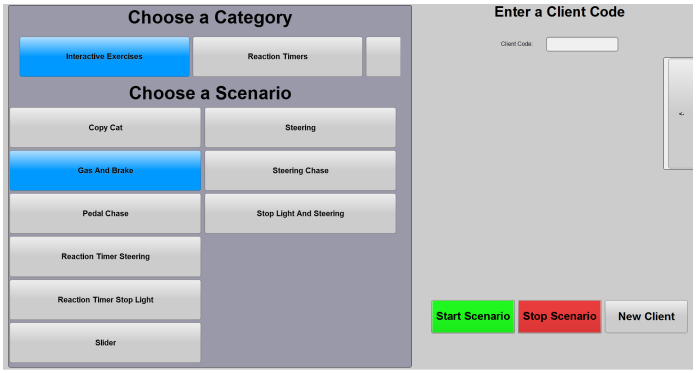
Side-by-side comparison is the origin for a criteria based procedure to assess the individual performance of a client. A criterion is a numerical surrogate for success. If it is met, the trial was successful and the procedure moves on to the next task. Otherwise the current task has to be repeated. Figure 2.2 shows a Microsoft Excel spreadsheet that calculates criteria for individual clients.

	A	B	AF	AG	AH	AI	AJ	AK	AL	AM	AN	AO	AP
1	**Hand Control Comparison Criteria**												
2													
3		**Criteria**											
4			HC32_T1	HC33_T1	HC34_T1	HC35_T1	HC36_T1	HC37_T1		HC39_T1	HC40_T1	HC41_T1	HC42_T1
149	* best is considered the least difference in avg speed from target speed and lowest standard deviation (per trial basis)												
150	** a speed zone is considered either 35, 45, or 55 mph section of driving scenario												
151			Stopped	Stopped	Stopped	Stopped	Stopped	Stopped	Stopped	Stopped	Stopped	Stopped	Stopped
152													
153	red = fail, green = pass												
154			HC32_T1	HC33_T1	HC34_T1	HC35_T1	HC36_T1	HC37_T1	HC38_T1	HC39_T1	HC40_T1	HC41_T1	HC42_T1
155	**ReactionTimerSteering (+/- .1 sec)**	training	0.65	0.55	0.57	0.58	0.59	0.54		0.60	0.50	0.66	0.53
156	**GasAndBrakeEasy (same bonks within 5 sec)**	training	0.53	0.45	49	0.47	0.46	50		0.48	0.44	0.5	0.46
157	bonks	no	0	0	1	0	0			0	0	0	0
158	**GasAndBrakeMedium (same bonks within 6 sec)**	training	0.54	0.46	51	0.5	0.48			0.5	0.5	0.53	0.48
159	bonks	no	0	0	0	0	0			0	0	0	0
160	**GasAndBrakeHard(same bonks within 7 sec)**	use	0.53	0.53	55	0.58	0.65			0.68	0.62	0.54	0.65
161	bonks	no	0	0	0	0	0			0	0	0	0
162	**ReactionTimerStoplight(+/- .1 sec)**	use	0.44	0.46	0.53	0.65	0.43			0.56	0.33	0.50	0.48
163	**StoplightAndSteering(no misses within 5 sec)**	MAYBE	18.0	15.0	21.0	16.0	14.0			16.0	13.0	18.0	15.0
164	**percentage correct**		100	100	100	100	100			100	100	100	100
165	misses	no	0	0	0	0	0	16		0	0	0	0
166	**SpeedControlStraight35 (within 3 stdev, diff from target avg. - within 3 mph**		35.7	35.9	36.0		35.6			35.5	36.9	35.0	35.7
167	std dev	training	0.1	0.1	2.3		0.7			0.3	0.1	0.9	1.0
168	diff from target	training	0.7	0.9	1.0	35.0	0.6	35.0	35.0	0.5	1.9	0.0	0.7
169	lane touches left 35	training	0	0	0		0			0	0	0	0
170	lane touches right 35	training	0	0	0		0			0	0	0	0
171	lane touches total	training	0	0	0	0	0	0	0	0	0	0	0
172	**SpeedControlStraight45(within 4 stdev, diff from target avg. - within 4 mph**		46.0	46.0			45.2			45.5	46.7	45.2	45.4

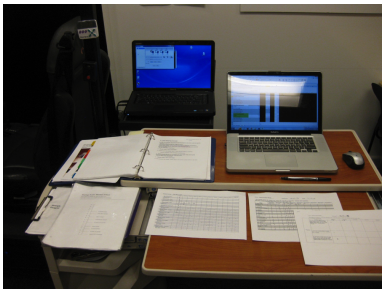
Figure 2.2: This spreadsheet is used for determining comparison criteria in a study. They are calculated for each client individually based on former performances.

The paper based instructions and notes, Microsoft Excel sheets for data collection, and a tablet computer running the simulator's Control Software (CS) (see

figure 2.3a) are at least three different systems a implementor has to use during a session while following this pattern: The implementor introduces the task by reading an introduction, displayed in the CS, to the client and answers questions. After that the client performs the task. At the end of each trial a list of measurements is displayed on the simulator's monitors and the implementor types them into the data collection sheet. According to the criteria setup and the actual performance, another trial of the same task may follow or the procedure moves on. During the whole session the implementor is not only responsible for data collection, but in the first place assuring the client feels well by periodically issuing a simulator sickness questionnaire and hand-writing the values on a data sheet. Figure 2.3b shows a typical set of documents an implementor has to handle while controlling the simulator.



(a) The simulator CS provides an interface to configure and control specific simulator scenarios.



(b) A lot of documents are involved in the process. Most of them are paper based and involve data sheets, questionnaires, and instructions.

Figure 2.3: An implementor has to handle several different tools during a simulator session.

This procedure is grown over the years and established in the daily routine of clinicians and researchers working with the simulator. It produces reliable, well documented data. However, with growing complexity of a criteria based data collection, problems have been noticed. Matrices, which are large and hard to read, are unavoidable using a Microsoft Excel based data collection. It is natural that typing in measurements during a high number of simulator sessions contains errors. The following error types were identified.

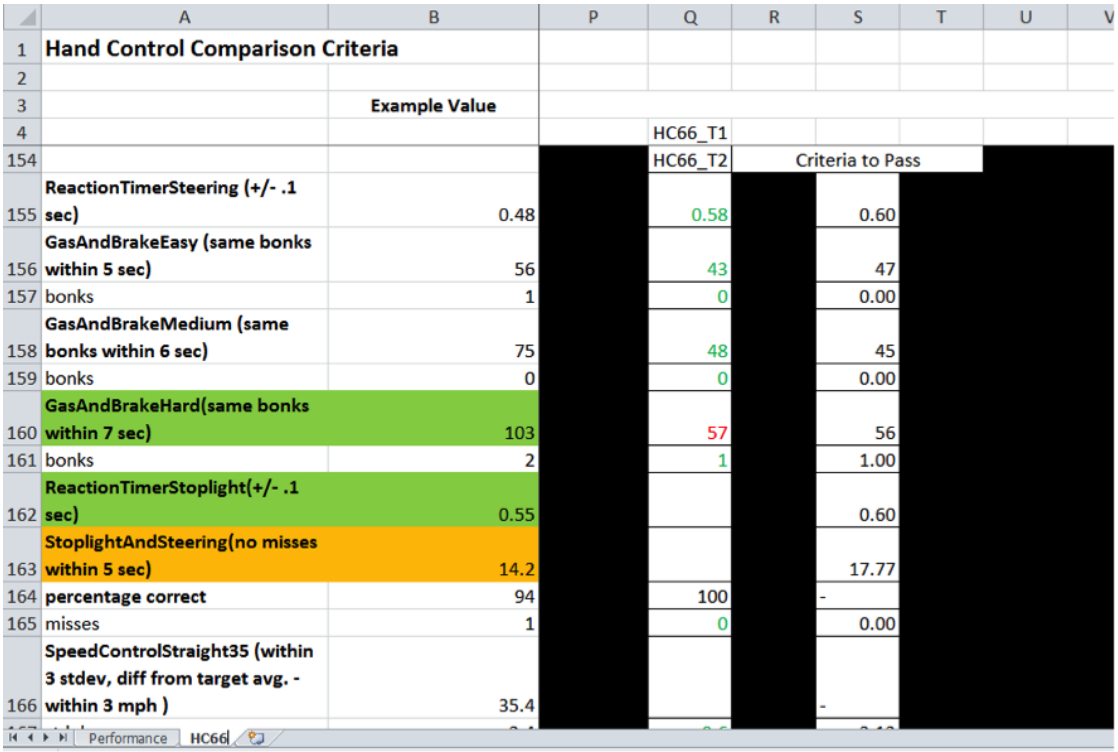
Data Entry Error Type 1

The individual criteria calculation in figure 2.2 (page 7) is based on data entered in figure 2.1 (page 6). For the calculation to work it is crucial that a client's data is entered in the correct column. If this is not the case, criterion calculation is invalid for this client. Columns AM through AP in figure 2.2 show examples of data that was entered into wrong columns initially, and corrected when the error was discovered. Two clients did move on in the procedure when they actually did not pass their individual criteria (see the red cells AM155 and AO155 in figure 2.1). They had to be subsequently excluded from the study and replacement clients were recruited instead, because every client could only participate once.

Data Entry Error Type 2

The second data entry error type is typographical error in a data field. For example entering a value of 0.50 while it was 50 in reality can make the implementor believe that the client passed a criterion when it actually did not happen. This error type occurred in row 156 in figure 2.2 (page 7). Values in this row were expected to be integers between 35 and 85 seconds. If mistyped values are not discovered right away, it results in invalid data and the client has to be replaced in the study.

To rectify both error types a modified spreadsheet was developed (see figure 2.4, page 10). Each client has not only an individual file which includes the client ID in the filename, but also all other clients are blacked out to prevent data entering into the wrong column. In addition the exact criterion is given for each row to remind the implementor of the correct syntax.



	A	B	P	Q	R	S	T	U	V
1	**Hand Control Comparison Criteria**								
2									
3		**Example Value**							
4				HC66_T1					
154				HC66_T2		Criteria to Pass			
155	**ReactionTimerSteering (+/- .1 sec)**	0.48		0.58		0.60			
156	**GasAndBrakeEasy (same bonks within 5 sec)**	56		43		47			
157	bonks	1		0		0.00			
158	**GasAndBrakeMedium (same bonks within 6 sec)**	75		48		45			
159	bonks	0		0		0.00			
160	**GasAndBrakeHard(same bonks within 7 sec)**	103		57		56			
161	bonks	2		1		1.00			
162	**ReactionTimerStoplight(+/- .1 sec)**	0.55				0.60			
163	**StoplightAndSteering(no misses within 5 sec)**	14.2				17.77			
164	**percentage correct**	94		100		-			
165	misses	1		0		0.00			
166	**SpeedControlStraight35 (within 3 stdev, diff from target avg. - within 3 mph)**	35.4				-			

Performance HC66

Figure 2.4: The modified spreadsheet tries to prevent both error types and is provided in a separate file for each individual client.

3

Concept

This chapter shows the concepts of the software design. It begins with an insight into the origins of the project and the problem. First the gathering of user requirements from several sources is shown. The conceptual solution to meet the identified requirements is the major part. It covers the software system's components and consists of concrete architectural patterns and implementation guidelines.

3.1 Requirements

The origin of the concept is the definition of requirements. Drawing on the driving simulator team's experience under the case study, helped to develop a real life oriented requirements specification.

3.1.1 Methods of Gathering

Personal interviews with with the researchers were the most important source of information. Together with a following observation, they revealed their expectations ins terms of usability as well as utility of a later software product and helped to understand processes, structures, and the working environment.

3.1.1.1 Personal Interviews

The strategy was to use the knowledge and experience of the driving simulator's research team to cover requirements of all involved perspectives. It is an interdisciplinary team of scientists and engineers, who work in cooperation with clinicians, patients, and other researchers. Meetings and interviews happened not only at the beginning of the project, but also during the development to synchronize expectations with the actual progress and identify possible problems early.

The initial interview series began with a potential research end user to determine the core functionality from the data analysis perspective and get an idea of the expected results. Mr. Evan Lowe, who is performing a study using the CDS, gave a strong insight in the expectations for the result data and possible features to prevent errors in the first place.

Interviews with a clinical experienced researcher, Dr. Johnell Brooks, helped to refine the core functionality, identify additional features to increase usability, and produce first UI sketches. Results are the UI's terminology, utility features, and usability aspects. The latter is especially important to develop a product that is accepted by the target users.

Technical requirements developed from dialogs with the team's software developer, Dr. Matthew Crisler, to fit the new software into the pre-existing technical infrastructure. A relevant part of the technical considerations was figuring out the possibilities and constraints of the simulator's data transmission abilities. In return we also had to clarify missing interfaces on the simulator side to allow full integration with the new software components.

3.1.1.2 Observation

The performed CDS session was a series of driving scenarios, supervised by Mr. Jimmy Bacon. He gave insight into the procedure of an implementor guiding a client. This observation showcased the workflow during a simulator session.

3.1.1.3 Statutory Regulations

The adherence of statutory regulations and laws regarding data security becomes mandatory as soon as real patient data is processed. To prepare the software's implementation for practical use they are part of the requirements.

Personal data, especially patient data, is always very sensitive. To protect people's privacy and prevent data abuse, certain statutory regulations exist in the USA. The applicable law for this scenario is the Health Insurance Portability and Accountability Act (HIPAA) because the target environment is hosted by a "[...] health care provider who transmits any health information in electronic form [...]" [2, §160.102(a)(3)]. Title II of the law defines i.a. national standards for administrative and technical procedures which involve health data. This makes title II of particular importance for this thesis.

3.1.2 Requirements Specification

The requirements specification is based on the standard of the IEEE [3]. It specifies the goals to be accomplished by a software design.

3.1.2.1 Product Perspective

The simulator infrastructure in the research laboratory has two major software components for potential interaction. They are the simulator's interactive Simulation Software (SS) and Control Software (CS). It is the author's assumption to find similar setups in other simulator environments. To make data transmission with those systems possible, a commonly usable server-side web interface is necessary.

The SS runs on three independent servers and operates the simulator by managing available driving scenarios, generating video output, and producing measurements during implementation sessions. The CS is used to interact with the simulator's operating system. Since communication with the SS is already up and running and the CS is self-developed, it is a valid option to limit the external interfaces to this software. This approach minimizes the effort to implement according interfaces because it involves only one external system. In addition it is conceivable that in other setups a CS is also easier to modify and extend than the SS.

Often the tasks performed during a session of therapy or study are not exclusively involving one source of data. For example a common addition to a single task or a series of tasks could be a questionnaire. From this follows that a software for data collection must be able to process data independent from its source. This requires a generic interface design, which is unspecific to the CDS. It further allows the concept and parts of the software to be reused in completely different environments which have the same goal of data collection.

3.1.2.2 Product Functions

The product's functions correlate to the identified use cases. Figure 3.1 shows the core use cases in a UML use case diagram representation. The actors are human users and the software system. Two user roles, planner or implementor, exist. Depending on the field of application a user may have both roles. A plan's implementation requires its previous creation.

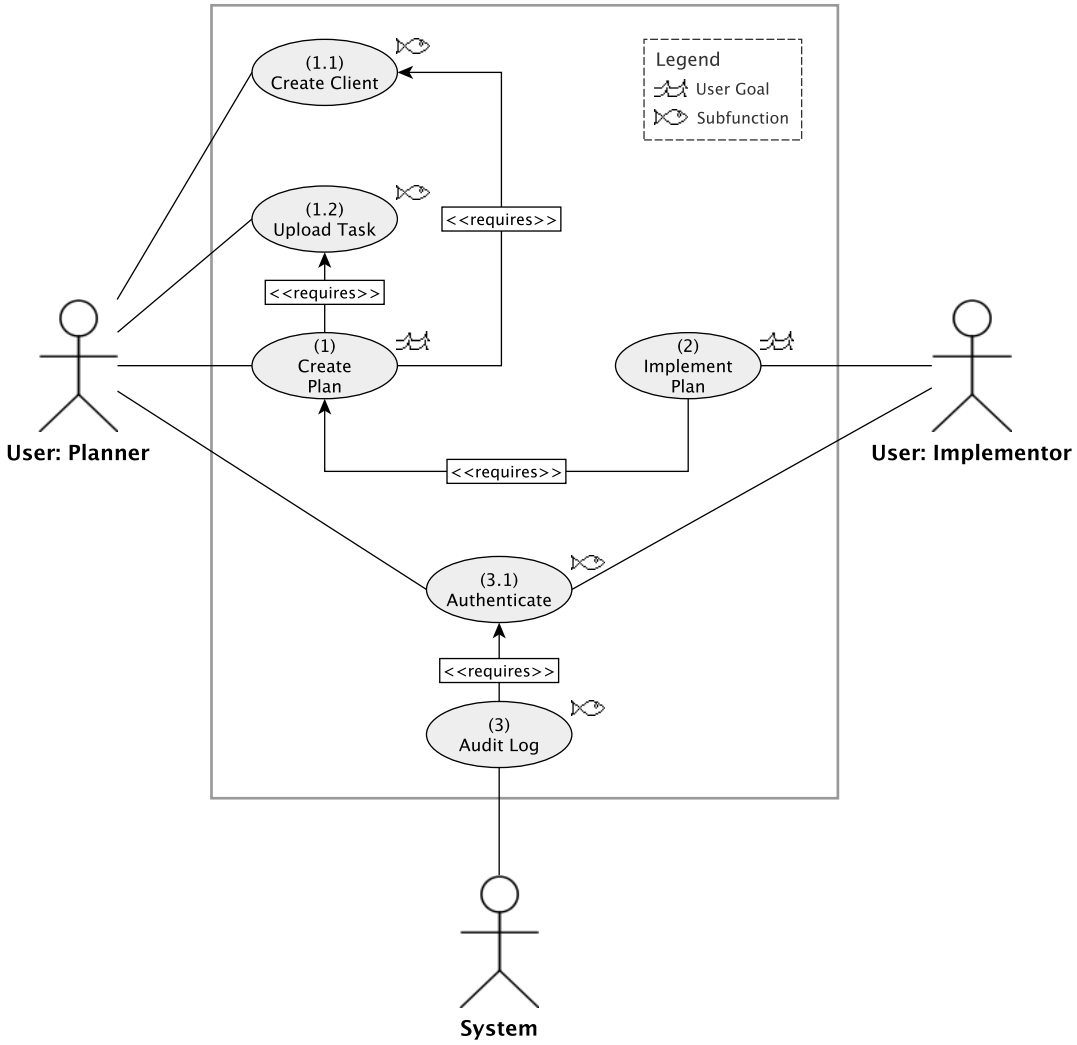


Figure 3.1: The system's basic functionality is represented in a UML use case diagram. It is annotated with goal levels by Cockburn's notation [4, pp. 62sqq.].

(1) Create Plan ! This is the origin of data collection. Its preconditions are declared as subfunctions.

1. The planner opens the plan creation form.
2. A name has to be set and tasks have to be added. Each task has to be specified in terms of its parameters and criteria. Assigning clients is optional and can also be done at implementation time.
3. When the user is done and all fields are validated the new plan will be persisted and available for implementation.

(1.1) Create Client ⁻ This is a subfunction of the plan creation and implementation. This function may also be available in the creation and implementation views. However, a client has to exist in the system before he can be assigned to a plan.

1. The planner/implementor opens the client creation form.
2. The only required field is an external ID (e.g. of a patient management system) which serves as an unique identifier. Other optional fields (e.g. names) can be entered.
3. When the user is done, all fields are validated, and the unique identifier is not duplicated the new client will be persisted and available for assignment to a plan.

(1.2) Upload Task ⁻ This is a subfunction of the plan creation and can only be done by a user in the planner role. Only existing tasks can be added. A task can not be edited or manually created using a form, because it would interfere with the intention of keeping the data synchronized with the according external system. The procedure is exporting a task definition file from its host system and importing it to this system.

1. The planner opens the upload dialog and chooses a path to the according file on his device.
2. If the file is valid and the task is not duplicated it will be persisted and available for plans.

(2) Implement Plan ! The implementation can be suspended before it is finished. If the user chooses to do so the current session is stored and can be resumed at a later time.

1. The implementor opens the plan implementation form.
2. He/she chooses an assigned client, assigns an existing one, or creates a new one.
3. From a list of available tasks one can be chosen to start data collection. According to the planner's choice the list might have a fixed order or not.
4. The implementor types in measurements trialwise and as they are available.
5. The input is validated, criteria are checked, and the user will be notified if the criteria are passed. This step may be repeated until passed or canceled.
6. When the implementor is done the session will be persisted and available for reviewing and possibly resuming.

(3) Audit Log – The system keeps track of data access and alteration using login information.

(3.1) Authenticate – This is a subfunction of audit logging. To identify the user, he/she has to log in with a unique username and a secret password before data access is allowed.

3.1.2.3 User Characteristics

Two types of users are intended to use the system. The first type is a *clinician* who will use the system for occupational therapy or skill assessment. The second type is a *data analyst* who will use the system for data collection, e.g. to support a study. Some users may be allocated to both types. For a software product to be accepted by clinical users, it is unavoidable to use a clinical terminology.

3.1.2.4 Constraints

For automatic data transmission the whole system is constrained by intranet access. Since the server application as well as external data sources are exclusively

accessible over the intranet, no functionality will be available on a workstation which is offline.

The quality of a plan is constraint by the quality of the task specifications it is based on. This means, only measurements provided by the specification files can be recorded at implementation time. Also, accuracy of values can not be higher than provided by external data sources.

The technical safeguards stated in NIST [5] are mandatory for this software project. Table 3.2 gives an overview over the relevant paragraphs (see appendix A, page 63) and the inferred requirements.

§164.312(a)(1)	unique usernames, automatic session termination, automatic data encryption and decryption
§164.312(b)	audit logging of any access, alteration or deletion of health data
§164.312(c)(1)	database encryption, versioned backups, and access control
§164.312(e)(1)	ensured transmission data integrity, transmission encryption

Table 3.2: The HIPAA obligates several requirements for applications that process patient related data. Those are the relevant paragraphs and their inferred requirements for this thesis' solutions.

Resulting from those safeguards the systems main functions can not be used without logging in. Due to the HIPAA undisputed authentication (e.g. via username and password) is necessary to access personal data. In addition this authentication will be the base of audit logging.

3.1.2.5 Assumptions and Dependencies

This requirement specification assumes the use in a driving simulator environment. It tries to lead into a concept that is not exclusive to this setup. However, it can not be assured to meet all requirements of a different, particular scenario.

It is further assumed that the use happens in the USA. Requirements regarding software security seek to satisfy regulations stated in the HIPAA and to issue the responsibility towards personal data of clients. However, different principles may apply in other countries. If results of this document are ever used outside the USA, the statutory regulations in this region have to be adapted first.

3.2 Architecture

This section presents the designed architecture of the software system. It serves as a guideline for software products that have the goal of compliance with the requirements specified in section 3.1.2 (page 13).

The thesis' duration of six months restricted the implementation of the whole concept because a simulator-side interface for communication with a new software system could not be completed in time. To still achieve the goal of providing a functional implementation by the end the development happened in two steps. The software temporarily works without direct communication with any external system at the expense of more manual user interaction. As a result the following architecture supports both steps of implementation.

The web application it is based on a client server architecture pattern. Figure 3.2 shows the architecture with two client applications: the planning client and the implementation client. While the planning client is used for the creation of study and therapy plans, the implementation client provides the required functionality for data collection. Both clients communicate independently with a web server application. The server has access to a database that stores the system's data (e.g. tasks, measures, etc.).

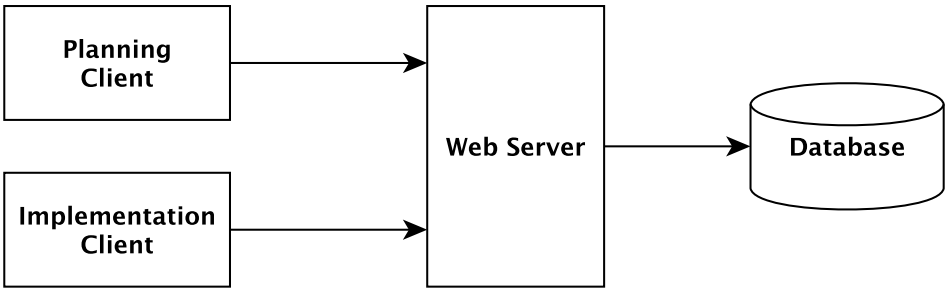


Figure 3.2: The software system's basic architecture consist of two client applications (for planning and plan implementation) and a server application with database access.

The reason for the client split is a separation of functionality to allow later integration of the implementation side into another software product, without the need to adapt the planning client. Another advantage is reduction of the application size a user has to download and consequently a higher performance. Users who only use one of the clients will not have to download the other one. The tradeoff is redundant code downloads which will come from the shared code base. Users who want to use both clients will download parts of the code twice.

3.2.1 Client Architecture

The fundamental design pattern for the clients' architecture, Model View Presenter (MVP) [6], is shown in figure 3.3. Its prevailing benefit is a clear separation between model and view. This allows changes in each of them without having to change the other one. The adaption happens in the presenter layer, which is at the same time easier to reuse across the software. It was chosen over the more traditional pattern Model View Controller (MVC) [7] for the reason of the strong decoupling focus, which also improves the unit testing capability.

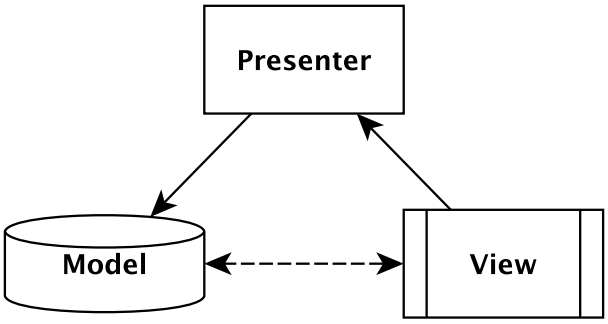


Figure 3.3: The Model View Presenter (MVP) design pattern separates model, presenter, and view. The arrows show associations between those layers.

MVP's variation that is used is the Supervising Controller pattern [8]. The model layer is simple and represents the abstract data model. It is presented to the user by the view layer which has a loose linkage to be easy replaceable (e.g. to add support for mobile devices). Views handle user input in terms of browser events, while higher logic is performed by presenters. A one-to-one-relationship between views and presenters allows each presenter to observe its view and react to events. In essence this means, the presenters update the model and views read the data from there.

Presenters need to exchange data between each other because views can not directly interact with other views. This cross-presenter communication uses the Event Bus pattern (see figure 3.4, page 20), which is a variation of the Observer pattern [9]. Instead of heaving multiple independent observer relationships one object, the event bus, broadcasts and filters notifications. Observers can subscribe to the bus for specific events. In this application the presenters are observers.

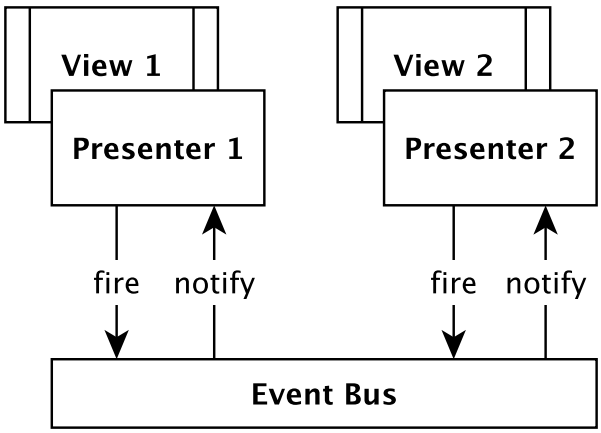


Figure 3.4: The event bus pattern defines a way of communication between units of the software that do not know each other. Any of them can send events to the bus and be a listener to any event of interest.

The final concept defines a standalone client application for planning and an integrated client for implementation. In this context integrated means to extend an already existing CS of the simulator for the according functionality. The integrated solution takes advantage of the fact that a dedicated CS already has to have communication with its simulator set up. This solution creates the requirement for the server application to communicate with two potentially totally different types of clients. Two separate code bases for the clients may lead to inconsistency in formats, terminology etc. which has to be taken into consideration while development of the software.

The interim concept specifies two standalone client applications: the planning client and the implementation client. While the planning client is used for the creation of study and therapy plans, the implementation client provides the required functionality for data collection. For the final concept an implementation client is not mandatory but can still be useful. A plan may contain physical tasks (e.g. sit-ups) together with simulator tasks. In this case the use of the simulator's CS may more inconvenient than useful. Another scenario is an environment where no dedicated CS exists or it can not be extended.

Some functionality overlaps both clients. In the case of the standalone implementation client it is a good programming paradigm to share code. The advantages are high maintainability, avoidance of redundant code, and easier testing. Shared functionality involves login functionality, user preferences and management, and layout elements.

3.2.1.1 Planning Client

The planning client's function is the use case of plan creation. Its preconditions are introduction of clients and tasks to the system. As a consequence this application has three segments: client management, task management, and plan creation. The server's interface for the planning client functionality must provide authentication and transmission of the three entities (client, task, and plan) in terms of storing and requesting.

Client management begins with a form for creation. It must gather identifying information such as names and an ID to match it to an external list or system. Once created editing must be possible. Deletion can only happen if no data is recorded for the person, in order to prevent data loss. An overview lists all clients known to the systems and allows browsing the information of each entry.

The base of task management is uploading of a Task Definition Files (TDFs) from the user's computer to the server. Browsing and deletion mechanisms are the same as in client management. Editing is not allowed, in order to prevent inconsistency with the task specification on its external host device.

Plan creation basically consist of a form to assemble tasks and configure their criteria. Optionally clients can be assigned at this point. This function is not exclusive for the planning side to increase an implementor's flexibility. Reviewing, editing, and duplicating are mandatory functions to ease the work with plans. Deletion can only happen if no data has been collected yet.

3.2.1.2 Implementation Client

The implementation client provides plan implementation functionality, which is basically data collection. Implementation requires prior creation of the plan. The goal is to collect data and make it available for further analysis. The required server interface functionality consist of authentication and transmission of clients, tasks, plans, and implementation results.

The implementation form presumes prior selection of a plan and a client. It provides a list of available tasks and for each task data field for measurement input. In conjunction with data collection criteria checking is performed. This way the implementor receives information about the client's passing status in real time.

Subsequent to implementation the results become available for review. If the data collection is not complete, a resume function allows to split implementation

into multiple sessions.

3.2.2 Server Architecture

The server application's purpose in the software system is to provide services to the clients. Those service requirements deduce from the clients' expected functionality introduced in the previous sections 3.2.1.1 and 3.2.1.2 (page 21).

Figure 3.5 shows the server's Service-oriented Architecture (SOA) [10] which is based on a layered structure. Each layer has a clear purpose. The privacy increases from left to right: web services are publicly available for requests while the database is the most sensitive part of the software system. Functionality is revealed using web services. The benefit is a loose coupling to client applications. Any other software that is capable of using the same protocol and knows the expected formats can communicate with the server application. The network protocol used for communication is today's standard, HyperText Transfer Protocol (HTTP).

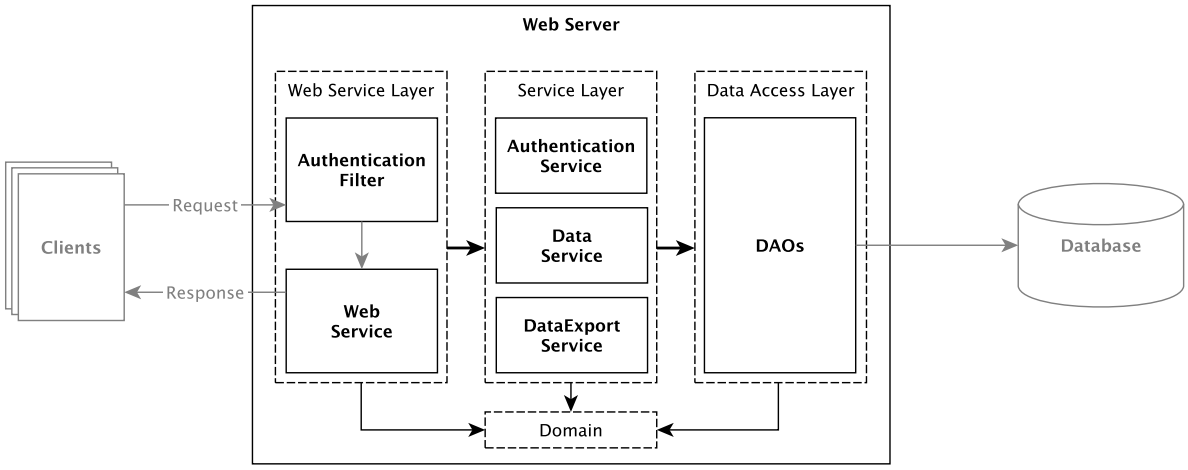


Figure 3.5: The web server architecture has three internal layers: web service layer, service layer, and data access layer. Via a public interface communications with client applications and has private access to a database.

Requests arrive at the server's external interface: the *web service layer*. Independent from its source and content any request will be intercepted by an *authentication filter*. If it is not an authentication request, a valid session has to exist or the filter refuses it. Valid requests are forwarded to the according web service

they addressed. Web services extract and check data from incoming requests and pass it to the according internal services. Functions required on the web interface are:

- Authentication with username and password
- Client storage, editing, deletion, and demanding
- Task uploading, deletion, and demanding
- Plan storage, editing, deletion, and demanding
- Implementation result storage, extension (after a suspended session), and demanding

The *service layer* contains the application's higher logic. It is composed of modules for authenticating users (authentication service), database access (data service), and assembling of downloadable documents (data export service). The authentication service uses session management to store and provide user data. Data related requests are processed by the data service. It communicates with the data access layer to gain indirect database access. Pre-processing of raw data (e.g. TDFs) happens here. The data export service generates exportable files from database entries. It is used to assemble implementation results for further analysis.

The server's base is the *data access layer*. Data Access Objects (DAOs) [11] communicate with the database, retrieve required data, and translate it to domain objects. The domain is a dependency of all layers and has a simple design for holding data. It contains no program logic.

3.2.2.1 Support of External Systems

All required functionality is already part of the concept's first version for the reference implementation. However, support for external systems, mainly a simulator CS, is required beyond this version. This concept suggests a Representational State Transfer (REST) [12] based implementation, which provides the benefits of simplicity, lightweight structures, and a resource efficient communication paradigm. It is also widespread and supported by frameworks in any programming language. In the case of the external CS being not flexible enough or extensible at all, in practice the server application may have to be adapted accordingly.

3.2.3 Input Validation

Input validation is one of the significant advantages dedicated software has compared to a pure document based solution. Two types of validation are part of this concept: validation of persistent entities and validation of data fields during data collection.

3.2.3.1 Entity Validation

When the system persists an entity (e.g. a task definition) it is important to check for structural and content-related errors first. Otherwise invalid data may be persisted which will likely lead into data retrieval effort for the system's administrator or data loss. When an error is noticed during or before the process of persistence the user is generally able to correct it right away.

The UML activity diagram in figure 3.6 (page 25) illustrates the generalized procedure of entity creation. Editing is very similar because the only difference is the starting point on the client application side. Validation has to happen on both, client and server, sides. The client side check can prevent unnecessary data transmission if the server will refuse the persistence request anyway. The server side check is more important because it prevents, independently from client's validation, the persistence of corrupt data.

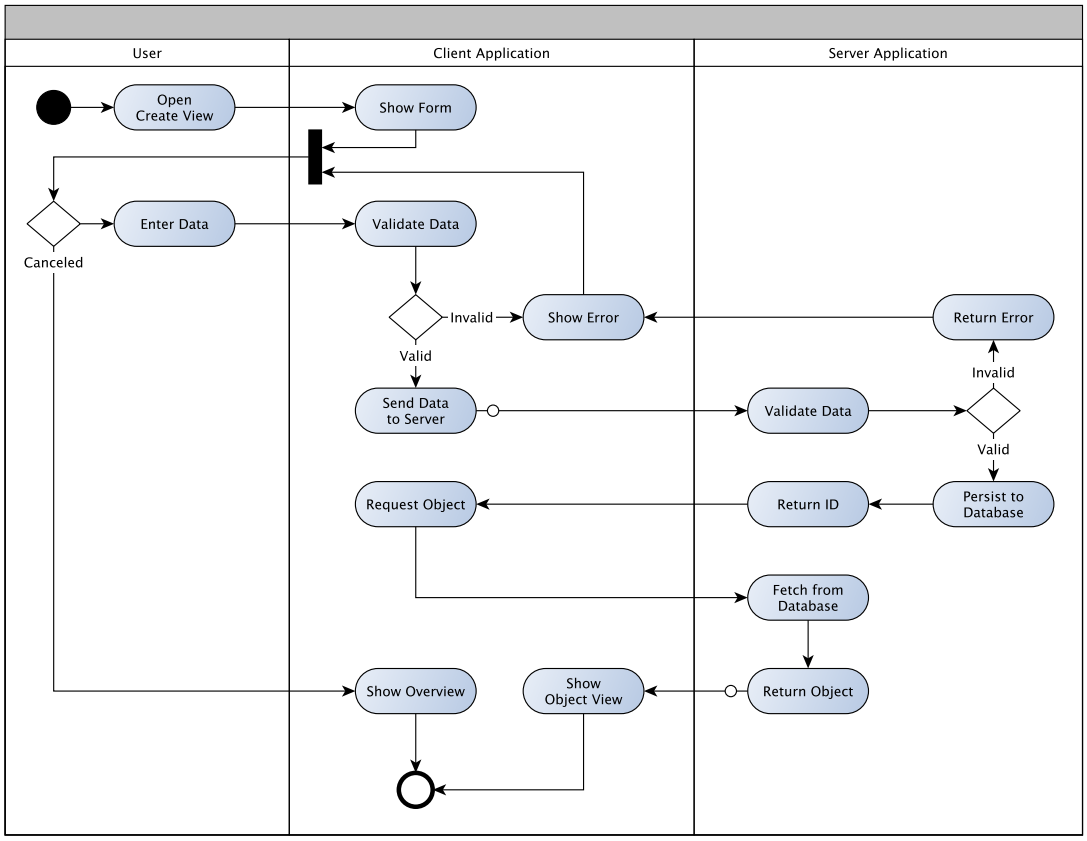


Figure 3.6: This activity diagram shows the procedure of entity creation. Validation is an important part and happens on client as well as server application side.

The actual validation parameters are very dependent on the particular entity and the system's domain structure. They have to include checks for collision of unique identifiers, violation of non-empty data fields, occurrence of invalid values, etc.

3.2.3.2 Standalone Implementation Client: Data Field Validation

A mechanism that helps to prevent typing errors (e.g. misplaced decimal points) during data collection is input validation of each data field. It can only detect violations of the anticipated range. Because measurement input validation is not necessary for an automatic data collection process, this part of the concept only applies for the standalone implementation client scenario.

The base for this kind of validation is the specification of a range of validity for

every single measure in a task. For example a measure *total time* could have a range of *[1;360]* in *seconds*. This means if the user tries to record a trial with a time of less than 1 second or more than 360 seconds the system will display a warning. This type of validation has to happen on client application side and in real time. Informing the user about a violation of the anticipated range provides a chance to correct the error right away when the data is still available.

3.2.4 TDF Format

Generally formats, for example of data transmission are decision during the software development process, based on data models etc. In this particular case an external system is addressed. This concept standardizes the format of the Extensible Markup Language (XML) representation of TDFs (see appendix B.1, page 64). This way it can be assured that task uploading works across different implementations of the same concept.

Other formats, like implementation result files, are to specific to a certain environment to standardize them.

3.2.5 Security Considerations

Especially for the reason that patient data is processed, any identity theft or data leak will cause major legal and trust issues to the software's operator. To make sure the software system is protected from potential threats, those threats have to be identified first.

The Open Web Application Security Project (OWASP) Foundation is an international charitable non-profit organization with the goal of enhancing application security all over the world. One of their services is the publishing of an annual list of the ten most widespread web application security breaches. The Top 10 of 2013 [13] listed the following risks.

1. Injection
2. Broken Authentication and Session Management
3. Cross-Site Scripting (XSS)
4. Insecure Direct Object References
5. Security Misconfiguration

6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross-Site Request Forging (XSRF)
9. Using Components with Known Vulnerabilities
10. Unvalidated Redirects and Forwards

Risk 10 is not considered relevant since the application does not require to redirect and forward at all. Any communication with other systems will be performed by the server application using the Application Programming Interfaces (APIs) defined in this concept. The risks 2, 4, 5, 6, and 9 have to be taken into account during and after implementation because they follow from implementation details and specific use of technologies, libraries, platforms etc. They can be revealed via code review and stress testing.

The relevant security breaches for this concept are 1, 3, 7, and 8. The following provides guidelines to minimize those risks.

3.2.5.1 Injection

The relevant interpreters to take into account are the database query builder, the JavaScript engine, and the user's web browser. Other interpreted technologies like LDAP are not necessary by concept and operating system commands should not be used at all.

The safest way to prevent database injection is using prepared statements and avoiding wildcards in their content [14, pp. 132-136]. Prepared statements predefine the query and escape "tainted" runtime parameters. They are available in any programming languages. JavaScript and HTML injection are covered by XSS, another type of code injection.

3.2.5.2 Cross-Site Scripting (XSS)

Preventing JavaScript and HTML injection, so called XSS, is an important task of browser and browser plugin developers. However, there are some countermeasures Shema [14, pp. 67-78] in the hands of developer. Most of them are about avoiding to write insecure code.

The code should provide a clear separation of trusted input (e.g. hard-coded constant strings) and untrusted user or external source input. Untrusted strings

should never be passed as unescaped JavaScript or HTML code to be executed by an interpreter. When in comes to JavaScript, JavaScript Object Notation (JSON) content is especially sensitive. Any JSON response from the server should be considered untrusted to be safe against attacks involving server response forgery or other kinds of manipulation.

In addition input validation of Hypertext Markup Language (HTML) forms, URLs etc. prevents malicious code from compromising the system in the first place but should not be considered safe without proper output validation and escaping. Shema [14, p. 78] sums it up: “The primary line of defense lays within the web sites themselves, which must filter, encode, and display content correctly and safely in order to protect visitors from being targeted by these attacks.”

During implementation the developer also has to consider the safety libraries he uses and potential security risks coming with calls of powerful JavaScript functions.

3.2.5.3 Missing Function Level Access Control

An usual mistake programmers make is implementing access control only on the client level of their software system. Since manipulation of a client application is by far easier than manipulating the server, this flaw may cause unauthorized access of server functionality. To prevent this any request has to be authorized on server level again.

This concept solves the issue by dedicating an authentication filter which checks session validity and user identity for every request. However, it is the developers responsibility to not reveal any functionality, which requires authentication, in a way that circumvents the authentication filter.

3.2.5.4 Cross-Site Request Forging (XSRF)

A common way to prevent XSRF is to include a secret session token in every request’s content. This token may be a copy of the session ID or the hash of the session cookie’s content. It prevents an attacker to circumvent the web browser’s Same Origin Policy (SOP). The SOP protects a cookies content from unauthorized access. However, this measure alone does not protect from XSRF [15]. A proper session expiration mechanism is mandatory. The mechanism expires a session after a fixed amount of time or renews the session ID if the user is still active.

4

Implementation

The Protocol Manager is the concept's reference implementation. Its goal is to prove the concept in terms of utility and make it testable. At the same time it will be used in CU-ICAR. This chapter introduces the environment, implementation details, and user guidance. It conveys a basic understanding of the software while focusing on interesting aspects, since a whole code listing is not possible.

4.1 Implementation Environment

The development environment consists of decisions made by the author to implement the designed concepts. The following introduces technology and tools choices which led to the software product.

4.1.1 Programming Language: Java

The development of the Protocol Manager started from scratch and did not have constraints in terms of technologies. For that reason the first decision had to be the appropriate programming language. The focal points were data security, maintainability, and extensibility. Today's common programming languages like C, C++, Java, Python, Ruby etc. can all meet those criteria. It depends more on the frameworks used and the written code in particular. The author therefore chose Java because this is the programming language he is most familiar with. Experience with a particular technology is crucial for creating secure code. Other advantages of Java are:

- Certain long-term support (by Oracle)
- JVM is a controlled and secure runtime environment
- Extensive, use case specific, standard libraries (e.g. Java EE for web development)
- Large variety of reliable third party libraries and frameworks
- Platform independence

There are also notable disadvantages which carry virtually no weight in this scenario. Because the JVM comes with computation overhead the performance may be worse than native languages like C. This is not an issue because the Protocol Manager is a low performance application. From the user's perspective, network latency is far more noticeable than the server's computation time of any request. Another risk to mention is the possibility of reverse engineering. A compiled Java file can be used to reconstruct the original source code. This is only an issue, if the code is not secure, because reverse engineering helps an attacker to find and abuse security issues. However, since the main focus of this concept and implementation is security, it does not rely on keeping source code secret.

4.1.2 Frameworks

The following frameworks available on the Java platform have been chosen to integrate certain functionality. Focal points to make those choices were security, lightweight code, and long-term success in the programming community.

4.1.2.1 Google Web Toolkit (GWT)

For web application development in Java a variety of frameworks do exist. As a part of this decision it should be considered that wide spread frameworks are usually more stable and secure, because large communities involve experience in best practices and reliable implementation patterns.

By the time this decision was made Java-Source[1] listed 67 different web application frameworks. The decision was in favor of GWT[2]. Its most outstanding

[1] `http://java-source.net/`
[2] `http://www.gwtproject.org/`

benefits are the variety of compatible libraries, useful IDE tools, and the innovative compiler. From that follows a possible high level of security, efficient testing and debugging options, as well as improved maintainability because the underlying client code is almost pure Java. The framework provides server application libraries which allow an efficient communication with GWT client applications.

4.1.2.2 Jersey and Jackson

One important requirement of the server application is compatibility with REST-speaking clients. The Java standard library defines a standard for that, called Java API for RESTful Web Services (JAX-RS) [16]. Its reference implementation is the framework Jersey[3]. It is lightweight and integrates well with Jackson.

Jackson[4] is a lightweight Java JSON processor. In combination with Java's standard implementation of Java Architecture for XML Binding (JAXB) [17] it provides a REST web service the ability to support JSON as well as XML data object transmission.

4.1.2.3 EclipseLink

The Java standard library provides the Java Persistence API (JPA) [18] standard for seamless mapping between Java objects and SQL. It requires Java Database Connectivity (JDBC) [19] on the database side to provide a database specific Java interface. The reference implementation of JPA 2.0 is EclipseLink[5] which is lightweight and robust and therefore meets this project's requirements.

4.1.2.4 Log4j

The logging framework of choice is Log4j[6]. It provides a proper number of logging levels, a useful message filtering system and the ability of logging to different files and database tables at the same time.

[3] `https://jersey.java.net/`
[4] `https://github.com/FasterXML/jackson`
[5] `http://www.eclipse.org/eclipselink/`
[6] `http://logging.apache.org/log4j/`

4.1.2.5 JUnit with EasyMock

JUnit[7] is a powerful testing framework for Java. It can be used for integration tests as well as unit tests. For the latter EasyMock[8] is a helpful extension which improves the mocking capability to isolate units of code. GWT supports JUnit for client side testing which makes it an easy choice.

4.1.3 IDE: Eclipse

The perhaps most wide spread open source IDE for Java is Eclipse[9]. It has a large community and an enormous amount of extensions. For example the GWT development environment fully integrates with Eclipse. In addition it comes with a lot of features that make a developer's life much easy. Some of them are: syntax coloring, pre-compile checking, code organizing support etc.

Other tools used in combination with Eclipse to organize builds, dependencies and the code handling are: Apache Maven[10], Jenkins[11], and Apache Subversion[12].

4.1.4 Hardware in CU-ICAR

The productive server at CU-ICAR is a physical machine using an Intel Core2Duo E7300@2.66GHz processor and 2GB RAM, running Ubuntu Server 12.04 LTS. It is an Apache HTTP Server that uses Apache Tomcat as its Java servlet container. The MySQL database is hosted on the same machine.

[7] http://junit.org/
[8] http://easymock.org/
[9] http://www.eclipse.org/
[10] http://maven.apache.org/
[11] http://jenkins-ci.org/
[12] http://subversion.tigris.org/

4.2 Implementation Details

The implementation follows the concepts and guidelines described in chapter 3. This sections gives an overview over the software product developed for the use in CU-ICAR. This document does not contain the entire software which consists of more than 22,000 lines of code. Exemplary source code extracts of key components are shown.

4.2.1 Modular Code Structure

The modular structure is the base for efficient and maintainable code. Since all components are composed of Java code it is easy to have shared code bases (see figure 4.1). Both clients share some model classes, logic, and view components. To simplify transmission between server and clients a shared code module exists. It includes Data Transfer Object (DTO) [20] classes and Java interfaces of the web service.

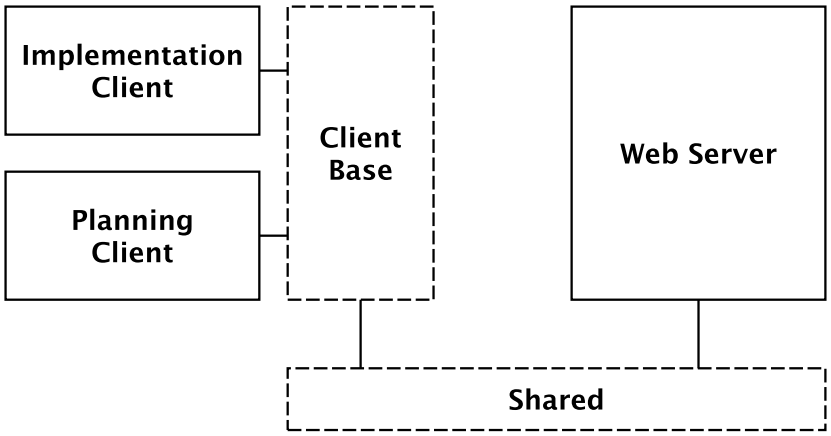


Figure 4.1: The modular code structure provides a non-redundant and efficient organization of the source code.

This division of codes allows to build all separate applications with only the necessary code included. For example the implementation client build is assembled of the implementation client module, the client code base module, and the shared code module. The web server includes the web server modules and the shared code module.

4.2.2 Client Implementation

The MVP is the architectural pattern of choice (see section 3.2.1, page 19). It harmonizes well with the GWT framework, which has a lot of MVP support included. The most relevant feature in this context is the addition of browser history management via the Activities and Places framework. According to the GWT-Project [21] in this context "an activity simply represents something the user is doing" and "a place is a Java object representing a particular state of the UI". This means that activities are equal to presenters and places are used to initialize a particular instance of the currently required activity. Figure 4.2 illustrates this concept using an exemplary extract of the Protocol Manager's class structure.

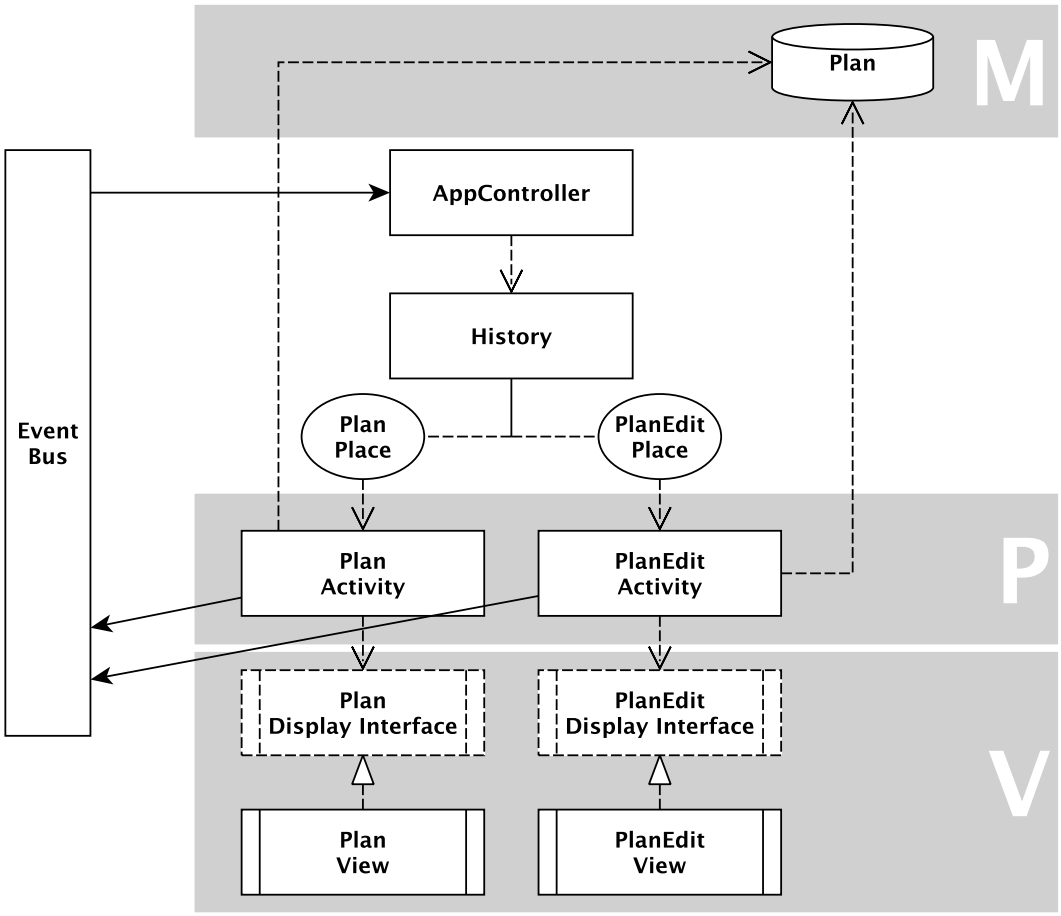


Figure 4.2: This illustration of the MVP pattern uses the example of plan display and edit functionality. The diagram is inspired by Ramsdale [22].

Model classes are simple Plain Old Java Objects (POJOs) which contain the data to be displayed in UI fields. The view layer is loosely linked using Java interfaces. Their implementations are exchangeable via configuration. This way the application supports different view sets (e.g. desktop displays, mobile device displays etc.). At the time of publishing the application included a set of desktop view implementations.

A common technique to design browser UIs in GWT is the UiBinder framework. The following statement describes how it works.

> "At heart, a GWT application is a web page. And when you're laying out a web page, writing HTML and CSS is the most natural way to get the job done. The UiBinder framework allows you to do exactly that: build your apps as HTML pages with GWT widgets sprinkled throughout them." [23]

The example of an XML file (see appendix C.1.1, page 65) and a Java file (see appendix C.1.2, page 66) illustrate the technology. The `ui:field` attribute of an XML element is the hook for a Java code element, which is annotated with `@UiField`.

4.2.3 Server Implementation

The Protocol Manager's server application is based Java, too. Plenty of frameworks and libraries are available for web application development and a lot of them are already included in the Java Platform, Enterprise Edition (Java EE) [24]. This section describes the implementation of the server concepts developed in section 3.2.2 (page 22) using Java EE servlets, GWT, and various frameworks (see section 4.1.2, page 30).

4.2.3.1 Domain

Domain classes are the data representation of database rows and consist of simple POJOs. The goal of modeling the domain is mirroring real structures on the level of abstraction suitable for the software's purpose. The center of the Protocol Manager's domain is a study/therapy plan which results in an implementation. Figure 4.3 (page 36) shows the UML class diagram on the highest level. It gives an overview over relations between the major entity classes of the project. A `Plan` consists of `TaskConfig` and `Client` instances. `TaskConfig`

is the class that specifies a setup of a `Task` for a specific `Plan`. The implementation of a `Plan` produces a `PlanResult`. It consists of `TaskResult` instances in combination with `ImplementationSession` instances to model the procedure implementation. Persistence of a `Plan` will store the `User` reference of its "planner" and every `ImplementationSession` contains a reference to the person who was logged in as "implementor". The `User` does not have fixed roles since no distinct rights are necessary for this field of application. This class essentially consists of login credentials (username and password) and the user's identity in terms of the person's names.

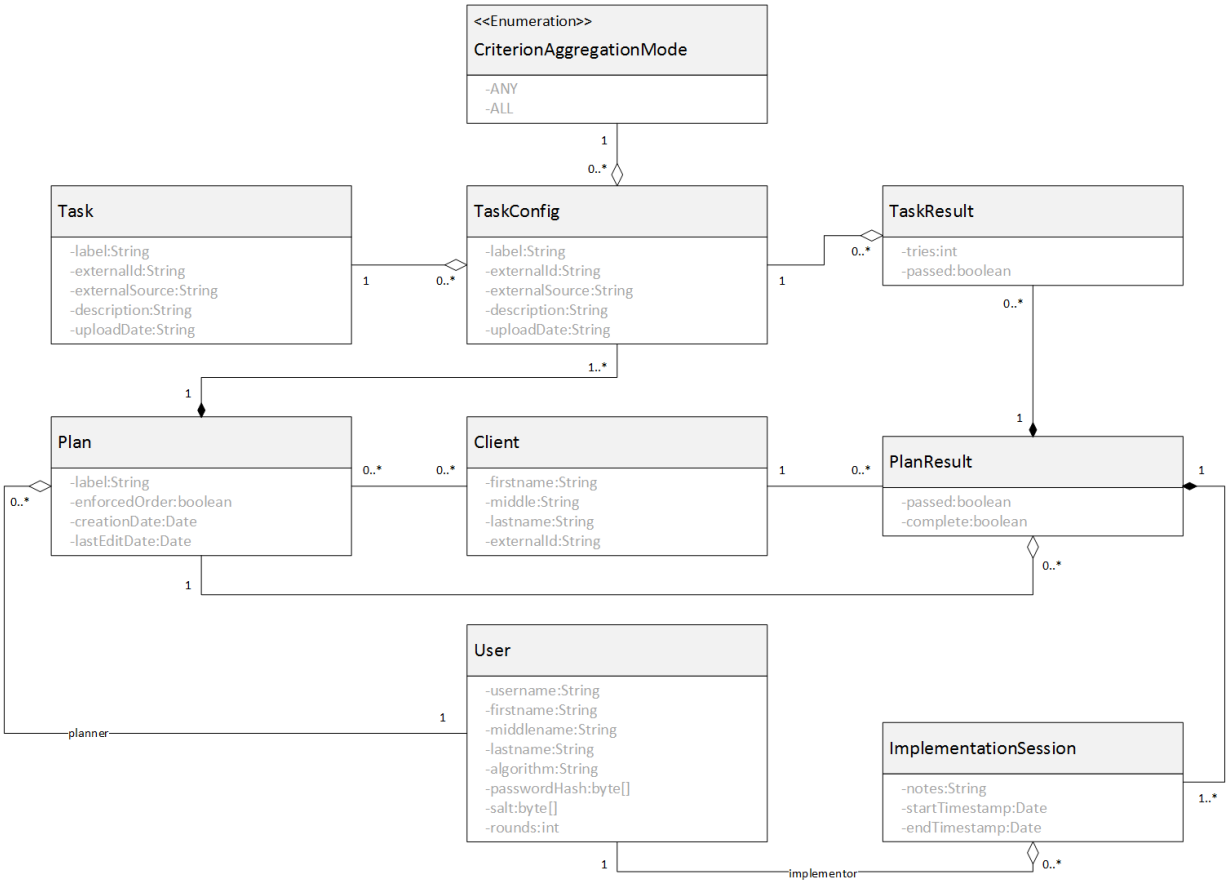


Figure 4.3: This UML class diagram shows the top level domain structure.

A `Task` can be a simulator scenario, a physical task (e.g. sit-ups), or basically anything else that is countable or results in numeric data. The domain structure mirrors this wide variety of possibilities in a flexible way (see figure 4.4, page 37). For this reason a `Task` has a list of `Parameters` which are task specific options for an external system (e.g. the CDS-250). Each `Task` can have multiple

SubTasks which allows a finer structure of certain steps. SubTasks consist of Properties and Measures. A Property defines a SubTask by providing additional numeric information (e.g. “step: 2” or ”speed limit: 40 mph”). A Measure represents a type of numeric data, this SubTask produces at implementation time. The following shows a simple example of a simulator task.

Task: SpeedControlStraight

- Parameter: Difficulty (Easy, Medium, Hard)
- SubTasks:
  - SpeedKeepingMeasures
    - Property: TargetSpeed: *35* mph
    - Measures: MaximumSpeed in mph, MeanSpeed in mph
  - SpeedKeepingMeasures
    - Property: TargetSpeed: *45* mph
    - Measures: MaximumSpeed in mph, MeanSpeed in mph
  - LaneKeepingMeasures
    - Measures: LeftEdgeTouches, RightEdgeTouches

Clients are usually managed by dedicated software products. In a hospital a common solution is a hospital information system or, in the case of a study, a list of participants has to exist. For that reason a Client is basically an external identifier tho match with those systems. Names can be used but are not mandatory.

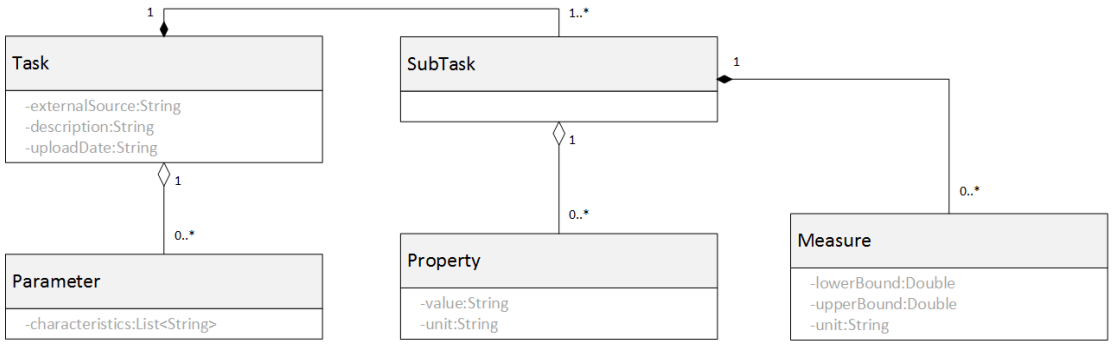


Figure 4.4: This UML class diagram shows the Task class substructure.

A Plan is essentially a list of TaskConfigs. It involves several options (e.g. task order enforcing at implementation time). A TaskConfig is linked to a specific Task and defines it by task-level options, ParameterConfigs, and SubTaskConfigs. ParameterConfigs define Parameters by specifying one of the

available characteristics. SubTaskConfigs contain information about Criterion setups. If a Measure is set up to be a Criterion it becomes a termination condition of the implementation of the Task and has to be specified with the exact condition (e.g. less than five errors). Criterion specification is based on structured data (e.g. comparison operator and numeric data fields for comparison values). In addition a Plan contains a list of Clients. This mirrors patient assignment to a therapy plan or participant assignment to a study.

The PlanResult class is the data collection part. It contains TaskResults which contain SubTaskResult, which contain MeasureResults. The latter represents actual data collected during the implementation. ImplementationSession data is stored to create a reliable history of sessions which happened during an implementation routine.

4.2.3.2 Data Access Layer and Database

The data access layer is a single class which uses EclipseLink, the JPA reference implementation, for database access. The original approach to create one DAO per domain entity is outdated, because it produces a lot of classes which makes it harder to maintain and contains plenty of redundant code [25]. For this reason the Protocol Manager has one generic DAO which is called DaoFacade (see appendix C.2, page 67). In addition it provides typecast methods (e.g. getAllTasks()). Listing 4.1 (page 39) gives an example of a generic method of the DaoFacade's implementation.

```
public synchronized long persist(AbstractEntity entity) throws
    EntityAlreadyExistsException {
        checkNotNull(entity, "entity");

        try {
                entityManager.getTransaction().begin();
                entityManager.persist(entity);
                entityManager.getTransaction().commit();
        } catch (PersistenceException e) {
                LOG.error(e.getMessage(), e);
                throw new EntityAlreadyExistsException(String.format("Entity of
                    type '%s' with id '%d':\n\r" + e.getMessage(), entity.
                    getClass().getSimpleName(), entity.getId()));
        }

        refresh(entity);
        return entity.getId();
}
```

Listing 4.1: DaoFacadeImpl.java: persist(AbstractEntity)

The MySQL database scheme is generated by EclipseLink. From that follows that the database scheme has the same structure as the domain model described in section 4.2.3.1. The following property in JPA's `persistence.xml` file enables automatic database table generation.

```
<property name="eclipselink.ddl-generation" value="create-tables" />
```

4.2.3.3 Service Layer

The service layer consists of three service classes (see appendix C.3, page 68).

The `AuthenticationService` handles user authentication by checking credentials. This is mandatory for session validation which is the base for application security and audit logging. Authentication happens by hashing the given password and comparing it to the database entry for the according user, as shown in listing 4.2 (page 40).

```
public User authenticate(String username, byte[] password) {
        User user = daoFacade.findUser(username);
        if (user != null) {
                HashingConfig config = getServerConfig(user);

                try {
                        byte[] hashedPassword = hasher.hash(password, config);
                        if (Arrays.equals(user.getPasswordHash(), hashedPassword))
                             {
                                LOG.exit(user);
                                return user;
                        }
                } catch (NoSuchAlgorithmException e) {
                        throw new RuntimeException(String.format("bad algorithm '%
                            s' in database", config.getAlgorithm()), e);
                }
        }

        return null;
}
```

Listing 4.2: AuthenticationServiceImpl.java: authenticate(String, byte[])

The `DataService` handles data related functionality. Listing 4.3 shows in a simple example how database entries are used to provide certain information. This class' writing methods can also translate network transmission formats (e.g.XML strings contained in TDFs) into persistable entities (e.g. an instance of the `Task` class).

```
public boolean isClientInUse(long clientId) {
        return daoFacade.getPlanResultCountForClientId(clientId) > 0;
}
```

Listing 4.3: DataServiceImpl.java: isClientInUse(long)

Audit logging happens in the service layer on top of any data access (see figure 4.5, page 41). Both, the `AuditLoggingDataService` and the real `DataService`, share the same Java interface.

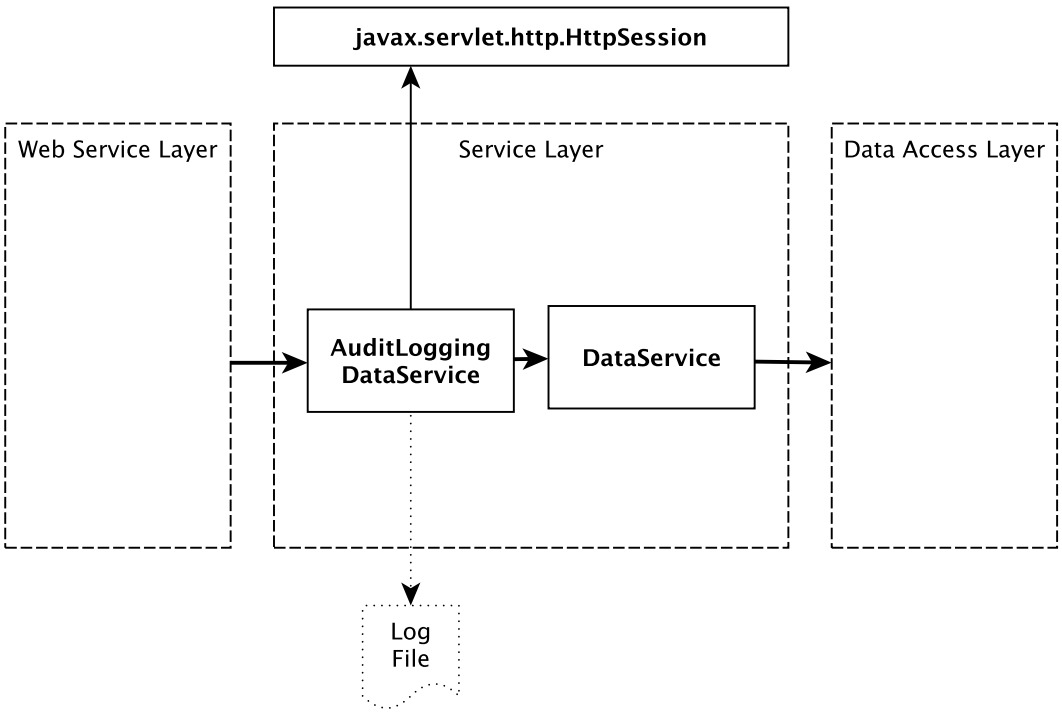


Figure 4.5: Audit logging happens when the data service is requested.

The logging service writes to an audit log file on the filesystem and passes the request to the actual service. Important information for an entry are timestamp, the executing user, and internal IDs of the addressed database rows. User information is provided by the `javax.servlet.http.HttpSession` class. Other sensitive information (e.g. patient data) have to be explicitly excluded from those log files. The following is an exemplary extract of an audit log file in a testing environment, during a sequence of task uploads.

```
[...]
Jan 20 2014 09:18:05 PM [hendrik@127.0.0.1] CREATED entity Task#307.
Jan 20 2014 09:18:09 PM [hendrik@127.0.0.1] CREATED entity Task#312.
Jan 20 2014 09:18:14 PM [hendrik@127.0.0.1] CREATED entity Task#316.
[...]
```

Assembling plan implementation results is the purpose of the `DataExportService`. It is basically a Comma-separated Values (CSV) parser which provides a file that is easy to process for further analysis.

4.2.3.4 Web Service Layer

In a Java context web services are called *servlets*, which platform specific term (see "What is a Servlet?" by Mordani [26, p. 1]). Figure 4.6 shows the adjusted servlet layer to allow communication with both: GWT's Remote Procedure Call (RPC) and REST speaking clients. According to the original concept any network request is filtered by an authentication filter which is independent from the underlying servlet implementation.

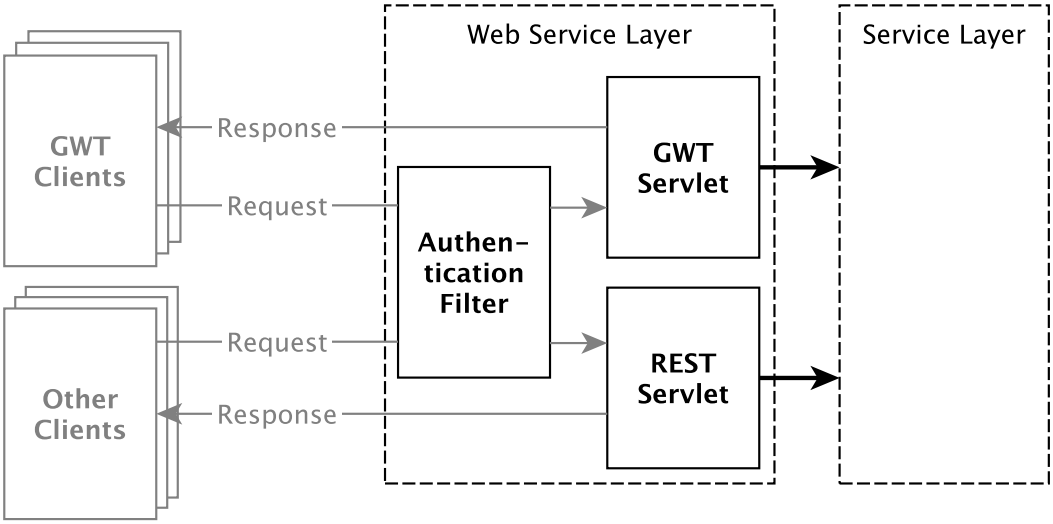


Figure 4.6: The adjusted servlet layer of the Protocol Manager's server implementation is capable of serving two different network technologies: GWT's RPC and REST.

The REST interface is used for communication with the simulator CS. This decision is based on REST being a wide-spread network communication paradigm, which is already being supported by the CS. JAX-RS provides the base for the implementation. Although GWT communication could be implemented using a different technology based on the HTTP, GWT's RPC framework is easier to maintain. It communicates in a asynchronous way and provides Java objects on both ends. The programmer does not have to convert the data into a transfer format.

GWT Remote Procedure Call (RPC)

RPCs are GWT's default way of communication. Figure 4.7 illustrates the Java class and interface hierarchy on client as well as server side. A synchronous and a mirrored asynchronous version of each service's interface have to exist in the shared code module. While the server runs the synchronous implementation, the client only uses the asynchronous interface. The GWT compiler creates an asynchronous proxy server which manages communication.

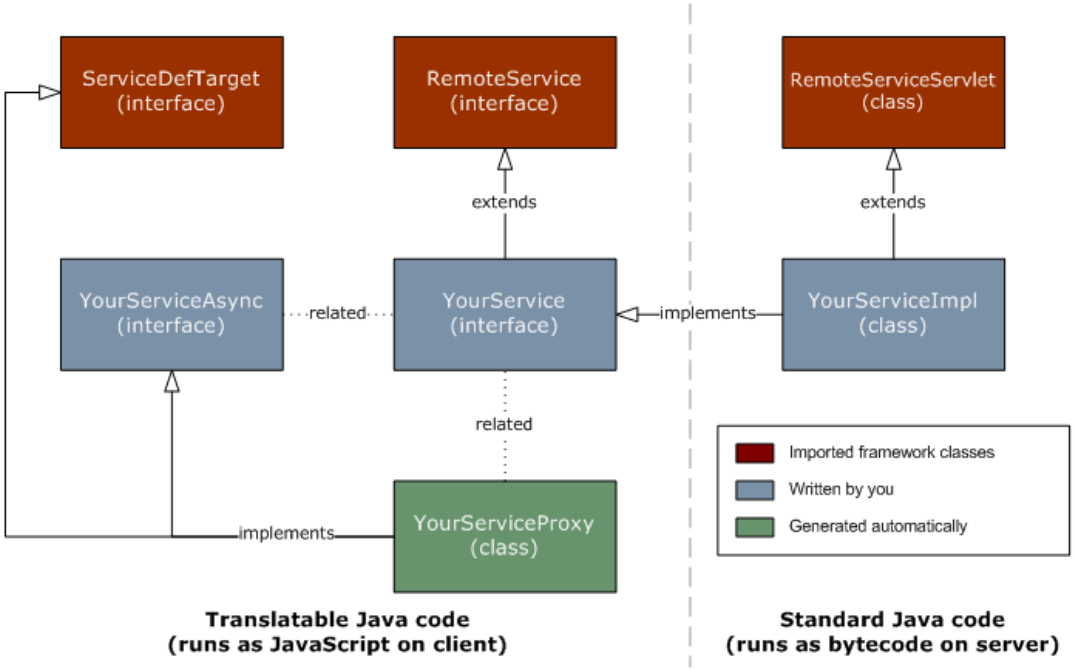


Figure 4.7: This class diagram illustrates the GWT RPC framework. [27]

REST

The REST interface provides access to functionality based on URL requests. To retrieve data from the server or send it to the server, the common HTTP methods GET and POST are supported. This makes communication for non-GWT clients easy. More complex data (e.g. a task) must be transmitted as XML or JSON media content. The following are examples of how to use the REST interface.

To get a task's content an HTTP GET request with the according ID as a parameter, is made by the client:

`http://<SERVER-ADDRESS>/task?id=123`

An HTTP POST of the task media file to the following URL transmits the data to the server:

`http://<SERVER-ADDRESS>/task/create`

4.2.4 Security

Section 3.2.5 (page 26) makes aware of common potential security threats. Some of them were damped or eliminated by architectural design others can only be obviated by implementation measures.

4.2.4.1 User Identity Management

An important responsibility of a server application developer is protecting user identities. If an identity is stolen the attacker can get access to sensitive data and all other protection measures become useless. In addition to encouraging users to choose strong passwords, there is a number of measures the developer can do when implementing authentication structures.

The Protocol Manager does not use any external identity management systems but stores user data in the server's own database. Since passwords are not stored but their hash values, the following password related columns are included in the user table: password hash, hashing algorithm, salt, and number of rounds. Per default the application uses a SHA-256 [28] hash, salted with a 32 byte long random sequence, which traverses hashing for 200 rounds. In addition the client has to request hashing instructions prior authentication which contains information to pre-hash the password by splitting the number of rounds between client and server. This way no plain text password has to be sent over the network. Because users tend to use the same password for different systems, this mechanism protects the user's plain password in the worst cases of a man-in-the-middle attack or server compromising.

4.2.4.2 Structured Query Language (SQL) Injection

Injection can happen when user input is passed to any kind of interpreter. On the server side this concerns the database SQL processor. A reliable method to prevent SQL injection is a combination of a query white list and parameter

escaping. JPA defines a mechanism to do exactly that: named queries, which is a variation of the prepared statements approach (see section 3.2.5.1, page 27). Listing 4.4 shows an example of a named query. Using it for finding an user by username makes it impossible to use the login form's username field for SQL injection. The server application uses exclusively named queries for database access.

```
@NamedQuery(name = User.FIND_BY_USERNAME, query = "SELECT u FROM User u WHERE u
    .username=?1")
public class User extends AbstractEntity { ... }
```

Listing 4.4: JPA Named Query: User.FIND_BY_USERNAME

4.2.4.3 XSS

The prevention of XSS means checking any non-constant input which is either passed to the user's web browser or the JavaScript engine. The GWT Project considers the following vectors as the only XSS vulnerabilities related to the framework [29]:

- JavaScript on the host page that is unrelated to GWT
- Code that sets `innerHTML` on GWT Widget objects
- Using the JSON API to parse untrusted strings
- Unsafe JavaScript Native Interface (JSNI) code

Since there is no JavaScript code unrelated to GWT and the JSON API is not used, only `innerHTML` method calls and JSNI code are left as potential risks. The only native method in the whole project is shown in listing 4.5. It redirects the browser to an address with `mailto:` prefix which usually triggers the client computer's default e-mail software. The address and the subject are always read from constant strings which leaves no way of injection for this method.

```
private native void mailto(String address, String subject) /*-{
       $wnd.location = "mailto:" + address + "?subject=" + subject;
}-*/;
```

Listing 4.5: JSNI: mailto(String, subject)

The last concern is calling the `innerHTML` method. Whenever it is necessary to set HTML to a GWT Widget (and thus to the resulting HTML page) GWT's `SafeHtml` type is used. It ensures XSS safety as long as the implementations

from the core GWT library are used. For that reason this project always uses GWT's `SafeHtmlBuilder` class to create instances of `SafeHtml`.

4.2.4.4 Missing Function Level Access Control

Section 3.2.5.3 (page 28) introduces the authentication filter as a conceptual solution to a potential circumvention of client side authentication. The Protocol Manager's `AuthenticationFilter` (see the code listing on page 70) intercepts all requests and checks for session validity. A session is valid if a user was successfully authenticated and it is not yet timed out.

4.2.4.5 XSRF

The RPC communication of the server application is protected against XSRF by using GWT's `XsrfTokenServiceServlet` class [30]. For any requests it calculates the MD5 hash of the session ID and expects the same hash code in form of an `RpcToken` in the client request's content. If they differ, it is either a programming error or an attempt of XSRF, and access is denied.

The Jersey solution for the same problem is the class `CsrfProtectionFilter` [31]. It implements the NSA's [32] and Stanford University's [33] guidelines for protection against XSRF. "The main idea is to check the presence of a custom header (agreed-upon between the server and a client – e.g. X-CSRF or X-Requested-By) in all state-changing requests coming from the client."

Since XSRF protection is based on a contract between client and server application, any additional clients using the web service interface have to adapt to the described methods of XSRF protection.

4.3 User Guidance

User guidance is a very important part of software development in general and of the focuses of this project. Usability and acceptance amongst the software's users have a strong correlation to the guidance through UIs. This sections shows how certain operations are performed in the Protocol Manager's two clients. It uses screenshots from the most recent version as examples. All screenshots were made in a testing installation of the software that mirrors the live system in functionality but uses no actual patient data. The involved tasks are real representations of CDS-250 scenarios.

For the use in CU-ICAR some terms deviate from terms used in this document. The following is a mapping of known terms and their synonyms used in this section's examples.

- plan = *protocol*
- property = *option*
- planning client = *protocol setup*
- implementation client = *data collection*

4.3.1 Task Upload

The first important step on the planning side is creating a list of available tasks. It is the base for any study or therapy protocol. To ensure that the tasks are synchronized with specifications on other systems, a manual creation or editing is not possible. A task definition file has to exist which ideally is exported from the according external device.

The protocol setup client provides an overview over available tasks (see figure 4.8, page 48). On the bottom of this page is a small form for uploading a new task. A TDF in XML or JSON can be selected and after successful parsing on the server-side it will be available in the overview list. From this view selection of a single task is possible, which gives more detailed information and the option to delete the task.

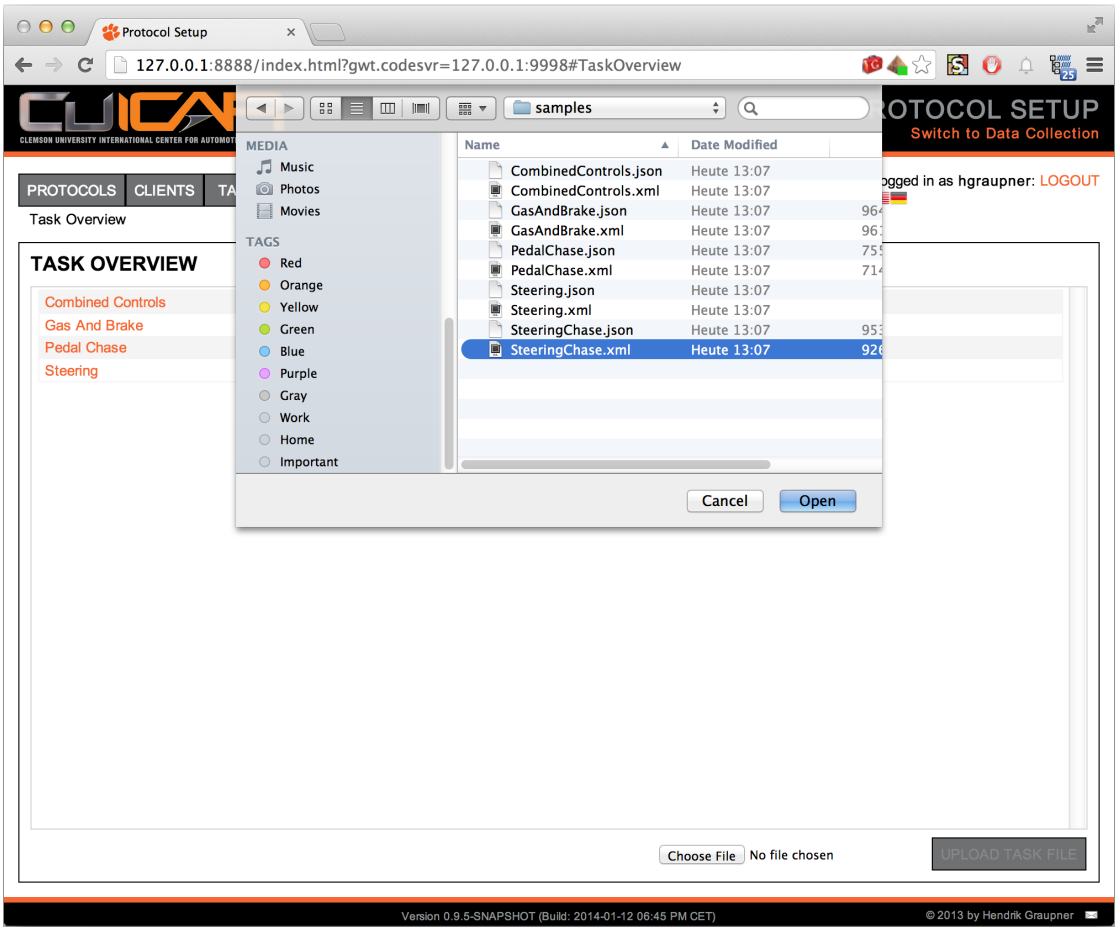


Figure 4.8: The task upload is initiated from the overview.

4.3.2 Client Creation

Clients can be introduced to the system using a short form (see figure 4.9, page 49). The only required field is *external ID* which links it to another system or list. After successful creation a new client will appear in the overview list. Client fields can be edited at a later time but the external ID is not changeable to prevent loss of identity. An entry can be deleted unless it is assigned to a protocol or is referenced by implementation results.

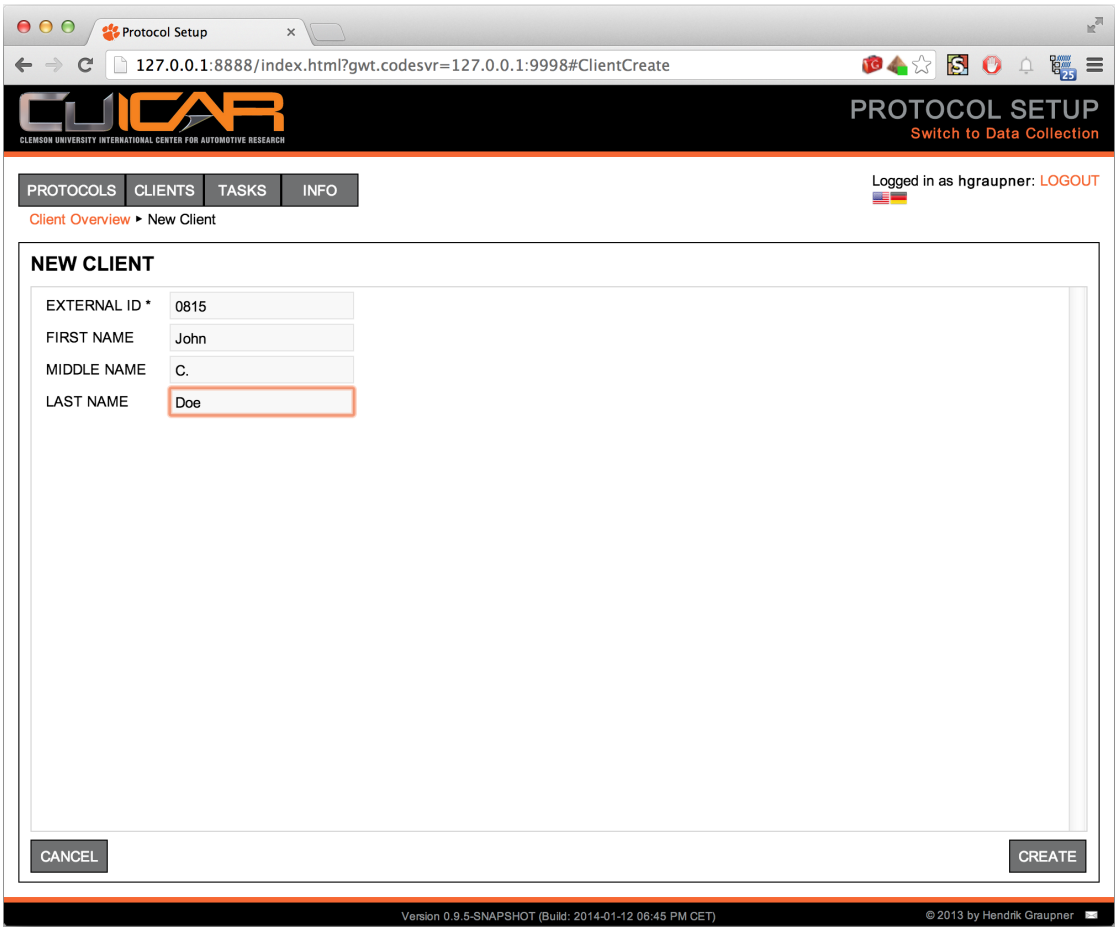


Figure 4.9: This form acquires information for client creation.

4.3.3 Protocol Creation

Once the system knows tasks a protocol can be created. Figure 4.10 (page 50) shows the form which is used for assemblage. It consists of a field for labeling, the option to fix the task order, and selection boxes for adding tasks and clients. Up and down arrows allow to change the task order at any time during the process.

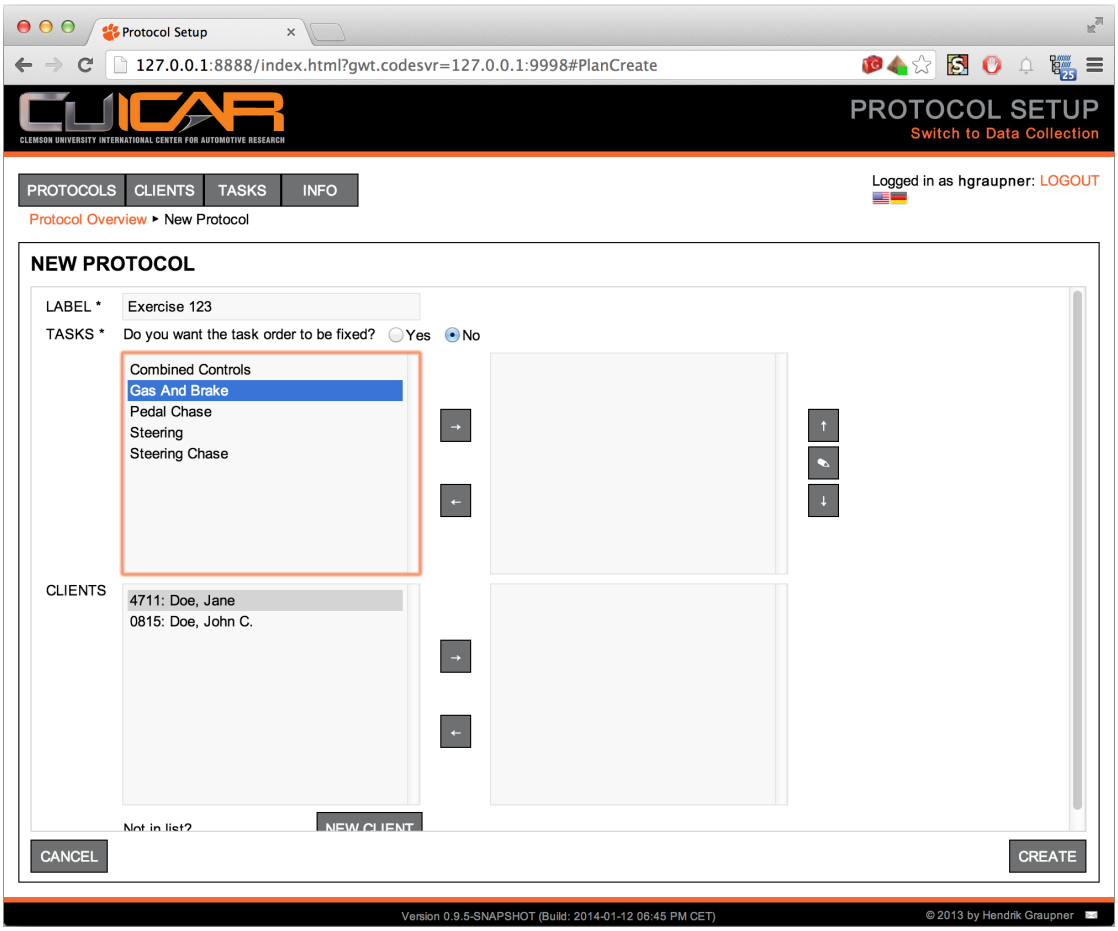


Figure 4.10: The assemblage of a protocol uses this form.

When a task is added a window (see figure 4.11, page 51) pops up to specify it for this particular setup. It can be labeled, options have to be defined, and criteria can be set. A task can also be added multiple times with different, or the same, configurations.

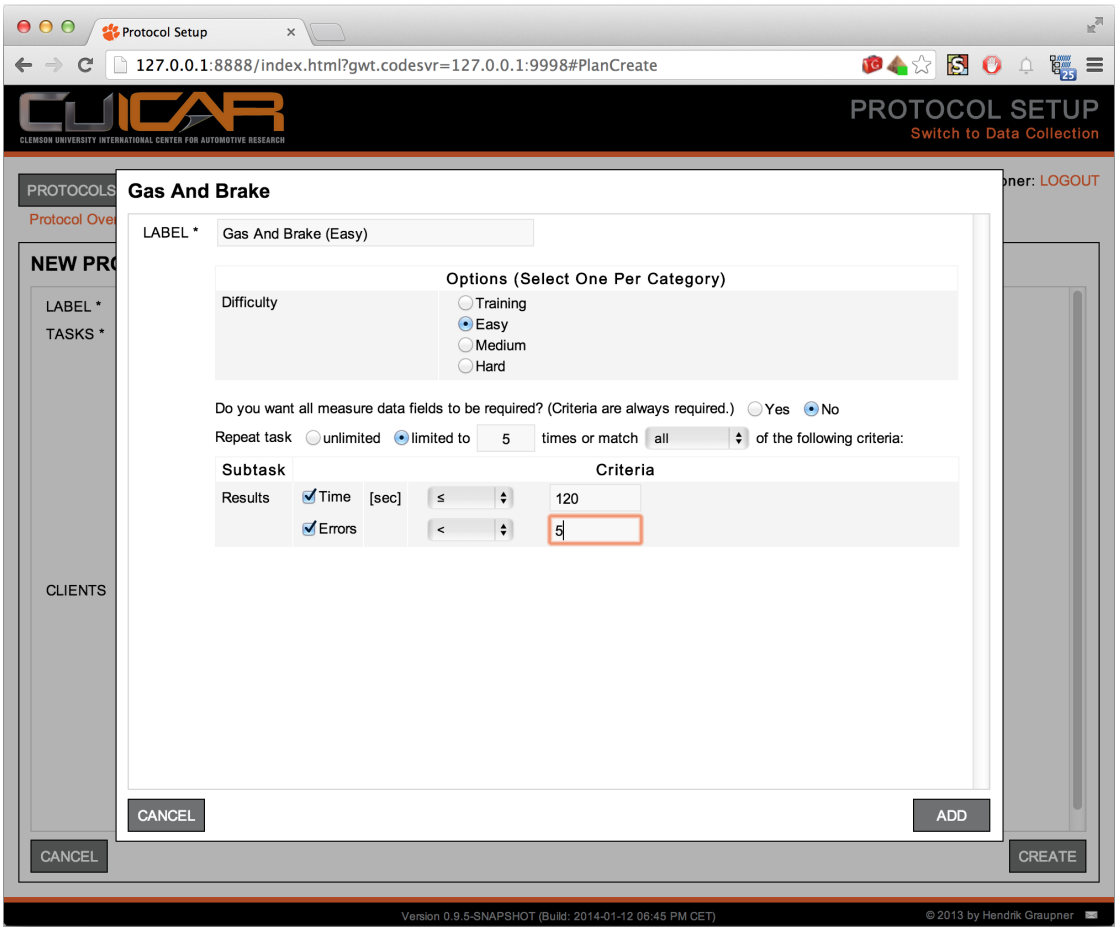


Figure 4.11: The task configuration window pops up when the user wants to add a task to the protocol.

Figure 4.12 (page 52) shows a protocol's detailed view after creation. In addition to deletion and editing it can also be duplicated. The latter means all configurations will be copied into a fresh protocol which has to be labeled differently.

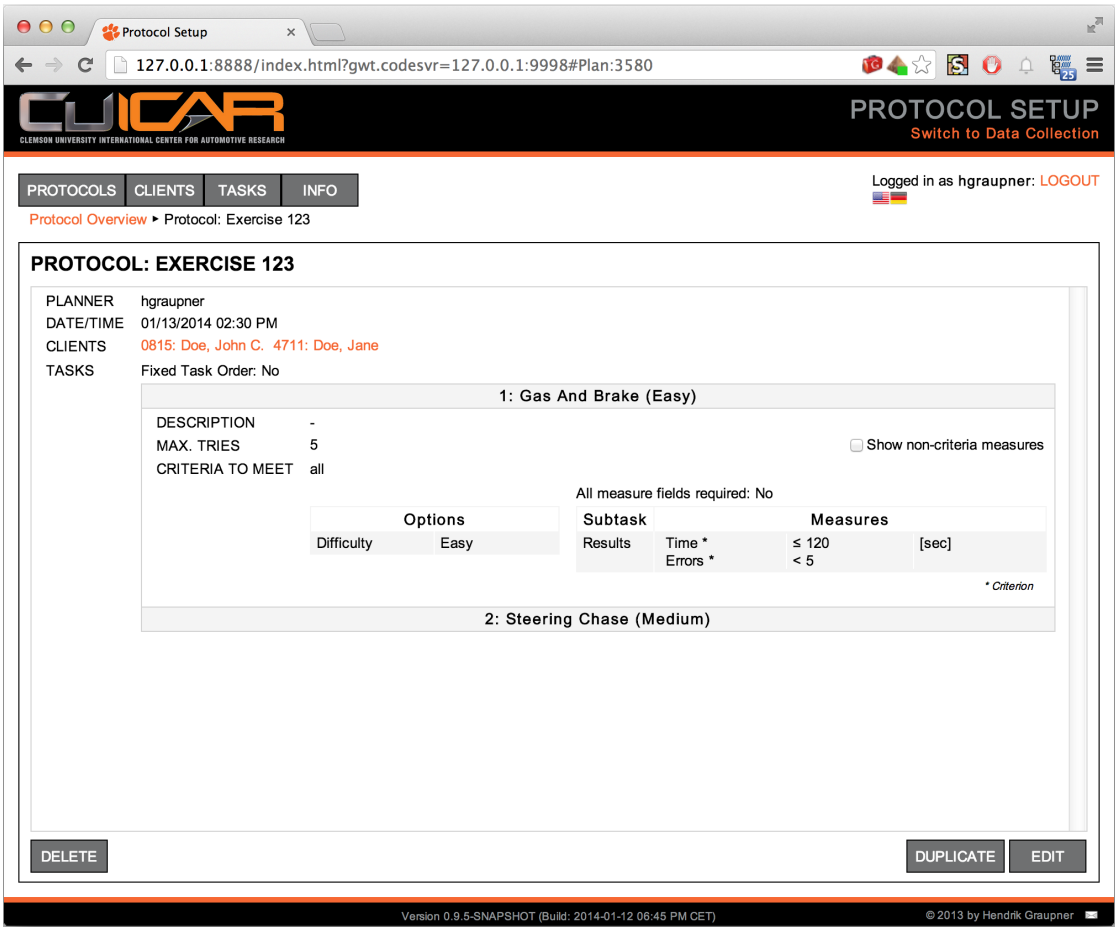


Figure 4.12: This view shows detailed information of a protocol.

Once a protocol exists it can be implemented. To do so in the user has to switch to the data collection client. In other fields of application this client may be included in another system like a dedicated CS.

4.3.4 Protocol Implementation

To start the data collection the user has to select a protocol and a client. Either can be selected first since both detailed views include a *Collect Data* button in the bottom right. The implementation view (see figure 4.13, page 53) shows a list of tasks defined for this protocol and a field for session notes. When the *fixed task order* option is activated, only the upcoming task can be selected. Otherwise, like in this example, the implementor can choose any to be next.

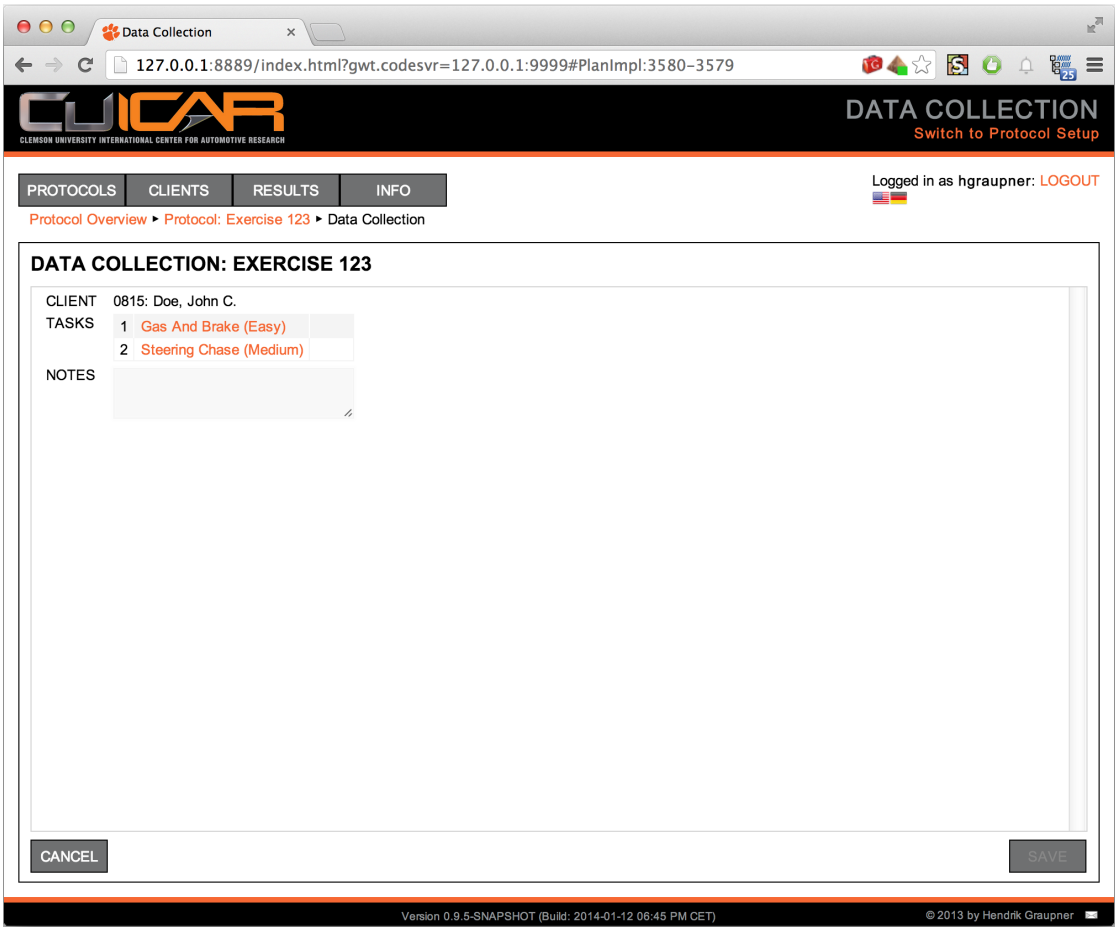


Figure 4.13: Protocol implementation starts from this view. Tasks can be selected from the list to enter measurements.

When a task is selected a data collection window pops up (see figure 4.14, page 54). It shows the task configuration's options (left) and a trial-wise measure input table (right). After each trial the user enters the according measures and presses the *Check Trial* button. Optional notes can be attached to each trial separately if something comes to the implementors attention.

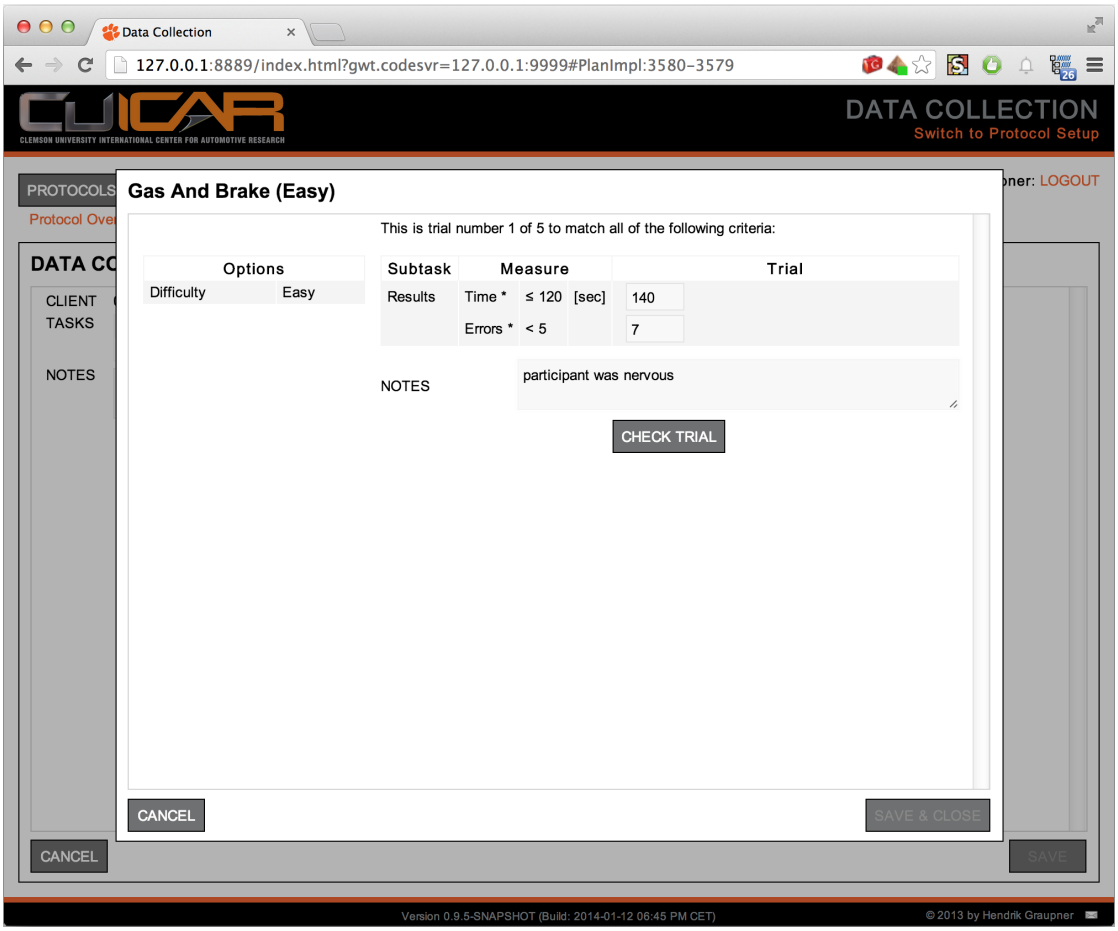


Figure 4.14: When a task was selected from the initial view this windows pops up to collect the session's measurements.

Once a task is successfully completed (see figure 4.15, page 55) the software returns to the initial implementation view. Success means that all predefined criteria are met. It is also possible to save the interim results and resume the data collection at a later time.

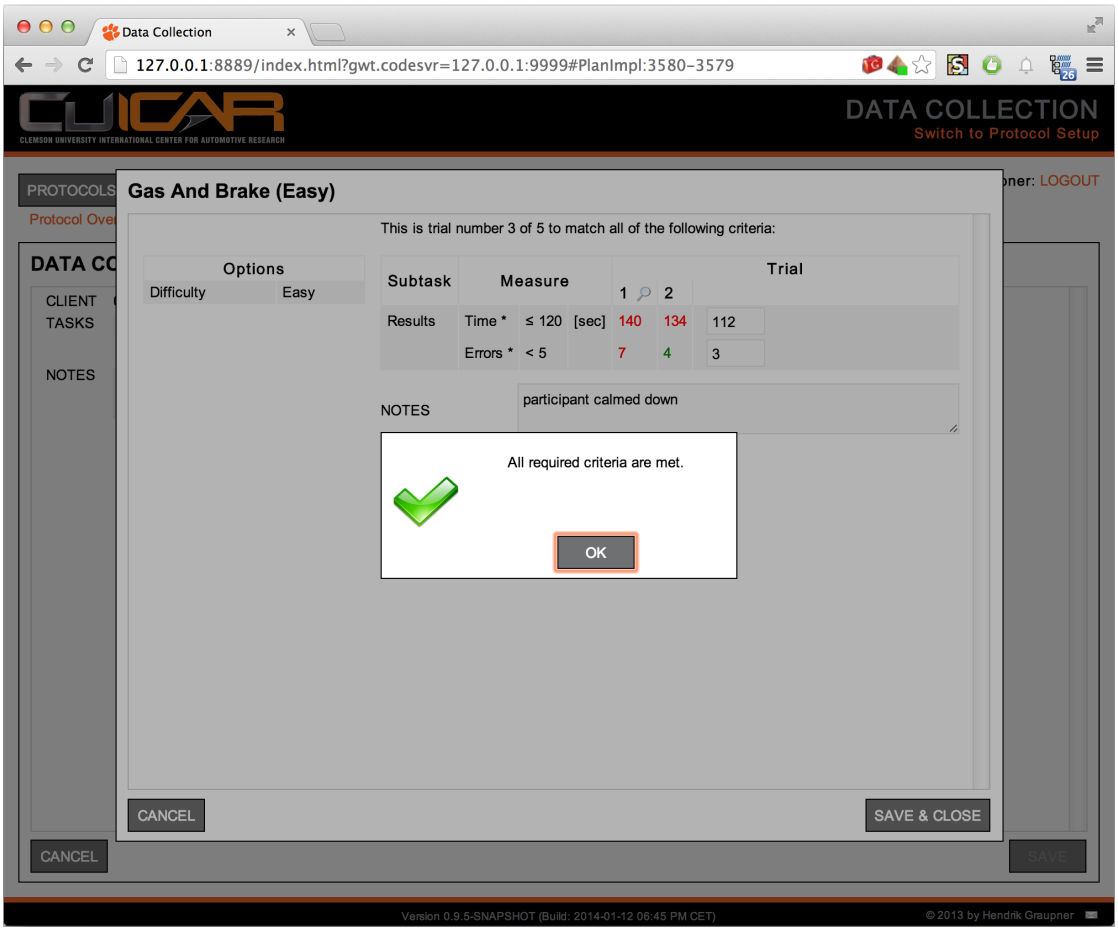


Figure 4.15: A green check-mark indicates the successful criterion check. Afterwards the user will be returned to the initial implementation view.

The initial view shows the state of a task as soon as the data collection started (see figure 4.16, page 56). A green check mark signals success. If the data collection for a task was not finished, the implementor can continue at a later time.

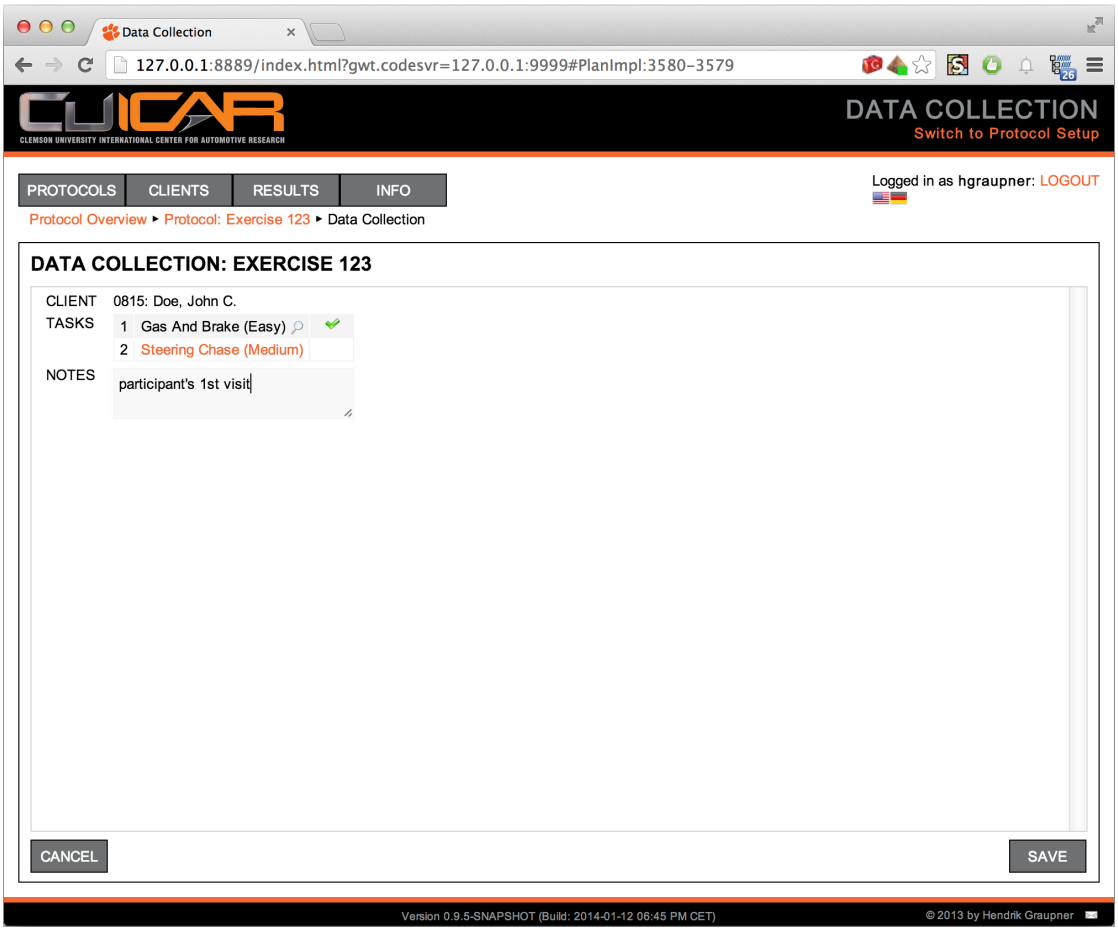


Figure 4.16: The implementation view shows a green check-mark next to the task when it is successfully completed.

At any time the user can decide to save data and leave the implementation by pressing the *Save* button. Any notes entered will be stored, together with a timestamp and a reference to the user logged in, for this session and can not be altered anymore. Afterwards the software redirects the user to the implementation result page.

4.3.5 Result Review and Export

Data that is already collected can be reviewed in the data collection client. Figure 4.17 (page 57) shows the implementation result view. The button *Resume Data Collection* allows resuming as long as not all tasks are completed. In this case it is greyed out like in the example. If resuming is available the user will be

returned to the initial implementation view. Successfully completed tasks are marked green while unsuccessful ones are marked red.

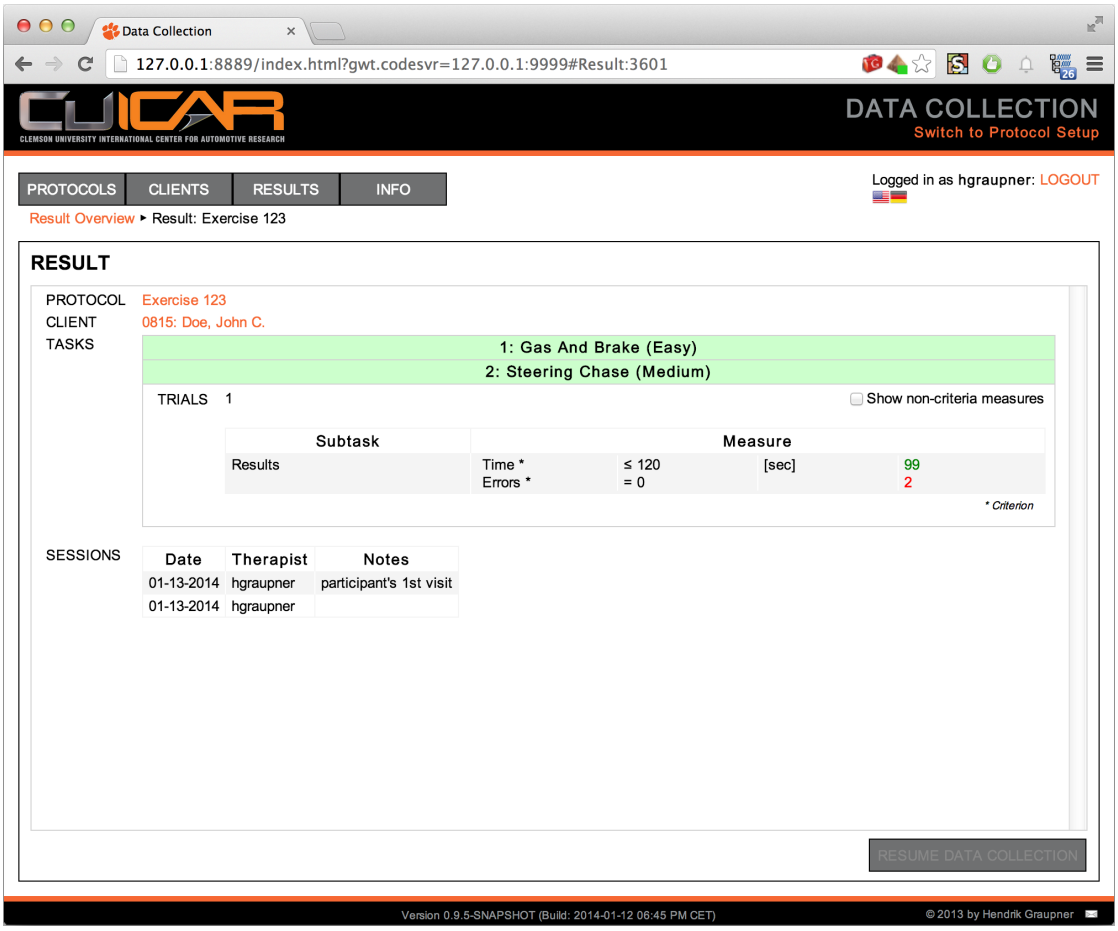
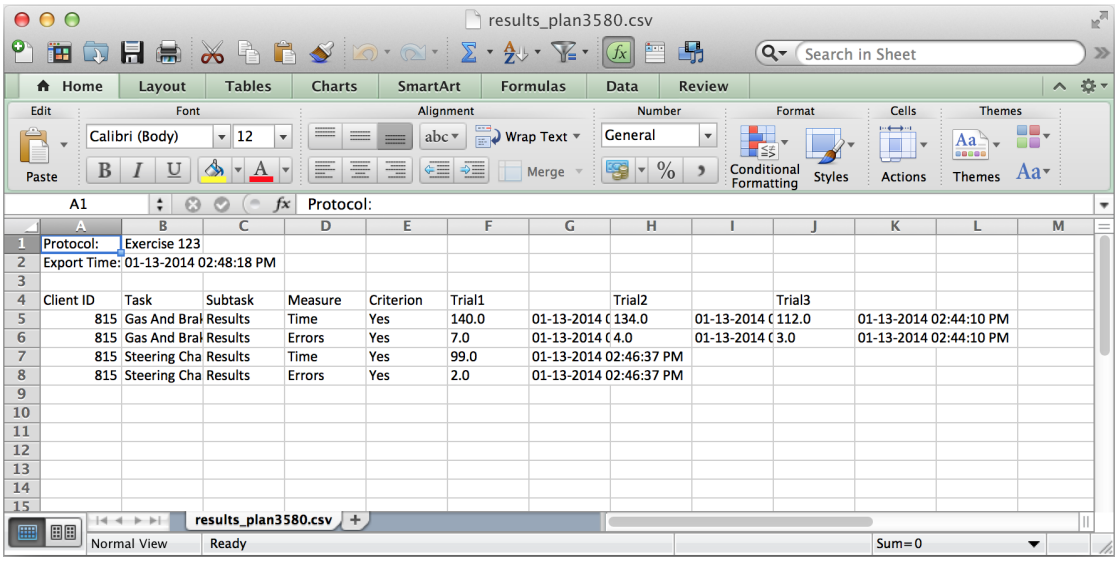
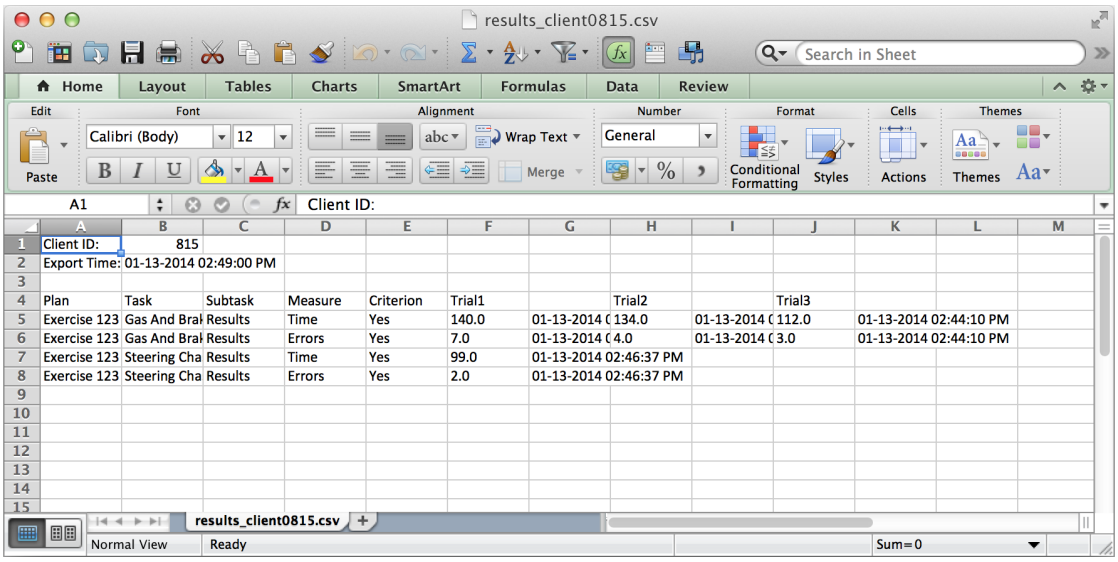


Figure 4.17: The implementation results are presented in a design similar to the protocol view. A green header indicates a successfully completed task.

Collected data can not only be reviewed in the software it can also be exported to a CSV file. Export is available protocol-wise (see figure 4.18a, page 58) and client-wise (see figure 4.18b, page 58). The export buttons can be found in the detailed view of a protocol or client. Both exports contain the collected measures sorted by tasks. Each measure has a timestamp of collection.



(a) The protocol-wise result export lists clients, who implemented the protocol, and their according measurements.



(b) The client-wise result export lists all protocols the person attended in sessions.

Figure 4.18: The two screenshots show result CSV files opened in Microsoft Excel. The upper file is the result of export for a specific plan and the lower file the result for a specific client.

5

Conclusion

This chapter summarizes the theoretical and practical work and passes the experience, gained during the process, on to potential follow-up projects. The validation presents requirements and results. The outlook shows opportunities discovered and ideas developed during the whole process.

5.1 Validation

This thesis' goal is to provide a conceptual solution for a therapy and study planning and implementation software system in a driving simulator environment. The reference implementation's goal is to apply the demands and guidelines, the conceptional part developed, to a real life environment. In the following the results, in terms of their accomplishment of the proposed goals, are validated.

5.1.1 Communication with other Systems

Section 3.1.2.1 (page 13) describes the technical infrastructure of a driving simulator environment. The concept is required to provide an interface which makes interaction with the two major software products, the CS and SS, possible. During the concept design it was verified that direct data transmission from the SS is not necessary. External communication is limited to the CS, which is supported by concept.

The reference implementation does not accomplish the goal of communication with the CS. It represents an intermediate version of the concept. Under this

thesis it was not possible to develop the CS-side interface for communication. For this reason the implementation is based on exclusively manual data input during the data collection process. However, first tests showed that simple REST based requests (e.g. authentication) from the simulator platform to the server's web service do work.

5.1.2 Functions

The two important user functions, plan creation and implementation, are defined in section 3.1.2.2 (page 14). For each of them a dedicated client application is specified. The planning client is dedicated to plan creation and the according preconditions tasks and clients. Its counterpart, the implementation client, provides data collection. All required user functions, including their sub functions, are supported by the conceptional solution.

In the reference implementation two separate standalone client applications exists, as defined by the architecture. They have been tested by the researchers under the case study. Each of them provides the specified functionality to the full extend.

5.1.3 Target Users

The target users defined in section 3.1.2.3 (page 16) are clinicians and data analysts. Design of the concept and development of the reference implementation happened in close cooperation with the human factors researchers at CU-ICAR. This team consist of psychologists and engineers, and can draw on clinical as well as research experience. The author believes that they are a representative target users although no active clinicians were involved. A potential follow-up project to refine the software should extend the testing team to clinical staff.

The incorporation of target users in the development process is a significant advantage of this project and the software's quality.

5.1.4 Legal Constraints

Based on the relevance of the HIPAA for this project (see section 3.1.2.4, page 16) technical safeguards are part of the concept as well as implementation. Automatic session termination and audit logging are part of the concept. In addi-

tion the productive installation at CU-ICAR involves transmission integrity and encryption via TLS [34], unique usernames, and database access control. Implications of the HIPAA which are not part of the concept or implementation are: automatic data encryption and decryption (besides transmission encryption), and database encryption and versioned backups.

The reference implementation is not certified in terms of its application security. Under such a process it has to be clarified to what extend the missing technical safeguards need to be implemented, considering other protecting factors (e.g. physical protection of the host computers). A definite statement to the security is, for the explained reasons, not possible at the time of publishing.

5.1.5 Expected Benefits

The most important benefit that is expected is the prevention of errors in complex data collection scenarios. Typing errors naturally occur when data is entered by hand into a system. In the concept this issue is solved by completely eliminating manual data input. It presumes that the data producing device is capable of sending occurring measurements directly to the server application. The server application provides a web interface to fetch and store data and provide it again in a processable format to the analyzing user. However, the reference implementation does rely on manual data input. A reduction of errors should still be evident by using the mechanism of data field validation (see section 3.2.3.2, page 25).

A secondary expected benefit is the increase of time efficiency. The basics of plan management, especially reutilization, intend to support the planning therapist. A list of available tasks and various plan settings are expected to prevent outdated or misleading information. The effect expected from those mechanisms is less time required for the planning process. For data collecting users the concept of client integration with a CS might be the most interesting feature. It allows the user to focus primarily on the client by reducing the number of different tools from three to one. The time-consuming process of filling out Microsoft Excel sheets is no longer necessary.

At the time of publishing the effects of error and time reduction have not been conclusively validated by a pilot study.

5.2 Outlook

This thesis presents the current state by the time of publishing. The author intends to initiate further development under a follow-up project. This section describes ideas as well as concrete plans of upcoming events.

The upcoming milestone for the Protocol Manager is be a real life study using the software performed in CU-ICAR in early 2014. It will be a solid base for the validation of this interim setup and should reveal a trend towards a higher level of time efficiency.

Once the concept is fully validated, the way is paved for finding new fields of application and environments, for example institutions that use driving simulators or other therapy devices. Especially a hospital is a good environment to set up a second installation to gain clinical experience. This step requires a certification in terms of data security and HIPAA compliance because real patient data will be processed.

A future step will be the full integration into the existing software infrastructure by extending the CDS-250 CS to the implementation client's functionality. The generated feedback to the concept may reveal performance leaks, the possibility of new features to increase utility etc. The following tries to inspire additional features to extend the software by additional functionality.

- Enhanced plan management (e.g. extended template function)
- Connectors to other external systems (e.g. for user management, more automatic data transmission etc.) as well devices (e.g. other simulator types)
- More flexible criterion definition (e.g. by using a formula editor)
- Enhanced result export (e.g. by introducing an interpreter of regular expressions)

Appendices

A HIPAA Extracts

§164.312(a)(1) *Access control.* Implement technical policies and procedures for electronic information systems that maintain electronic protected health information to allow access only to those persons or software programs that have been granted access rights as specified in §164.308(a)(4).

§164.312(b) *Audit controls.* Implement hardware, software, and/or procedural mechanisms that record and examine activity in information systems that contain or use electronic protected health information.

§164.312(c)(1) *Integrity.* Implement policies and procedures to protect electronic protected health information from improper alteration or destruction.

§164.312(e)(1) *Transmission security.* Implement technical security measures to guard against unauthorized access to electronic protected health information that is being transmitted over an electronic communications network.

[5]

B Format Definitions

B.1 Task.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE task [
  <!ATTLIST task id ID "0">
  <!ELEMENT task (label,description?,externalSource?,externalId?,uploadDate?,
      options?,subTasks)>
  <!ELEMENT label (#CDATA)>
  <!ELEMENT description (#CDATA)>
  <!ELEMENT externalSource (#CDATA)>
  <!ELEMENT externalId (#CDATA)>
  <!ELEMENT uploadDate (#CDATA)>

  <!-- Options/Parameters -->
  <!ELEMENT options (option+)>
  <!ELEMENT option (label,externalId?,values)>
  <!ATTLIST option id ID "0">
  <!ELEMENT values (value+)>
  <!ELEMENT value (#CDATA)>

  <!-- SubTasks -->
  <!ELEMENT subTasks (subTask+)>
  <!ELEMENT subTask (label,externalId?,properties?,measures?)>
  <!ATTLIST subTask id ID "0">
  <!-- SubTasks: Properties -->
  <!ELEMENT properties (property+)>
  <!ELEMENT property (externalId?,value,unit?)>
  <!ATTLIST property id ID "0">
  <!ELEMENT value (#CDATA)>
  <!ELEMENT unit (#CDATA)>
  <!-- SubTasks: Measures -->
  <!ELEMENT measures (measure+)>
  <!ELEMENT measure (externalId?,lowerBound?,upperBound?,unit?)>
  <!ATTLIST measure id ID "0">
  <!ELEMENT lowerBound (#CDATA)>
  <!ELEMENT upperBound (#CDATA)>
]>
```

C Code Extracts

C.1 UiBinder Example

C.1.1 TaskOverviewViewDesktopImpl.ui.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ui:UiBinder SYSTEM "http://dl.google.com/gwt/DTD/xhtml.ent">
<ui:UiBinder xmlns:ui="urn:ui:com.google.gwt.uibinder" xmlns:g="urn:import:com.
    google.gwt.user.client.ui" xmlns:d="urn:import:edu.clemson.carsim.client.
    view.desktop.widget">
  <ui:with field="cc" type="edu.clemson.carsim.client.i18n.I18NConstants" />
  <ui:with field="st" type="edu.clemson.carsim.client.resources.StyleNames" />
  <g:SimplePanel styleName="{st.contentPanel}">
    <d:SmartHeaderPanel styleName="{st.innerContent}">
      <g:Label text="{cc.taskOverview}" styleName="{st.heading} {st.pageHeader}"
          />
      <g:SimplePanel styleName="{st.scrollPanel}">
        <g:VerticalPanel width="100%">
          <d:SmartFlexTable ui:field="taskTable" />
          <g:Label ui:field="emptyMessageOutput" visible="false" />
        </g:VerticalPanel>
      </g:SimplePanel>
      <g:FormPanel styleName="{st.pageFooter}" ui:field="uploadForm">
        <g:HTMLPanel>
          <table width="100%">
            <tr>
              <td align="right" valign="middle">
                <g:FileUpload ui:field="fileUpload" name="uploadFile" />
                <g:Button ui:field="submitButton" text="{cc.uploadTaskFile}"
                    enabled="false" />
              </td>
            </tr>
          </table>
        </g:HTMLPanel>
      </g:FormPanel>
    </d:SmartHeaderPanel>
  </g:SimplePanel>
</ui:UiBinder>
```

C.1.2 TaskOverviewViewDesktopImpl.java

```
public final class TaskOverviewViewDesktopImpl extends
    AbstractOverviewViewDesktopImpl implements TaskOverviewView {

        interface Binder extends UiBinder<Widget, TaskOverviewViewDesktopImpl>
            {}
        private static final Binder binder = GWT.create(Binder.class);

        private Presenter presenter;

        @UiField
        SmartFlexTable taskTable;

        @UiField
        FormPanel uploadForm;

        @UiField
        FileUpload fileUpload;

        @UiField
        Button submitButton;

        @UiField
        Label emptyMessageOutput;

        public TaskOverviewViewDesktopImpl() {
                initWidget(binder.createAndBindUi(this));
        }

        @Override
        public void setPresenter(Presenter presenter) {
                this.presenter = presenter;
        }

        [...]
}
```

C.2 DaoFacade.java

```
public interface DaoFacade {

        long persist(AbstractEntity entity) throws IllegalArgumentException,
            EntityAlreadyExistsException;

        <E extends AbstractEntity> E merge(E entity) throws
            IllegalArgumentException, EntityNotFoundException;

        void delete(AbstractEntity entity) throws IllegalArgumentException;

        <E extends AbstractEntity> E find(Class<E> type, long id) throws
            IllegalArgumentException;

        User findUser(String username) throws IllegalArgumentException;

        List<Task> getAllTasks();

        List<Plan> getAllPlans();

        List<PlanResult> getAllResults();

        List<Client> getAllClients();

        int getTaskConfigCountForTaskId(long taskId);

        int getPlanResultCountForPlanId(long planId);

        int getPlanResultCountForClientId(long clientId);
}
```

C.3 Service Layer Interfaces

C.3.1 AuthenticationService.java

```
public interface AuthenticationService {

        User authenticate(String username, byte[] password);

        User getUser(String username);

        HashingConfig getClientConfig(String username);
}
```

C.3.2 DataService.java

```
public interface DataService {

        long create(AbstractEntity entity) throws IllegalArgumentException,
                        EntityAlreadyExistsException;

        <E extends AbstractEntity> E update(E entity) throws
            IllegalArgumentException,
                        EntityNotFoundException;

        <E extends AbstractEntity> void delete(Class<E> type, long id) throws
            IllegalArgumentException,
                        EntityNotFoundException, DeleteForbiddenException;

        <E extends AbstractEntity> E find(Class<E> type, long id) throws
            IllegalArgumentException;

        User findUser(String username) throws IllegalArgumentException;

        List<Task> getAllTasks();

        List<Plan> getAllPlans();

        List<PlanResult> getAllPlanResults();

        List<Client> getAllClients();

        long createTaskFromXml(String xml) throws EntityAlreadyExistsException,
            BadFormatException;

        long createTaskFromJson(String json) throws EntityAlreadyExistsException
            , BadFormatException;

        boolean isTaskInUse(long taskId);

        boolean isPlanInUse(long planId);

        boolean isClientInUse(long clientId);
}
```

C.3.3 DataExportService.java

```
public interface DataExportService {

	String convertPlan(Plan plan);

	String convertClient(Client client);
}
```

C.4 AuthenticationFilter.java

```
@Singleton
public final class AuthenticationFilter implements Filter {

	@Override
	public void doFilter(ServletRequest req, ServletResponse resp,
	    FilterChain chain) throws IOException, ServletException {
		HttpServletRequest request = (HttpServletRequest) req;
		HttpServletResponse response = (HttpServletResponse) resp;

		if (SessionUtil.isValid(request.getSession()))
			chain.doFilter(req, resp);
		else
			response.sendError(HttpServletResponse.SC_UNAUTHORIZED);
	}
}
```

Bibliography

[1] M. Rapoport, B. Weaver, A. Kiss, C. Zucchero Sarracini, H. Moller, N. Herrmann, K. Lanctôt, B. Murray, and M. Bédard. "The Effects of Donepezil on Computer-Simulated Driving Ability Among Healthy Older Adults". In: *Journal of Clinical Psychopharmacology* 31.5 (Oct. 2011), pp. 587–592.

[2] United States Department of Health and Human Services (HHS). *HIPAA Administrative Simplification Statute and Rules*. URL: `http://www.hhs.gov/ocr/privacy/hipaa/administrative/securityrule/index.html` (visited on 10/15/2013).

[3] Institute of Electrical and Electronics Engineers (IEEE). *Recommended Practice for Software Requirements Specifications*. 1998. ISBN: 0-7381-0332-2.

[4] A. Cockburn. *Writing Effective Use Cases*. Agile Software Development Series. Addison-Wesley, 2001. ISBN: 978-0-2017-0225-5.

[5] National Institute of Standards and Technology (NIST). *Health Insurance Portability and Accountability Act, Administrative Simplification*. Feb. 2006.

[6] M. Potel. *MVP: Model-View-Presenter. The Taligent Programming Model for C++ and Java*. Tech. rep. Taligent, Inc., 1996.

[7] S. Burbeck. *Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)*. 1992. URL: `http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html` (visited on 01/28/2014).

[8] M. Fowler. *Supervising Controller*. URL: `http://www.martinfowler.com/eaaDev/SupervisingPresenter.html` (visited on 01/28/2014).

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[10] Microsoft. *SOA in the Real World*. Tech. rep. Microsoft, 2014. URL: http://msdn.microsoft.com/en-us/library/bb833022.aspx (visited on 01/28/2014).

[11] Sun Microsystems, Inc. *Core J2EE Patterns - Data Access Object*. Tech. rep. Sun Microsystems, Inc., 2002. URL: http://www.oracle.com/technetwork/java/dataaccessobject-138824.html (visited on 01/28/2014).

[12] R. T. Fielding. "Architectural Styles and the Design of Network-based Software Architectures". PhD thesis. University of California, 2000.

[13] N. Smithline and T. Gigler. *OWASP Top 10 Application Security Risks - 2013*. URL: https://www.owasp.org/index.php?title=Top_10_2013-Top_10&oldid=154333 (visited on 12/29/2013).

[14] M. Shema. *Hacking Web Apps*. Syngress, Aug. 2012. ISBN: 978-1-59749-951-4.

[15] X. Lin, P. Zavarsky, R. Ruhl, and D. Lindskog. "Threat Modeling for CSRF Attacks". In: *International Conference on Computational Science and Engineering* (2009), pp. 486–491.

[16] S. Pericas-Geertsen and M. Potociar. *JAX-RS: Java™ API for RESTful Web Services*. May 2013.

[17] S. Vajjhala and J. Fialli. *The Java™ Architecture for XML Binding (JAXB) 2.0*. Apr. 2006.

[18] L. DeMichiel. *JSR 338: Java™ Persistence API, Version 2.1*. Apr. 2013.

[19] L. Andersen. *JDBC™ 4.0 Specification*. Nov. 2006.

[20] Microsoft. *Data Transfer Object*. URL: http://msdn.microsoft.com/en-us/library/ms978717.aspx (visited on 01/29/2014).

[21] GWT-Project. *MVP Activities And Places*. URL: http://www.gwtproject.org/doc/latest/DevGuideMvpActivitiesAndPlaces.html (visited on 01/06/2014).

[22] C. Ramsdale. *GWT MVP*. URL: http://www.gwtproject.org/articles/mvp-architecture.html (visited on 12/01/2013).

[23] GWT-Project. *UiBinder*. URL: http://www.gwtproject.org/doc/latest/DevGuideUiBinder.html (visited on 01/07/2014).

[24] L. DeMichiel and B. Shannon. *Java™ Platform, Enterprise Edition (Java EE) Specification, v7*. Apr. 2013.

[25] P. Mettqvist. *Don't repeat the DAO!* URL: http://www.ibm.com/developerworks/library/j-genericdao/ (visited on 01/29/2014).

[26] R. Mordani. *Java™ Servlet Specification*. Dec. 2009.

[27] GWT-Project. *GWT Remote Procedure Calls*. URL: http://www.gwtproject.org/doc/latest/DevGuideServerCommunication.html (visited on 12/04/2013).

[28] National Institute of Standards and Technology (NIST). *FIPS PUB 180-4: Secure Hash Standard (SHS)*. Tech. rep. National Institute of Standards and Technology, Mar. 2012.

[29] D. Morrill. *GWT Security*. URL: http://www.gwtproject.org/articles/security_for_gwt_applications.html (visited on 12/05/2013).

[30] GWT-Project. *Security Rpc Xsrf*. URL: http://www.gwtproject.org/doc/latest/DevGuideSecurityRpcXsrf.html (visited on 01/12/2014).

[31] M. Matula. *Jersey and Cross-Site Request Forgery (CSRF)*. URL: http://blog.alutam.com/2011/09/14/jersey-and-cross-site-request-forgery-csrf/ (visited on 01/12/2014).

[32] National Security Agency (NSA). *Guidelines for Implementation of REST*. Tech. rep. National Security Agency, Mar. 2011.

[33] A. Barth, C. Jackson, and J. C. Mitchell. "Robust Defenses for Cross-Site Request Forgery". In: *15th ACM Conference on Computer and Communications Security* (Oct. 2008).

[34] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol: Version 1.2*. Aug. 2008.

Affidavit

I hereby declare that the master's thesis I have submitted is my own work. Where the work of others has been quoted or reproduced, the source is always given.

This thesis or parts have not yet been presented to an university as part of an examination or degree.

Heilbronn; January 30, 2014 _____

Hendrik Graupner