SLRP

HIBU STUDENT PROJECT 2011

CORENA S1000D 4.0 PROCESS DATA MODULE RENDERER

Arild Oldervoll

Marius Haraldseth

Eirik André Eidså

Olav Brandt

DOCUMENTS

ameo

DOCUMENT OVERVIEW

HIBU STUDENT PROJECT 2011
CORENA S1000D PROCESS DATA MODULE RENDERER

Arild Oldervoll

Marius Haraldseth

Eirik André Eidså

Olav Brandt

TABLE OF CONTENTS

REVISION HISTORY			
Version Number	**Date**	**Changes**	**Assigned**
1	05-Jan-2011	First version	AOL
2	07-Mar-2011	Second Official Version	MHA
3	01-Jun-2011	Third Official Version - Updated documents with new chapters and added documents prior to third presentation.	MHA

1 INTRODUCTION

Our assignment to develop a logic engine for support execution of S1000D process Data Modules version 4.0, is given by CORENA Norge A/S. This logic engine can take input from users and/or external applications, such as sensors, to decide its next step in a flow of process Data Modules. It can branch and loop through the selected process Data Module. The logic engine will be an interpreter for process Data Modules. We will make it a stand-alone logic engine in a web based application running on Jetty/Tomcat. The user of the application should be able to move back and forth through the steps using previous and next buttons. The user should be able to save the session to continue at a later time. The logic engine should support the International S1000D Specification Issue 4.0.

1.1 INTENTION OF THIS DOCUMENT

This document overview will provide a listing of the documents we have produced for this project, together with a short description of each of them.

1.2 SYNONYMS, ACRONYMS AND DEFINITIONS

Synonym	Description
Process	Set of instructions and conditions being executed
Data Module	Container of data/information.

1.3 AUTHOR(S)

Olav Brandt (OBR)

Marius Haraldseth (MHA)

1.4 ASSIGNED

Olav Brandt (OBR)

2 DOCUMENTS PRIOR TO THE FIRST PRESENTATION

2.1 VISION DOCUMENT

The *Vision Document* defines the basis of the project, and was the first piece of documentation produced. It provides an introduction to our assignment, the company behind it - CORENA, and some initial requirements. Additionally, the document answers the question of why we want to do this assignment, and our vision for the project.

Content (chapters):

1. Introduction
2. Participants
3. The Assignment
4. Expected Results
5. Why CORENA need this
6. Why we want to do this assignment
7. Our vision for this project

(File name: Vision Document.pdf)

2.2 PRESTUDY REPORT

The *Prestudy Report* builds on the vision document, and goes further into detail on the subjects described there. The report goes into depth regarding the assignment, the system it is a part of, some problems and limitations, as well as containing a list of high level project activities. It also looks at some of the project's surroundings; the environment it will be developed in, our project development model and required resources. Finally, it examines some related projects, alternative solutions to the project's requirements, and a discussion of the consequences should we choose to start or not start this assignment.

Content (chapters):

1. Introduction
2. Project Environment
3. Related Projects
4. The System

5. The Assignment
6. Alternative Solutions
7. Consequences for Each Participant of the Project, if We Start or Don't
8. Problems and Limitations
9. High Level Activity Overview
10. Project Development Model
11. Resources Required in the Project Split on Activities
12. Activities and Resources Until the Project Plan is Ready

(File name: Prestudy Report.pdf)

2.3 PRODUCT BACKLOG

The *Product Backlog* contains requirements in the form of *User Stories*. The User Stories are prioritised by the Product Owner, and each has a dimensionless and comparative time/effort estimate. Linked to the User Stories are *test cases*, one or more per User Story.

Content (sheets):

- Product Backlog
- Explanation
- Removed User Stories

(File name: Product Backlog.pdf)

2.4 TEST CASES

The *Test Cases* document contains all of our tests for the User Stories. The tests are linked to specific User Stories. The test cases describe the testing procedure, with the action(s) to be taken by the tester and the expected result.

Content:

- Test Cases
- Action(s)
- Expected Result(s)

(File name: Test Cases.pdf)

2.5 QUALITY ASSURANCE

The *Quality Assurance document* describes what and how we will be conducting the document reviews and software testing in our project. It starts off by introducing our document review procedures, as well as various important concepts, such as verification and validation, functional and non-functional tests, and test levels. It also examines specific techniques and approaches to the testing process. In the latter parts of the document, it goes into detail on each type of (functional) testing we will be performing; from unit testing to system and acceptance testing.

Content (chapters):

1. Introduction
2. Document Review
3. Software Testing

(File name: Quality Assurance.pdf)

2.6 PROJECT PLAN

The project plan is a layout for the rest of the project, as well as a further continuation of the documentation produced in the prestudy report. Subjects described here include, amongst others, the goals, limitations, preconditions and organisation of the project. It is then followed by a list of 3-week Sprints with start and end dates, an estimated amount work hours available and an initial goal for the time period.

Content (chapters):

1. Introduction
2. The Task
3. Project Goals and Limitations
4. Preconditions for This Project
5. Project Organisation
6. Roles and Responsibilities
7. Project Control
8. References

(File name: Project Plan.pdf)

2.7 TECHNOLOGY DOCUMENTS

Technology descriptions are separated into individual documents. They are meant to provide a basis for the understanding and use of technologies we will be utilising, and are written primarily for our own purpose.

2.7.1 GWT

GWT is, to quote its technology document, *an open-source, Java-based framework for creating AJAX applications.* The document goes into more detail on what GWT is, why we're using it, some disadvantages, and suggests some further reading.

2.7.2 SPRING FRAMEWORK

Spring is a Java framework. The technology document explains further what that actually means, what it consists of, and some advantages of using it.

2.7.3 XML

XML is a mark-up language and is among others used to write the process Data Modules. Our technology document provides some background information, the format and elements of the language, as well as an example.

2.7.4 TEST-DRIVEN DEVELOPMENT

Test-Driven Development is type of development process where tests are written before business code. The document introduces some key aspects and terms, a couple of different approaches, and pros/cons to adopting it.

2.8 SCRUM

We have produced documentation for the development methodology our group is using, Scrum. The document is meant to give an insight into what using Scrum entails, why and how we use Scrum, what having Scrum as a development methodology means for our project, and more.

Content (chapters):

1. Introduction

2. Why Scrum
3. Scrum Project Planning in General
4. More on Scrum Tools
5. The Pros and Cons of Scrum Project Planning
6. Calculations of Time Estimates in Scrum

(File name: Scrum Procedures.pdf)

2.9 BUDGET

The budget contains a list of incurred and expected expenses. The budget is approved by CORENA and it is important for us to monitor our expenses using the Budget.

3 DOCUMENTS PRIOR TO THE SECOND PRESENTATION

3.1 ANALYSIS DOCUMENT

The *Analysis Document* will contain documentation describing the background for the decisions we make regarding the analysis, and descriptions of the decisions themselves.

Content (chapters):

1. Introduction
2. S1000D Specification
3. GWT and Client/Server
4. Application Stucture
5. Reference(s)

(File name: Analysis Document.pdf)

3.1.2 S1000D DOCUMENTATION

We've conducted an analysis into the S1000D specification. It defines a structure for the process DMs, the elements it consists of, and requirements for software implementations.

The S1000D documentation is located in the *S1000D pDM* folder.

Contents (documents):

1. S1000D PDM - Data Module Sequences
2. S1000D PDM - External Application Interface
3. S1000D PDM - Logic Engine
4. S1000D PDM - Process Flow Construct - Collection of user information
5. S1000D PDM - Process Flow Construct - Preliminary requirements and Close requirements
6. S1000D PDM - Process Flow Construct – Process
7. S1000D PDM - Process Flow Construct - Variable declarations
8. S1000D PDM – References
9. S1000D PDM - Variables

3.2 DESIGN DOCUMENT

The *Design Document* will contain a closer, in depth overview of the application we are creating, S1000D Logic Engine and Renderer for Process Data Modules (SLRP). It is important to notice that this document only covers the main architecture and design of the parts that has been developed so far.

Content (chapters):

1. Introduction
2. Layout / GUI / View
3. System Structure
4. Model-View-Presenter
5. Fetch Pdm
6. Client and Server
7. Document Object Model (DOM)
8. Database System
9. Saving and Loading Sessions
10. DisplayPackage Handling
11. Expressions
12. Reference(s)

(File name: Design Document.pdf)

3.3 IMPLEMENTATION DOCUMENT

The *Implementation Document will contain our implementation process, javadoc, code standard and decisions on a lower level than design.*

Content (chapters):

1. Introduction
2. Code Standard
3. Test Driven Development
4. State Table Database
5. XML Handling and Data Binding
6. JavaDoc
7. Reference(s)

(File name: Implementation Document.pdf)

3.3.2 JAVADOC

The javadoc is the API documentation of the code in our application. This can be found under:

/javadoc/SLRP Javadoc

4 DOCUMENTS PRIOR TO THE THIRD PRESENTATION

4.1 PROJECT REVIEW

The *Project Review* is a total review of our project, how it has worked out, things we could have done better etc.

Content (chapters):

1. Introduction
2. The Project
3. Scrum Development Methodology
4. What We Could Have Done Better
5. Technical
6. Goal Achievement
7. Self-Evaluation
8. Conclusion
9. References

(File name: Project Review.pdf)

4.2 HAND-OFF DOCUMENT

The *Hand-Off Document* is a document intended for CORENA when they will take over the development of the application. It is a document which will tell where we are in the development, what the state of our application is and other important information.

Content (chapters):

1. Introduction
2. Server Components
3. Client Components
4. What We Support From The Specification
5. The Build
6. References

(File name: Hand-Off Document.pdf)

PRESTUDY REPORT

HIBU STUDENT PROJECT 2011
CORENA S1000D 4.0 PROCESS DATA MODULE RENDERER

Arild Oldervoll

Marius Haraldseth

Eirik André Eidså

Olav Brandt

TABLE OF CONTENTS

REVISION HISTORY Version Number	Date	Changes	Assigned
1	19-Oct-2010	First Official version of document	MHA
2	05-Jan-2011	- Minor changes to activity diagram. - Smaller changes in text. - Added introduction for the project. - Moved references. - Corrected references to standard. - Syntax error correction. - Document Review by all team members. - Edited introduction.	MHA

1 INTRODUCTION

Our assignment to develop a logic engine for support execution of S1000D process Data Modules version 4.0, is given by CORENA Norge A/S. This logic engine can take input from users and/or external applications, such as sensors, to decide its next step in a flow of process Data Modules. It can branch and loop through the selected process Data Module. The logic engine will be an interpreter for process Data Modules. We will make it a stand-alone logic engine in a web based application running on Jetty/Tomcat. The user of the application should be able to move back and forth through the steps using previous and next buttons. The user should be able to save the session to continue at a later time. The logic engine should support the International S1000D Specification Issue 4.0.

1.1 INTENTION OF THIS DOCUMENT

This document is a report on the result of our prestudy for the project assignment. It will describe the task in more detail, and the basic strategy for how we will complete the assignment. This will be more detailed in the *Project Plan* document.

1.2 SYNONYMS, ACRONYMS AND DEFINITIONS

Synonym	Description
Process	Set of instructions and conditions being executed
Data Module	Container of data/information.
IETP	Interactive Electronic Technical Publication - Technical manual prepared in digital form on a suitable medium.
XML	Extensible Markup Language - Set of rules for encoding documents in a machine language.
Process DM	Process Data Module
HiBu	Høgskolen I Buskerud
KDA	Kongsberg Defence and Aerospace
XP	Extreme Programming
TDD	Test Driven Development

1.3 AUTHOR(S)

Arild Oldervoll (AOL)

Marius Haraldseth (MHA)

Eirik André Eidså (EAE)

Olav Brandt (OBR)

1.4 ASSIGNED

Marius Haraldseth (MHA)

2 PROJECT ENVIRONMENT

2.1 SCRUM PROJECT MANAGEMENT TOOL

For this project we have decided to use Scrum as our development methodology. Therefore it is important for us to find a project management tool that fulfils the following three criteria:

- Scrum support, including
  - User Stories
  - Product Backlog
  - Burn-down charts
- Free to use for the entire team and for the entire project period
- Online interface and hosting

To our surprise, there are very few project management tools that provide these three things - the two last requirements are especially hard to find support for. We decided to use Pivotal Tracker. This is an online project planning tool based on agile development methods. It does not completely fulfil all the points in the first requirement, but it comes very close and is satisfactory for our use. For example; it lacks true burn-down charts, which are an important artefact in Scrum, but it has visual reports that represents the same information that a burn-down chart does. It is also free for an unlimited number of users and projects. It includes bug-tracking and it is possible to include attachments to User Stories and bugs.

After testing Pivotal Tracker for some time, we decided that it did not support our needs in a satisfactory manner. Instead, we are using a set of physical aids and tools in true Scrum-form; whiteboards and Post-it notes. We also came up with an Excel spreadsheet for the Sprint Backlog and Burndown Chart, adjusted for our requirements.

2.2 TIME TRACKING TOOL

We have found a good time tracking tool named Toggl. This is an online tool with the possibility to download an application for Android and iPhone. It makes it very intuitive to track time online and makes good reports so we can see how much time is used on each task.

2.3 DOCUMENT SHARING

To share documents between project team members, we use Dropbox. Dropbox synchronises all files in a selected folder on the computer with the other team members' files, so that everybody will see each other's files as soon as they are saved in the folder. Since a local copy is located on each team member's machine, backup of all the documents are ensured almost instantly when a file is saved. Dropbox also provides a change history for each file and the possibility of reverting to any earlier version within the last 30 days. It is also possible to restore deleted files.

Dropbox is also a secure service; all files are password protected and encrypted, and all transmissions are made through SSL.

2.4 COMMUNICATION

Our main tool for online communication is Skype. Skype is a free program that offers free communication with text, voice and video, either one-to-one or for group conversations. This is the main communication tool used by CORENA, and therefore it is also easier to communicate with anyone in CORENA directly if needed.

2.5 SOURCE CODE SHARING

For sharing and collaborating on source code we will use a version control system in the project. We have decided on using the Apache Subversion (SVN) system to share and commit source code. This is a very feature rich and free, version control system. Together with SVN we will also use TortoiseSVN to check files in and out from SVN. Both SVN and TortoiseSVN are also used by CORENA.

2.6 DOCUMENT EDITING SOFTWARE

Høgskolen i Buskerud (HiBu) requires the project team to provide documents in open file formats, like PDF or DOCX. We will be using Microsoft Word to create these documents. It is a very common office tool with a lot of features and the possibility to save the documents in several different open formats. We will go with the open format PDF for our official documents.

2.7 JAVA IDE

We will use Eclipse IDE for our development. This is a free and very feature rich IDE also used by CORENA and has support for SVN.

3 RELATED PROJECTS

3.1 RELATED INTERNAL PROJECTS AT HØGSKOLEN I BUSKERUD

To our knowledge and as far as our research has shown, there have not been any projects directly relating to the task we have been given by CORENA. However, there have been several computer science projects done at HiBu, and in our prestudy we have looked at these projects.

We have decided to focus on two previous projects; Rivet and Scrat. They are both project groups that finished in June 2010, both with a very nice end result and good grading from their sensors. These are also projects we are already familiar with since we have attended their project presentations and have been talking a lot to the team members. Because of the time proximity, they are also the two computer science projects that have been working under conditions most similar to ours, in regards to the development environment and requirements from HiBu.

Rivet's assignment was provided by Kongsberg Defence & Aerospace (KDA), where they updated a simulator of a Remote Weapon System developed by KDA. Scrat did an assignment for Luminext where they created a client/server solution for easy configuration of the Luminext Segment Controllers, as well as backup and restoration abilities for configuration and firmware updates.

Neither of these teams used the same development tools and methodologies that we are using, but the documentation they produced followed the same requirements ours do, and is thus still of interest to us. It is also interesting to read their end reports to see what they have learned during their project and what they would do differently for another project. These are things we as well should strongly take into consideration for our project.

3.2 RELATED EXTERNAL PROJECTS

Inmedius S1000D is a competitor to CORENA S1000D. Inmedius has developed an implementation of the logic engine to S1000D, called Inmedius S1000D Publishing Suite. It is the same kind of software as CORENA S1000D, but their implementation also provides an editor for the process DMs, which is not a part of our assignment. Our assignment is to develop a standalone version of the logic engine, which will then be implemented by CORENA in both CORENA IETP and in CORENA S1000D.

4 THE SYSTEM

4.1 EMPLOYER

CORENA Norge AS
Industritunet
Dyrmyrgt. 35
N-3611 Kongsberg
Norway
Phn: +47 3271 7200

CORENA S1000D is a product for configuration control, maintenance, viewing and production of technical documentation for large vessels and equipment. Customers using it are e.g. Pratt & Whitney for their aircraft engines in the F-35 combat aircraft, Boeing, Bombardier's C-series and new regional jets, Goodrich, Eurocopter (Norwegian defence's new NH-90 Helicopters), etc. Previously this was software only available for the military, but lately it has also become available for the civil market.

CORENA is one of the three leading software companies in the world in their area, and their goal and ambition is to become number one.

4.2 S1000D PROCESS DM

A S1000D process Data Module (DM) is an XML schema which describes different steps in an information process. Based on interactive user dialogs, sensors, applications and external services communicating through an interface, the logic engine decides the next DM and/or step in the sequence. Decision points (branching), looping and selective filtering are supported in the S1000D process DM. The process DM can be seen as a process flow script, where the return values define the next action. The S1000D process DM represents a procedural flow consisting of several DMs and/or steps that are sequenced.

The process DM is input which is processed in a logic engine. The logic engine is an interpreter for the process DMs, and throughout the process it defines states. Our assignment is to develop this logic engine for the execution of a process DM's components. Based on the components in the DM, an interaction between the user and the IETP should occur (see figure 4.1).

The process DM is especially well suited to represent procedural data, fault data and descriptive data. The process DM is not well suited to represent wiring data, parts data and schedule data.

To control branching, looping and context filtering, S1000D process DMs declare variables. The logic engine maintains these variables in state tables, containing them for decision points and filtering. Variables can be declared from users, external applications, presets or post-sets.

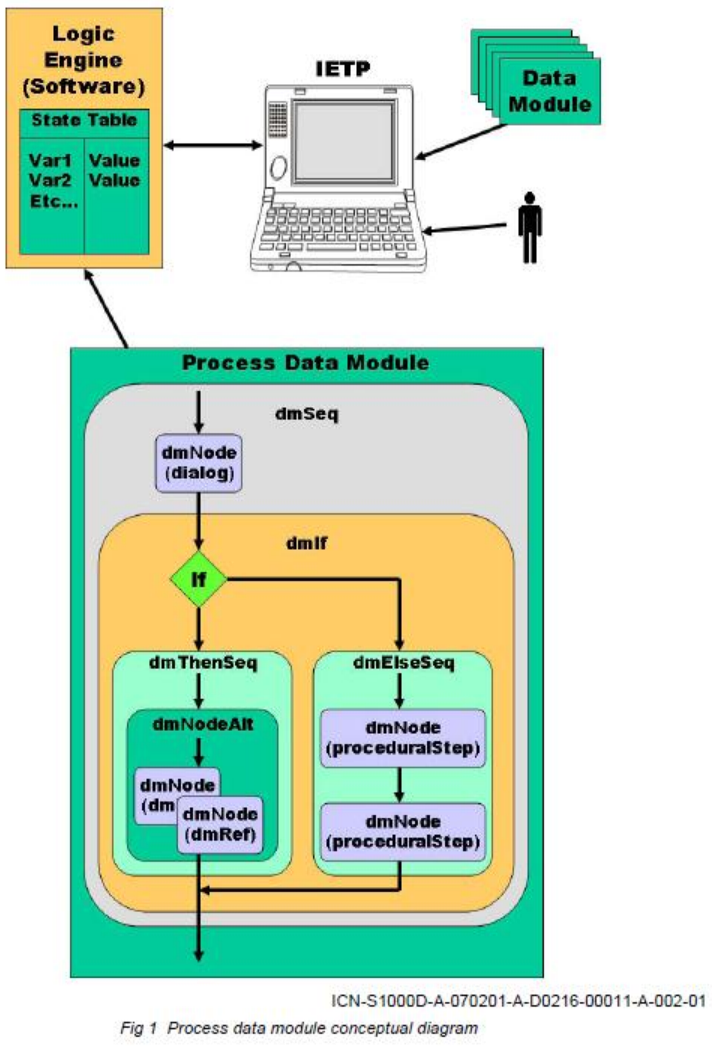


FIGURE 4.1 PROCESS DATA MODULE CONCEPTUAL DIAGRAM

4.3 BENEFITS TO THE S1000D PROCESS DM

The S1000D process Data Module adds advanced capability which you will not find in other DMs. The downside to this is complexity. Because of this it is necessary to define whether or not you need this advanced capability, or if it will just cause unnecessary effort.

The S1000D process DM is more interactive than other DMs. It can ask the user questions and interact with external applications, the logic engine stores the input data and uses it to direct the users in the procedural flow. The data in the process DM is also used to customise the output for the user’s display.

Conditional looping, if-then-else, is supported in the S1000D process DM. This to simplify the maintainer’s navigation so he/she will only get what is relevant at the current time.

5 THE ASSIGNMENT

5.1 OUR PROJECT

CORENA is expecting us to develop a logic engine which should be integrated in an independent web-based solution based on Jetty/Tomcat. The user should be able to choose a process DM, and step through this module step by step with the logic engine. Through user input, and communication with integrated sensors, the next step in the process DM is determined. The user should be presented with the choices 'previous' and 'next', so they can navigate back and forth through their choices, and/or change some of their previous answers. The user should have the option of continuing from a previously saved session.

We need to develop a method for recognising the S1000D process DMs from other DMs in our logic engine. Our logic engine should support S1000D v 4.0, and it should be possible to add support for other versions of the S1000D process DMs. In other words, we should not hard code specification support.

CORENA has a vision that our end product will be of such a high quality that they can implement it into CORENA IETP and CORENA S1000D.

5.2 REQUIREMENTS FOR THE LOGIC ENGINE

- Previous function
- Next function
- Log the states
- Save current position with history before exit
- The support execution of the logic engine should be according to the international S1000D specification issue 4.0 which can be integrated with CORENA's products.
- The implementation should be based on technology which is base for CORENA S1000D Web Client and CORENA IETP products.
- The interpreter is supposed to execute in an independent web based IETP application running on Jetty/Tomcat.

S1000D resources: http://www.s1000d.org

Relevant chapters:

- 4.11 – Information management – Process Data Module
- 7.7.1 – Guidance and examples – Logic Engine

- 7.7.2 – Guidance and examples – Process Data Module Nodes

Common technology platform for the CORENA S1000D Web Client and CORENA IETP products is:

- Java
- XML / XSL
- Spring Framework
- POJO
- Web services
- GWT / GXT (Google Web Toolkit)
- Ajax

The implementation we are going to develop should be stable, well tested and secured against failures, even if the process DMs contain faults, or are made for another version of the S1000D standard.

5.3 OTHER REQUIREMENTS

- All technical documentation which will be used later in CORENA should be in English.
- CORENA wants us to use the Scrum development methodology.

6 ALTERNATIVE SOLUTIONS

In this project, several parts are locked into the technology requirements from CORENA. The largest one is the S1000D standard, which is the core of the project. Therefore we only look at alternative solutions for the web user interface and web server alternatives.

6.1 GOOGLE WEB TOOLKIT ALTERNATIVES

CORENA wants us to use Google Web Toolkit. GWT is a development toolkit for building and optimising complex JavaScript-based applications. They do so by translating Java to JavaScript that runs in the web clients. There are other similar technologies, like Adobe Flex/Flash, Microsoft Silverlight and Java FX. They do much of the same, so here we will discuss just one of these alternative solutions: Adobe Flex/Flash.

6.1.1 ADOBE FLEX/FLASH

Adobe's technology is a highly productive, free, open source framework for building expressive web applications that deploy consistent on all major browsers, desktops, and operating systems by leveraging the Adobe Flash Player and Adobe AIR runtimes. While Flex applications can be built using the Flex SDK in Eclipse, Adobe Flash Builder software can accelerate development through features like intelligent coding, interactive step-through debugging, and visual design of the user interface layout. For the backend logic it is possible to use PHP, Java, etc.

Flex and GWT provides many of the same features. Flex is better at creating a nicer looking application faster, but GWT is also a very good tool. Most likely Java and GWT will be a faster working solution for our application, and because this will be integrated with CORENAs' systems it should support their technologies.

6.1.2 HTML5

It would be nice to use HTML5, but HTML5 is still under development and is not finished.

6.2 APACHE

Apache web server is one of the most used and safest web server applications. To get an even more light weight solution, it is possible to strip Apache down, or use another alternative completely. However, Apache is the standard and we feel it is the best choice.

7 CONSEQUENCES FOR EACH PARTICIPANT OF THE PROJECT, IF WE START OR DON'T

7.1 PROS AND CONS FOR CORENA

If we start this project it means that CORENA might take some load off of their shoulders, not having to develop this implementation of the logic engine for the S1000D process DMs themselves. CORENA also gets an opportunity to see if we are capable, and if we can be worth to hire. CORENA will save a lot of money if we can develop a good implementation for them.

If we don't start the project, CORENA will have to do the work themselves, and they already have a lot to do. They will not get to see us work, and will therefore not know more about us in consideration of a future cooperation.

7.2 PROS AND CONS FOR US

If we start this project it means that we get the chance to develop a process Data Module solution for CORENA, which has been requested by one of their customers. This is a complex task, and will give us a lot of understanding on how to work in a larger software project. We get the chance to work on a project which will be delivered to this customer, if we manage to successfully develop an implementation that is acceptable. We get the opportunity to work in a professional environment, and we have been offered to use the CORENA facilities during our project. CORENA will give us good support, and this gives us excellent opportunities to improve our skills, both in scripting- and programming languages, project management working with the Scrum development methodology, and maybe end up in a job offer at the end of the project. When doing this for CORENA we will also see a little bit of how it is to work for them. This will give us a picture of what the place is like to work at.

If we don't start this project, we will probably get in trouble finding another project in such a short time. We will also send out a message saying we are not in control, and should have done better research beforehand, or that we have not done what we should have. This will make a bad impression, and probably lower our grade for this project. It will also be a mental factor of strain for us at the end of this semester.

8 PROBLEMS AND LIMITATIONS

The biggest problem, or more accurately - limitation, is time. Originally, the application also contained an process Data Module editor. However, our group consists of only four people, so the scope of the project has been adjusted down. This is done in order for us to be able to produce a complete and high quality product within the given timeframe.

The team is fairly inexperienced with several of the technologies we will use in this project (XML/XSL, Spring Framework, POJO, Web services, Google Web Toolkit and Ajax). However, we have experience with web-programming in general, and also with Java. Many of the technologies are also based on Java, so after all; the chance of finishing the project in a satisfying manner is high.

8.1 MAIN MILESTONES

The milestones in this project are the three presentations.

- Presentation 1: 2. Week of 2011
- Presentation 2: Between 14. March and 15. April 2011
- Presentation 3: Beginning of June 2011

9 HIGH LEVEL ACTIVITY OVERVIEW

Activity Number	Activity Name
100	**PROJECT MANAGEMENT**
101	PLANNING
102	ECONOMY
104	PROGRESS REPORTS
105	TIME TRACKING
200	**DOCUMENTATION**
201	TEMPLATES
202	VISION DOCUMENT
203	PRESTUDY REPORT
204	REQUIREMENTS SPECIFICATION
205	TEST SPECIFICATION
206	TECHNOLOGY DOCUMENTATION
207	USER MANUAL
208	PROJECT PLAN
209	SCRUM DOCUMENTATION
299	OTHER DOCUMENTATION
300	**MEETINGS**
301	MEETING PREPARATION
302	SCRUM MEETINGS
303	STATUS MEETINGS
304	MEETING MINUTES
305	PLANNING MEETINGS
399	OTHER MEETINGS
400	**RESEARCH**
401	TECHNOLOGY
402	TOOLS
499	OTHER RESEARCH
500	**ANALYSIS AND DESIGN**
501	ANALYSIS
502	DESIGN
600	**IMPLEMENTATION**
601	PROTOTYPING
602	PROGRAMMING
700	**QUALITY ASSURANCE**
701	TESTING

702	DOCUMENT REVIEW
800	**GENERAL**
801	WEBSITE
802	PRESENTATIONS
803	STANDARDS
804	LECTURES
899	OTHER

10 PROJECT DEVELOPMENT MODEL

We have chosen Scrum as our project development methodology.

Our choice was made on the basis of some key characteristics:

- It is an iterative and incremental approach, as opposed to being linear and sequential, like the waterfall model. This means that the project will be divided into several iterations (Sprints), where modules of the product (beginning with the highest prioritised part) will go through an entire development process from inception to implementation and testing.
- One important property of this is the minimisation of the consequences by unforeseen and project-ending events. If we are unable to fully complete the project, the end result will be a product where, for instance, the most important 70% of the requirements will be 100% finished and functional, instead of a product where 100% of the requirements will be 70% finished.
- It is a lightweight and agile model fit for small development teams. Our primary alternative is the Rational Unified Process (RUP), which is also an iterative and incremental model - and thus shares several of the positive attributes with Scrum. However, RUP is a heavier and more comprehensive model, features that are not particularly desired by our small team.
- Another possibility is the so called Extreme Programming model (XP). Like Scrum, XP belongs to the *Agile* family of development methods, but is *too* lightweight to be the ideal model for our project and the documentation it requires.
- Our project provider, CORENA, uses Scrum as their development model, and they have expressed a wish for us to do so as well.

Following are some central features of Scrum.

10.1 ROLES

- Scrum defines a number of roles, all of which are divided into two main categories of stakeholders; **pigs** and **chickens**. The pigs are the people who are committed to the product and its development, while the chickens are the customers and non-team managers.
- The “Pig” roles are: the ScrumMaster (the project manager in traditional methodologies), the Product Owner (the voice of the customer – responsible for prioritising requirements) and the Team (the actual developers).

10.2 SPRINTS

The development work in Scrum is done in iterations called *Sprints*. This is where the "iterative and incremental" part really shows itself. A Sprint typically lasts 2-4 weeks and should usually deliver at least one fully completed feature of the product, potentially ready for release. We will use Sprints lasting 3 weeks.

It is not allowed to alter the product backlog the Team has committed to during a Sprint.

Sprints are *time boxed*; they cannot last longer than planned. If some of the work were not completed, the requirement it belonged to would have to be completed in another Sprint.

Every day during a Sprint, a 15-minute meeting is held, the Daily Scrum Meeting, where each team member explains what was done since the last meeting, what will be done until the next meeting, and if there are/were any problems.

There are other meetings as well, including a planning meeting at the beginning of a Sprint, and a retrospective meeting when a Sprint has been completed.

10.3 LISTS/ARTEFACTS

There are three main types of documents, or artefacts, in the Scrum model.

- The **Product Backlog** is a list of requirements sorted by priority. This list describes *what* the product should do, written in a non-technical "User Story" form. Each requirement has an accompanying *priority*, set by the Product Owner, and a *development effort*, set by the Team.
- The **Sprint Backlog** contains *tasks* to be performed during the Sprint.
- The **Burn Down** chart shows what work has yet to be completed in the Sprint Backlog.

11 RESOURCES REQUIRED IN THE PROJECT SPLIT ON ACTIVITES

Resource ID	Name
R100	**Tools**
R101	Project Planning Tools
R102	Excel
R103	Word
R104	Toggl
R105	Eclipse
R106	SVN Tool
R107	PowerPoint
R108	Visual Paradigm
R200	**Human Resources**
R201	Team
R202	Internal Sensor
R203	External Sensor
R204	Internal Mentor
R205	External Mentor
R206	Product Owner
R207	Other CORENA Resources
R300	**Hardware**
R301	Webserver
R302	Apache server
R400	**Documentation**
R401	Documentation from CORENA
R402	Documentation from HiBU
R403	Previous Project's Documentation
R404	Other Technical Publications
R405	International S1000D Specification Issue 4.0, 2008-08-01
R500	**Facilities**
R501	Meeting Room
R502	Presentation Room
R503	Office at HiBu
R504	Office at CORENA

Activity	Resource(s) Required
101 – Planning	R101, R402, R403
102 – Project Plan	R101, R401, R402
103 – Economy	R102
104 – Progress Reports	R102, R103
105 – Time Tracking	R102, R104
201 – Templates	R103
202 – Vision Document	R103, R401, R402, R403, R405
203 – Prestudy Report	R103, R401, R402, R403, R405
204 – Requirement Specification	R103, R201, R206, R207, R401, R403, R405
205 – Test Specification	R103, R201, R401, R403,R405
206 – Technology Documentation	R103, R401, R405
207 – User Manual	R103
299 – Other Documentation	R103
301 – Meeting Preparation	R103
302 – Scrum Meetings	R201, R206, R501
303 – Status Meetings	R201, R204, R501
304 – Meeting Minutes	R103
399 – Other Meetings	R201, R501
400 – Research	R103, R205, R401, R402, R403, R404, R405
501 – Analysis	R103, R108
502 – Design	R103, R108
601 – Prototyping	R105, R106, R302, R404, R405
602 – Programming	R105, R106, R302, R404, R405
701 – Testing	R105, R302
702 – Document Review	R103
801 – Website	R301
802 – Presentations	R110, R502, R201-R205
803 – Standards	R103, R401, R402, R403
804 – Lectures	R402

12 ACTIVITIES AND RESOURCES UNTIL THE PROJECT PLAN IS READY

Activities	Resources
101: Planning - **Create a project plan.**	R101
103: Economy - **A budget containing estimated project expenses and economic resources.**	R102
201: Templates - **Template for the requirement specification.** - **Template for the test specification.**	R103
204: Requirement specification - **Defining and prioritising all requirements in collaboration with CORENA.**	R103, R201, R206, R207, R401, R403, R405
205: Test specification - **Create test strategy.** - **Create the test specification.**	R103, R201, R401, R403, R405
399: Other Meetings - **Requirement meeting(s) with CORENA.** - **Meeting with internal mentor.**	R201, R204, R206, R207, R501
402: Tools - **Determine a project tracking tool to use.**	R106

13 REFERENCE(S)

1. *Assignment description,* CORENA Norge AS, 2010

2. *International Specification for technical publications utilizing a common source database S1000D*, Technical Publications Specification Maintenance Group (TPSMG), 01-Aug-2008 (Issue 4.0)

3. *Agile Project Management with Scrum*, Ken Schwaber, Microsoft Press, 1-Feb-2004 (1.edition), ISBN-13: 978-0735619937

4. *Agile Software Development with SCRUM*, Ken Schwaber and Mike Beedle, Pearson Education, 21-Mar-2008 (1.edition), ISBN-13: 978-0132074896

5. Scrum (development), Wikipedia, http://en.wikipedia.org/wiki/Scrum_%28development%29 (last visited 12-Oct-2010)

PROJECT REVIEW

HIBU STUDENT PROJECT 2011
CORENA S1000D 4.0 PROCESS DATA MODULE RENDERER

____
Arild Oldervoll

____
Marius Haraldseth

____
Eirik André Eidså

____
Olav Brandt

TABLE OF CONTENTS

REVISION HISTORY			
Version Number	Date	Changes	Assigned
1	01-Jun-2011	First official version	AOL

1 INTRODUCTION

Our assignment to develop a logic engine for support execution of S1000D process Data Modules version 4.0, is given by CORENA Norge A/S. This logic engine can take input from users and/or external applications, such as sensors, to decide its next step in a flow of process Data Modules. It can branch and loop through the selected process Data Module. The logic engine will be an interpreter for process Data Modules. We will make it a stand-alone logic engine in a web based application running on Jetty/Tomcat. The user of the application should be able to move back and forth through the steps using previous and next buttons. The user should be able to save the session to continue at a later time. The logic engine should support the International S1000D Specification Issue 4.0.

1.1 INTENTION OF THIS DOCUMENT

This document is meant to give a report of our project all the way through. It will give an overview on how our project has been accomplished, how our development model Scrum has been working out for us, challenges and our thoughts of the project from a retrospective point of view.

This document will explain what we did well, and what we could have done in a different way.

1.2 SYNONYMS, ACRONYMS AND DEFINITIONS

Synonym	Description
Process	Set of instructions and conditions being executed
Data Module	Container of data/information.
process DM	Process Data Module
HiBu	Høgskolen i Buskerud
XML	eXtensible Markup Language
GWT	Google Web Toolkit - Development toolkit framework for building complex browser based applications for Java.
GXT	Ext GWT (Extension for GWT)
SQL	Structured Query Language
UML	Unified Modelling Language
MVP	Model-View-Presenter - Architecture Pattern
RPC	Remote Procedure Calls
TDD	Test Drive Development
GUI	Graphical User Interface
CSS	Cascading Style Sheets
HTML	HyperText Markup Language

1.3 AUTHOR(S)

Marius Haraldseth (MHA)

Arild Oldervoll (AOL)

Eirik André Eidså (EAE)

Olav Brandt (OBR)

1.4 ASSIGNED

Marius Haraldseth (MHA)

2 THE PROJECT

The idea of this project started when Arild and Marius worked at CORENA Norge AS from January 2010 and through the summer. During that time they were presented with a project proposal for the bachelor thesis at Høgskolen i Buskerud. They thought the project sounded interesting and challenging, so they presented it for other students at HiBu. The group gathered quickly, and we ended up with a group of four people. Because of the projects size, we had actually wanted to be a full group of six people, but we also wanted to only have group members that we knew were hard-working and dedicated as we realised that it was a big project.

The project would have its basic in Java, but it included several unfamiliar technologies and techniques like Google Web Toolkit (GWT) with the GXT extension, Spring Framework, XML, Ajax in addition to the comprehensive S1000D specification.

When we started the project it was a discussion if we should implement our application in their existing software, CORENA IETP, but after some meetings and discussions we decided on that we should make it a standalone application with communication against CORENA IETP for loading process Data Modules and other data.

Since CORENA uses Scrum as their development method they wished for us to do the same, we considered this against other methodologies and agreed on that Scrum would probably be a good choice for us. Now we can look back at ten meaningful, challenging and demanding months.

CORENA required that all documentation which they would use should be in English. We therefore decided to write all of our documentation in English.

The name of our group, “Ameo” comes from the first letter of the group members first names:

- **A**rild – **M**arius – **E**irik – **O**lav

The name of our application, “SLRP”, is made from:

- **S**1000D **L**ogic Engine and **R**enderer for **P**rocess Data Modules

It is pronounced: [sl3:p]

2.1 CORENA NORGE AS – OUR EMPLOYER

CORENA is a company with offices at several locations around the globe. One of their offices is located in Kongsberg with approximately 30 employees.

CORENA is one of the three leading companies regarding software for viewing, configuration management, maintenance and production of bigger maintenance documentation in the world. They hold resources which might the world’s strongest collection of S1000D domain experts.

CORENA is working with customers like Pratt & Whitney, Bombardier, Goodrich, Eurocopter, BAE and KDA.

“Complex reality made easy”

3 SCRUM DEVELOPMENT METHODOLOGY AND ACCOMPLISHMENT

Throughout our project we have used Scrum as our development methodology. The first reason we thought of Scrum was because CORENA use it as their methodology. CORENA suggested that we used it, and we took a deeper look into it and thought it looked interesting, and we wanted to try something new and got the school's approval.

In the beginning we used much time gathering information about Scrum and trying it out. During the first semester we got pretty much into it, and by the first presentation we had gotten rid of most of the initial problems.

Because we are the first group at HiBu to use pure Scrum, this has been a challenge both for us and for the school, because Scrum doesn't produce a lot of documentation and without documentation the foundation for our grades would be very limited. This required some more documentation from us than what's traditionally produced by our development methodology. So a real challenge has been to find a balance between Scrum's documentation requirements and the school's documentation requirements, the solution to this has been to include the documentation as part of the product we are developing. Arild and Olav have more experience regarding documentation than Marius and Eirik since they had the subject "Basic Game Programming".

In a real Scrum project, we would have started developing almost immediately. We would probably have used less than one Sprint to map the most basic User Stories for what we would develop before the real development started.

We had our own project room at HiBu, room C171. This has been a perfect room for our project, just big enough.

In January, we got the horrible message that Eirik had got cancer, of type osteosarcoma. This is a very aggressive type of cancer, and Eirik started an intensive treatment which would last for approximately six months. This would of course affect our project as well. Since we used Scrum which is a very agile development method we have solved this better than we could have done with any other project model. We had not used a lot of time planning for things who would now have been a waste. Instead we have adjusted our work from that point and onward. None of the work we have done was a waste; the only thing that was influenced is how far down the Product Backlog we will reach. When that is said, Eirik has done a fantastic work even though he has been ill and absent for treatments a lot. He has met up with us as often as he has been able, and besides of that we had a continuous communication through e-mail and Skype. Because of the agility of Scrum we have been able to include Eirik as much as he has been able to. There has always been a task for him when he has had energy to do it. At this point in time all tests indicate that Eirik will recover from the cancer. Except from Eirik, the rest of us have kept healthy through the whole project, without any absence from the project.

Because of the size of our assignment we really could have used more people in our project group, at least one more would have been very helpful. Our selection of people for the group was not random either, which makes it even harder to get enough people. We got together four people which all knew what to expect from each other; skilled and hardworking students.

Another big advantage Scrum has given us is that we have been agile regarding new requirements which have come up while working with the S1000D specification. The specification is very big (almost 2800 pages) and complex. This has been a challenge, but adjustment along the way has not been a problem. Scrum development methodology makes this fully legal.

Every morning we have started with our daily Scrum meetings, which have been very good in order to keep up with each other's work, especially when Eirik has been gone, we could still meet over Skype and keep up to date. Other than this we have mostly been sitting next to each other and had a good cooperation so we have mostly been quite aware were the rest of the team have been in their work.

In the beginning of each Sprint we had a Sprint planning meeting doing the planning for the upcoming Sprint. This meeting was time-boxed to four hours. In these meetings we took User Stories from the Product Backlog after how much velocity we had calculated for that Sprint, and we split these User Stories up into several tasks. The velocity for each User Story is calculated using poker planning.

In the end of each Sprint we had a Sprint demo for CORENA. This has also been a good place for us to discuss solutions with them during the project. Since each Sprint has been approximately three weeks, this has been each three week period. We had some meetings besides this as well, especially in the start of the process to secure that we were developing the right product. Our external mentor Tommy Sivertsen has also taken part in some of our daily Scrum meetings, either by Skype, or he has showed up at HiBu. CORENA has been very easy to ask in situations outside of these meetings when we have been needing assistance or their opinion in some matters. We also had a Sprint Retrospective by the end of each Sprint. Up to and including Sprint 7 we produced a "Sprint Retrospective" report for the Sprints, but after this we changed the way we held the retrospective to be more verbally, which did not produce a report.

Scrum has also been new to our internal mentor Karoline Moholth, but she has been reading up on Scrum and we have kept a continuous dialog in our weekly status meetings regarding Scrum. This has solved some of the uncertainties in the best way, and it has worked out very well. In our opinion we have really succeeded with Scrum, and with everything that has happened during our project we got to explore some of the real advantages of Scrum and agility. We're in the opinion of that we have delivered a good project with this methodology.

Our internal communication and cooperation within our group has worked splendidly. When Eirik got cancer we got to test this more than we could have wanted, but we're in the opinion that we have got through that situation the best way we could. This situation has not lead to any bad motions between any of us. We have supported each other in a good way through it all.

For our project management we have the Product Backlog which we have in an Excel spread sheet. In this we have a prioritised list of User Stories, or requirements which we work our way from top to bottom. As we have worked with the specification and requirements for the User Stories, we have discovered new requirements. It has led to splitting of some User Stories into smaller ones. The Product Backlog has been our main requirements list, and within each Sprint we have developed tasks for each User Story, gathered in a Sprint Backlog. In this Sprint backlog we also had in an Excel spread sheet showing the sprint burndown, and in addition we used a whiteboard in our project room as our main task board. Our main background for project management comes from the introduction lessons in this subject, “Bachelor Thesis with Project Management”. In these lessons we had lectures where we learned a lot about project management, documentation, test strategies and other requirements for us in such a project. This was good information to bring into the start of the project.

Other than this we have used a web based time logging tool for logging time on different activities, Toggl. This tool has worked out very well, we have the opportunity to get reports from it, and split up into several “projects” and “activities”. We have also logged on task level in our Sprint Backlogs.

Together with Scrum we have used test driven development (TDD) for as much as it were appropriate for. We discussed this with CORENA, and they also had the vision that it sometimes did not fit, especially for some of the GUI programming. But for all parts not a part of the GUI, we have used TDD.

Pair-programming is common in Scrum, but because of our small group this was difficult to do in bigger parts of the time, but we did some of it in more complex challenges.

4 WHAT COULD WE HAVE DONE BETTER?

4.1 DECIDE ON STANDARDS

Something we have experience through this project is that before we start working on something, we should have decided a standard for what we do. There were several times in the start of the project were we all started working on something, e.g. writing a type of document etc. were we pretty soon noticed that the one of us did it in a different way than one of the others. This was something we could have clarified better at an even earlier stage than we did. It did lead to some editing after hand.

4.2 UNDERESTIMATES

Estimation of correct work effort for the different tasks is new to all of us. When we in the start were producing mostly documentation it went very well, and we thought we had the hang of it. But, when the development started, we came out for the typical programmer's error; in the estimate of developing tasks we thought most of the things were less complex than what they actually were. Very soon we got the taste of this, and adjusted our estimates. Even then we often underestimated development tasks.

So what would we done different? We would add even more time to our estimates regarding development.

4.3 BREAKDOWN OF USER STORIES INTO TASKS

We would have broken down the User Stories to tasks in a slightly different way, now with our new experience in estimating and planning.

4.4 EARLIER PROTOTYPING

We would have started even earlier prototyping of some parts of the functionality for the application. Sometimes we discussed a little too much before we started developing and prototyping, and in this phase we often came up with either other challenges we hadn't thought of, or smart solutions. This was something we later in the project changed, and used more prototyping in parallel with analysis and design. This is the way it should be in Scrum as well.

5 TECHNICAL

The technical was, in addition to our development methodology, the absolutely biggest challenges of our assignment. The logic engine and renderer we should develop is based on a specification which is very big and complex. It is almost 2800 pages, and to retrieve all of our requirements (in our project User Stories) from this specification has been a real challenge. We used one Sprint just to analyse the most important parts of this application and document this, but it turned out that one Sprint was not sufficient to achieve accurate estimates, and combined with our lack of experience in estimating this led to too low estimates.

Therefore, to develop the logic engine and support for elements in this specification is a vast task which we have been noticing.

5.1 ANALYSIS AND DESIGN

When it comes to analysing and designing our tasks, the knowledge from our subject "Analysis and Design" came in handy. In this subject we learned a lot about the different forms of diagrams which we have used to explain our application structure and configuration. We have used UML for this, and this was the main aspect of this subject.

We have developed a lot of analysis documents from the specification, because of the size and complexity of the specification we used a whole Sprint for doing this. Even though we used so much time on the specification, there were hidden complexity which we at later stages discovered which also made our application even more complex and made us re-analyse the design.

Most of the architecture analysis we did together as a group. This gave us a good and stable architecture with input from all of us. We worked a lot on a common whiteboard.

5.2 CLIENT – SERVER

To develop an application with a client – server relationship has been a bigger challenge than just to develop a desktop application without this relationship. We had to make RPC services between these two components, and for this we used the command pattern which implements an asynchronous call from the client to the server with a following callback.

5.3 TECHNIQUES

5.3.1 MAIN PROGRAMMING LANGUAGE

In this project we have used Java as our main programming language in addition to some XML scripting. We have used frameworks like Google Web Toolkit (GWT) with the Ext GWT (GXT) extension. We tried out Spring in the start of the project, but after some research later on we discovered that for what we were going to develop we didn’t have to use it. We used regular Java objects instead.

Most of our basic Java knowledge comes from our Java courses in the first two semesters in this three year bachelor education. But parts of it also come from programming in subjects like “Analysis and Design”, “Network”, “Operating Systems” and “Real Time Systems”. We have also used the knowledge from “C++ Programming” regarding templates, as we have used generics in Java.

Since we used Java, this was an advantage to us because this was probably the programming language which we knew best.

5.3.2 DATABASE

For our database system we have used a lightweight Java, JDBC and SQL based database from Apache, Derby. Together with this we have used Hibernate as an object-relational mapping system. This mapping system has given us some challenges with difficult error messages and a mapping and configuration system which were new to us.

When working with, and developing the database aka the state table in the application we got to use a lot of our knowledge from the “Database” subject at HiBu. Especially when it comes to modelling, and knowing how the database should work.

5.3.3 DATA BINDING

For data-binding between XML and Java classes we have used the XStream library which serializes objects back and forth from XML to Java. We use a lot of XML since the process Data Modules are written in XML.

5.3.4 PATTERNS

We have used several patterns in our application, patterns like singleton pattern, command pattern, the event-driven messaging design pattern and the model-view-presenter (MVP) architecture pattern.

We have some general knowledge of patterns from the “Analysis and Design” subject, but most of this we had to learn through the project.

5.3.5 RPC SERVICE – CLIENT TO SERVER

To communicate between the client and the server we use the command pattern. This pattern uses asynchronous RPC services. RPC’s is something we all have a little knowledge from the subjects “Operating Systems” and “Network”, but the use of it we had to learn.

5.3.6 HTML, JAVASCRIPT AND CSS

We gathered some useful knowledge through the subject “Programming for internet”, because we are developing a web application. We have used a little JavaScript, CSS and html.

5.4 TESTING

Testing was something we had no experience with from the past at all. In our classes at school we are told that testing is a good thing, but nothing about how to do it. We learned some test strategies in the “Project Management” course, but the actual use was brand new to all of us. This follows therefore that we have learned a lot about testing. We have written a lot of automatic JUnit4 unit tests. We have also used EasyMock3 library to mock objects in the tests.

On the client side we had to extend *GwtTestCase* in some tests to test classes which generate dynamic GUI (Widgets). It was something we did not desire, as testing with GwtTestCase is extremely slow.

5.5 EXCEPTION HANDLING

This was also something which was pretty new to us, we have been told to use it, but the actual use itself we had little knowledge of. So we have gathered some knowledge about this as well throughout the project.

6 GOAL ACHIEVEMENT

We have reached nearly as far as we had planned in the Product Backlog, something we are very proud of. We have developed a partly functional logic engine with support for the most regular operations, elements and functions from the S1000D specification. We have developed an application that is solid, extensible and of good construction so it is ready to be used and developed further on by CORENA.

We have also done a lot of analysis and research regarding the specification of the process Data Modules, which CORENA has indicated is of great value for them in further development of the application and in one of their other projects which is to generate an editor for these process Data Modules.

7 SELF-EVALUATION

7.1 ARILD OLDERVOLL

For me, as the project leader and Scrum master, it was really exciting from the beginning of the project to see how Scrum would work out. None in the group had any previous experience with it, neither did our internal mentor, and pure Scrum has never been used in a HiBu bachelor project before us. Therefore I'm relieved that I'm able to say that Scrum was a great benefit and success for our project. Since my role as a Scrum master isn't a team leader as in a traditional project methodology, but a "facilitator" for an effective team, the success of the project was dependent on a committing and self-organising team, which we definitely had.

I have really appreciated this chance to be part of the complete development cycle from initial assignment and vision discussions to the hand-off of the project. The project has taught me a lot about everything from planning, talking to customers and creating requirements, to the analysis, design and implementation of a larger application, and I have had the chance to utilise a broad span of knowledge and skills acquired during this course in this project.

Eirik's sudden cancer diagnosis got us all startled. But I'm glad we've had the team we had, which has supported him and each other and has been willing to put in an extra effort in order to best accommodate for Eirik's participation and to limit the impact on the project, and I believe that it's thanks to good friends and team members, and the use of Scrum, that we have been able to adjust well to this situation. I also want to add that Eirik's effort has been nothing but impressive, and very motivating for the rest of us.

This project has had a handful of challenges, but together we have been able to handle them. I would say that some of the biggest challenges have been the use of Scrum, use of a lot of new technologies like GWT, Hibernate, etc. and the comprehensive S1000D specification, which easily can hide a small detail that has a big impact on our project. This has led to a strong focus on creating a solid architecture and underlying functionality, which I am sure CORENA will have a great benefit from when they overtake our product.

7.2 MARIUS HARALDSETH

This project has been a real challenge and a new experience for me. I'm very glad to be in such a positive and skilled group, to work with these people has made it all much simpler. Even Eirik with cancer has been in a better mood and more positive than I could have imagined. The composition of these points has been of great value for our success through this project.

During this project I have gained a lot of knowledge when it comes to working in a project, and I have learned a lot about testing and working with the test driven development way. This was something I wanted to be more familiarised with before and I know got a perfect opportunity to learn. To think this way is totally different than what I've been used to, and I can now see why they say that it generates more stable and fault free code. Our project was a pure software project, and this was exactly what I wanted the most.

My responsibilities in the project have been analysis and documentation. The analysis of this assignment has been a real challenge because we had a very massive and complex specification to proceed to. The documentation have been a little difficult to determine how far we should go in this Scrum project, where the balance should be, but now in the end I am very happy and proud of what we have both developed of the application and the documentation we have produced.

I think Scrum has been a very good development methodology, and that we have kept to it as much as we possibly could do. This way of doing a project is also a new way in concern to everything I have learned here at HiBu.

7.3 EIRIK ANDRÉ EIDSÅ

In this project I have especially learnt that things do not always go as one believe, plan or hope. I also see how privileged you are when you're in a group that is so incredibly nice and good at adapting itself when you are so unfortunate to end up in a hospital in the middle of the project. This has made me able to do what I could and been capable of.

The project methodology we chose has made it relatively easy to adapt to this situation, and I feel we've all learned a lot from it! I got sick pretty much exactly at the time the development was about to start. Because my presence got pretty unstable, I was not calculated for the planning of the Sprints, but rather considered a bonus if I managed to get anything done on a User Story I was assigned. Which I did!

A lot of my responsibilities for this entire project have been the graphical parts of basically, everything. Logos, presentations, documentation, GUI, etc. This is work I enjoy a lot and I think we succeeded pretty well with it!

7.4 OLAV BRANDT

The work on SLRP has been a terrific opportunity to utilise the skills and knowledge I have accumulated during my three years as a student at HiBu. As a soon-to-be computer engineering graduate, the pure software nature of our project assignment was an ideal match. As someone interested in the entire process of software development, getting to follow an entire life cycle of such a large project as a key participant has been an excellent experience.

The technical challenges have been of multiple varieties and not far between. I'm especially partial to those existing in the higher levels of abstraction, of which there have been plenty during this project. Making sure the solutions we choose are well thought out and properly designed has been an area I've spent a lot of time and energy.

Of course, in a team of our size, and especially with a team using Scrum, the work doesn't stop there. Implementing the design also makes up a large part of the time I've spent on development, and in reality the process is more organic and intertwined than it might appear on paper. Going back and forth between analysis & design on the one side, and prototyping and implementation on the other, has been the MO for the majority of the time.

The ability to bounce ideas between team members - getting and providing input amongst the team, is something I value highly, and something I feel we've accomplished in the project. Every member is a highly capable developer and valuable resource, and the cooperation between us has been a high point.

The challenges have not been all-technical, with Eirik's situation standing out from miles away. Losing much-needed project hours has been a real loss, but his excellent attitude and continued effort has boosted the group's morale and allowed us to push that much harder. Given his situation, he has performed well above all expectations, and in the end, all of us as a group have gotten further as a result.

8 CONCLUSION

All in all, this project has been a real challenge, and a really good experience. Our team has managed to stay focused through some stressful and more difficult times. We have all learned a lot about management, teamwork, planning and work estimation. We are satisfied and proud of the final result we have achieved and the way we have run the project.

PROJECT PLAN

HIBU STUDENT PROJECT 2011
CORENA S1000D 4.0 PROCESS DATA MODULE RENDERER

CORENA

HØGSKOLEN i Buskerud

ameo

____
Arild Oldervoll

____
Marius Haraldseth

____
Eirik André Eidså

____
Olav Brandt

TABLE OF CONTENTS

REVISION HISTORY **Version Number**	**Date**	**Changes**	**Assigned**
1	05-Jan-2011	First Official Version of Document	AOL
2	07-Mar-2011	Second official version. - Adjusted estimates for hours and Sprints. - Added section 7.2.2	AOL
3	01-Jun-2011	Updated estimates for hours and Sprints. Set new delivery date and added third presentation date	MHA

1 INTRODUCTION

Our assignment to develop a logic engine for support execution of S1000D process Data Modules version 4.0, is given by CORENA Norge A/S. This logic engine can take input from users and/or external applications, such as sensors, to decide its next step in a flow of process Data Modules. It can branch and loop through the selected process Data Module. The logic engine will be an interpreter for process Data Modules. We will make it a stand-alone logic engine in a web based application running on Jetty/Tomcat. The user of the application should be able to move back and forth through the steps using previous and next buttons. The user should be able to save the session to continue at a later time. The logic engine should support the International S1000D Specification Issue 4.0.

1.1 INTENTION OF THIS DOCUMENT

This project plan contains all of the information on how we are planning on executing the project. It describes the assignment, goals, limitations, and the plan and strategy from our first presentation in the beginning of January and until delivery of the finished product in mid-May.

1.2 SYNONYMS, ACRONYMS AND DEFINITIONS

Synonym	Description
Data Module	Container of data/information.
Process Data Module	One of the module types specified in the S1000D standard. Contains logic rules and statements combined with data modules
PDM	Process Data Module
HiBu	Høgskolen i Buskerud
IETP	Interactive Electronic Technical Publication
XML	eXtensible Markup Language

1.3 AUTHOR(S)

Arild Oldervoll (AOL)

Marius Haraldseth (MHA)

1.4 ASSIGNED

Arild Oldervoll (AOL)

2 THE TASK

A process Data Module (process DM) is an XML schema that describes different steps in an information process. Based on input from interactive user dialogs, sensors, applications, external services and states saved from previous steps, the logic engine decides the next DM and/or step in the sequence. Decision points (branching), looping and selective filtering are supported in the S1000D process DM. The process DM can be seen as a process flow script, where the return values define the next action. The S1000D process DM represents a procedural flow consisting of several DMs and/or steps that are sequenced.

The process DM is input data that is processed in a logic engine. The logic engine is an interpreter for the process DMs, and throughout the process it defines states. Our assignment is to develop this logic engine for the execution of a process DM's components. Based on the components in the DM, an interaction between the user and the IETP should occur (see Figure 2.1).

The process DM is especially well suited to represent procedural data, fault data and descriptive data.

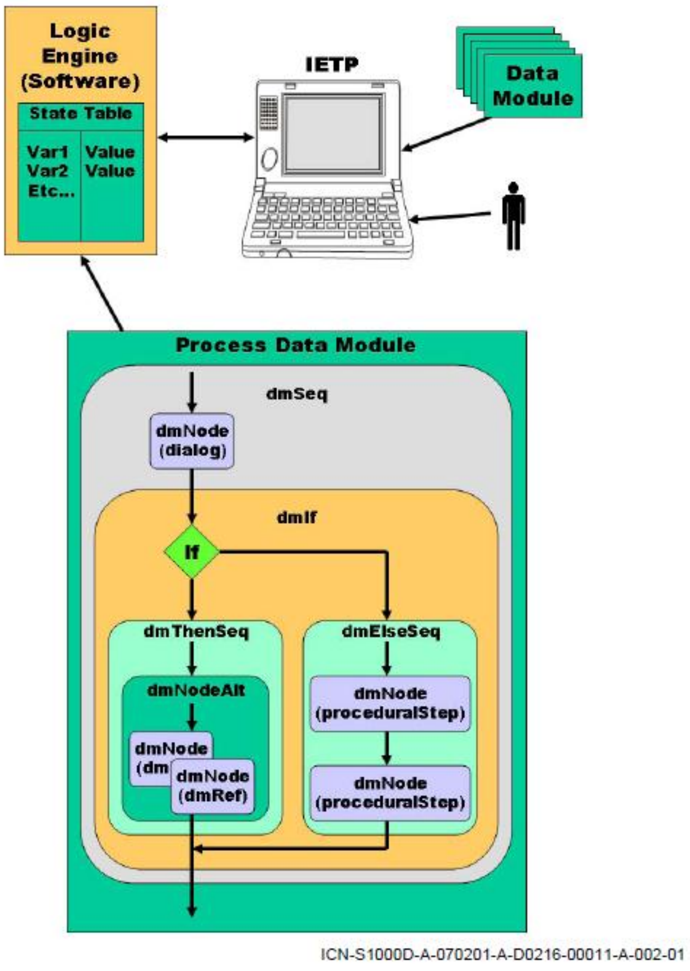


FIGURE 2.1 PROCESS DATA MODULE CONCEPTUAL DIAGRAM

2.1 WHAT ARE THE REQUIREMENTS FOR THE LOGIC ENGINE

- Previous function.
- Next function.
- Log the states defined through the process.
- Save current position with history before exit.
- The support execution of the logical document should be according to the international S1000D specification issue 4.0 that can be integrated with CORENA's products.
- The implementation should be based on the same technology that the CORENA S1000D Web Client and CORENA IETP products are based on.
- The interpreter is supposed to execute in an independent web based IETP application based on Jetty/Tomcat.

S1000D specification resources [8]:

Relevant chapters:

- 4.11 – Information management – Process Data Module.
- 7.7.1 – Guidance and examples – Logic Engine.
- 7.7.2 – Guidance and examples – Process Data Module Nodes.

Common technology platform for the CORENA S1000D Web Client and CORENA IETP products is:

- Java
- XML / XSL
- Spring Framework
- Web services
- Ajax, GWT / GXT (Google Web Toolkit)

2.2 DOCUMENTATION

HiBu and CORENA require some documentation from us to be delivered before each presentation. A full description of the documents can be found in *Document Overview* [5]. Table 2.1 shows a list of the documentation that will be delivered:

Document	Delivered by presentation #
Vision Document	1
Prestudy Report	1
Project Plan	1,2,3
Product Backlog	1,2,3
Test Specification	1,2,3
Sprint Reports	1,2,3
Document Standard	1,3
Quality Assurance	1,3
Analysis Document	2,3
Design Document	2,3
Technology Documents	1,2,3
Scrum Procedures Documentation	1,2,3
Meeting Notifications	1,2,3
Meeting Minutes	1,2,3
Budget	1,3
Contract	1
Time Reports	1,2,3
Code Standard	1
Weekly Status Reports	1,2,3
Post Project Report	3

TABLE 2.1 DOCUMENT OVERVIEW

3 PROJECT GOALS AND LIMITATIONS

The goals and limitations of this project can be divided into two categories: *Product goals and limitations* which are the main objectives concerning the product we are developing (the *scope* of the project), and *Educational goals and limitations* which are our expected learning outcome from the project when we are finished.

3.1 PRODUCT GOALS AND LIMITATIONS

We want to deliver a product we can be proud of to our customer, CORENA. If we achieve the following goals it is accomplished:

3.1.1 A STABLE PRODUCT

The product should be stable and not crash or encounter any other major failures under normal circumstances. There should also be allowance for common abnormal situations and errors in the input data being processed.

3.1.2 FULLY SUPPORT S1000D 4.0 STANDARD

We want to support all features in the S1000D 4.0 process Data Module specification. We want to do this in a modular way with little or no hard-coding of data and values to allow for support of other versions of the S1000D standard to be added later. The support for other versions will not be a part of this project.

3.1.3 A RELEASABLE PRODUCT

The *Code Coverage*, or *Test Coverage*, is a measure of to which degree the source code has been tested. To be a releasable product, CORENA requires a code coverage of at least 30-40%, and this is something we want to achieve for our product. The goal is to have a product that is ready for implementation into CORENA's existing products when we hand over our project to them.

3.1.4 ALL CORE FUNCTIONALITY IMPLEMENTED

In Scrum, the development methodology Ameo is using for this project, the Product Backlog will not show exactly what will be done during the project timeframe, since the Product Backlog is subject to several major and minor updates during the project's lifespan. In most cases it is also "never ending", meaning that it's not possible to complete a Product Backlog as new features and requirements are continuously added by the Product Owner as the market change and the customers' demands are developing. But since we are doing this project as part of our program at HiBu we have to limit the scope of the project. Therefore we are aiming on completing all the User Stories that are in the Product Backlog when this project plan was made (see [4], *Product Backlog*). The Product Owner can add and modify our

Product Backlog during the project, but he should discuss these changes with the project group first. It is important to notice that in a true Scrum project this would not be the case; the Product Owner changes the Product Backlog freely based on input from the customers, and only needs to discuss this at the Sprint planning meetings. He can discuss User Stories with team members to get a better estimate of the effort required.

3.1.5 THE PROJECT SHOULD BE WELL DOCUMENTED

The product and project in general should be well documented, with three focus areas:

- Allow the examiners to determine the progress and success of the project by studying the documentation.
- The product should be documented in a way so that it can easily be implemented by CORENA after delivery of the project.
- Add business value to CORENA; the documentation is part of the product, and by documenting everything we learn about the technologies, methods and tools we use that can be of interest for CORENA, this will add additional value to the company.

3.2 EDUCATIONAL GOALS AND LIMITATIONS

Maybe more important than the product we deliver is what we learn from this project, as we are doing this project in a learning environment with higher acceptance for partially failing with the product if we can learn and improve from our unfortunate experiences. Therefore we also want these educational goals to be achieved if we are to say that the project was successful:

3.2.1 SCRUM METHODOLOGY

We have decided on using Scrum as our development methodology, and we want to learn how to use it, and to get familiar with the strengths and weaknesses of this methodology. We believe we have achieved this goal if we succeed with our project using Scrum to the extent possible. We also want to evaluate how Scrum is works as the methodology used in a HiBu student project.

3.2.2 DEVELOPING LARGER PROJECTS IN TEAMS

All our work at HiBu so far has been smaller projects and teams than the scale of this project. With this project we want to learn how to tackle a larger assignment and use the bigger team to our advantage.

3.2.3 WORKING ON A PROJECT FOR A CUSTOMER

This is the first time we are developing a project for a customer, with all the new challenges and considerations this implies. We want to learn good and practical procedures for communicating with customers so they get what they want instead of what we think they want, and we want our customer to spend a minimum amount of resources on this communication.

4 PRECONDITIONS FOR THIS PROJECT

4.1 KNOWLEDGE AND SKILLS REQUIRED

To start this project we need to familiarise ourselves with the technologies it will require, and we need to understand the assignment. The project requires some technologies which we haven't used before, like Spring Framework and Google Web Toolkit. Since our product will be a stand-alone web based application, it will be an advantage that we have experience with web development in some way or another in the past. We also need knowledge acquired from our study at HiBu so far, including, but not limited to, programming, software analysis and design, project control and planning and networks.

4.2 SOFTWARE

We need to make sure we have access to all the software we are going to use, including the IDE, text editing software, software for source version control, XML editing tools, time tracking tools, a spreadsheet program and so on. We also need a server up and running for the source version control, our website and to run our application on.

4.3 DOCUMENTATION REQUIRED BY DEVELOPMENT START

We will begin development in January after our first presentation. Before this, we must have a Product Backlog, which will be our requirements since we are working with Scrum. We need to have developed test cases, we need technology documents and we need this project plan document. For our Product Backlog to be finished we need to have developed all of the high-level user stories, this will be done in cooperation with our employer CORENA. A vision document which functions as an application for our project is already written and approved by both HiBu and CORENA. See Table 2.1 for a complete overview of all the documentation to be delivered.

4.4 FACILITIES

We need a project room where we can meet to work on the project, have our Scrum meetings, internal mentor meetings and a place for us to have Scrum utilities like the task board.

4.5 EXTERNAL RESOURCES

We need support from, and rapid contact with, our assignee CORENA for this project to succeed. Tommy Sivertsen from CORENA is the Product Owner and our external mentor, and we need to cooperate closely with him to prioritise the Product Backlog (see [4]), and to get the technical and domain expertise we need to be able to develop this product successfully.

5 PROJECT ORGANISATION

5.1 THE PROJECT GROUP

The project group consists of four members, all studying computer engineering at HiBu. Two of the group members study embedded systems, while the other two study simulation and game development. The name of the project group, Ameo, is the first initial from each group member.

Name	Initials	Mail	Phone
Arild Oldervoll	AOL	arild@oldervoll.com	414 52 960
Marius Haraldseth	MHA	marius@haraldseth.net	415 20 610
Eirik André Eidså	EAE	eirik@eidsa.no	458 59 244
Olav Brandt	OBR	olav@brafa.net	984 48 717

5.1.1 DESCRIPTION OF GROUP MEMBERS

Arild Oldervoll (AOL)
Arild is the project leader. He's from Os, not far from Bergen. Arild has an education as a pilot from the USA, and in his spare time he tries to fly as much as possible, when he can afford it. He also uses a lot of his time on the Red Cross, where he is a member. He is in the Search and Rescue (SAR) team. Arild also does workouts in the gym. In his past he has played instruments for about 11 years, mostly the tuba, and he has run a lot of orienteering, he has actually won the Norwegian Championship in his class.

Other than being the project leader, Arild has responsibility for the code we are going to develop during this project.

Marius Haraldseth (MHA)
Marius comes from a farm located in Nes in Hallingdal. He now lives in Drammen with his girlfriend where they moved two and a half years ago when he started his bachelor degree in computer engineering. Marius and his girlfriend have just bought their own apartment in Drammen, and are moving there this December 2010. In his spare time Marius plays a little guitar, tries to learn some video and photo editing, does training work out and sometimes he takes a ride to the track with his Subaru Impreza. Marius has a sparkle for nice cars. In his past he has done a lot of track and field, and he has been doing shooting, both shotgun and saloon rifle.

Marius did his compulsory military service in the army as a motor pool responsible at Terningmoen, Elverum.

Marius is responsible for documentation and analysis.

Eirik André Eidså (EAE)
Eirik is from a farm in Eidså, almost as far west as you can get in Norway. He moved to Kongsberg when he started his bachelor degree in computer engineering. Eirik has a background in media and communication. In his spare time he plays the piano, cello and his Nikon camera is almost never more than an arm's length away on mountain trips all year around.

The areas of responsibility for Eirik during this project are web page and design.

Olav Brandt (OBR)
Olav is the only one in this group who actually is from Kongsberg. He has already finished a degree in political science, and is now soon done with the bachelor degree in computer engineering like the rest of us. In his spare time Olav likes to read books, listen to music, watch movies and play the computer game Super Meat Boy.

Olav's areas of responsibilities are economy and test.

5.2 EXTERNAL RESOURCES AND CONTACTS

This project has four resources and contacts external to the project. All communication from the project group to any of these resources and contacts should be initiated by the project leader. Since the project group is using Scrum, any communication with the team from any of these contact and resources should be directed to the project leader/ScrumMaster to avoid the introduction of impediments directly on the team:

Name	Role	Mail	Phone
Olaf Hallan Graven	Internal Sensor	olaf.hallan.graven@hibu.no	473 21 012
Øivind Ottersen	External Sensor	Oivind.Ottersen@corena.com	32 71 72 26
Tommy Sivertsen	External Mentor	Tommy.Sivertsen@corena.com	32 71 72 33
Karoline Moholth	Internal Mentor	karoline@moholth.com	920 81 428

6 ROLES AND RESPONSIBILITIES

6.1 INTRODUCTION TO ROLES AND RESPONSIBILITIES

When working in Scrum, we are trying to avoid dividing the work into roles, as the entire team is committed to completing all tasks and all team members can work on any task. Instead, we are dividing the work into responsibilities. That means that there is one person in the team that oversees that a general task is completed in time and according to our defined standards. Each team member will have at least one administrative responsibility and one technical responsibility.

6.2 ADMINISTRATIVE RESPONSIBILITIES

6.2.1 PROJECT LEADER

The project leader is the contact person for all contacts not part of the team. The project leader will also have the overall responsibility for delivering a product meeting all the requirements within the given timeframe. The project leader will also be the ScrumMaster, ensuring that the Scrum process is followed correctly.
Responsible: Arild Oldervoll

6.2.2 DOCUMENTATION

The person responsible for the documentation has the overall responsibility for making sure that all documentation being delivered follows all standards, that all documentation is consistent and that the documentation is delivered on time, to all eligible receivers and in the format requested by the receiver.
Responsible: Marius Haraldseth

6.2.3 WEB

The web responsible, or *web master,* must ensure that a website for the project group is developed and to keep it up to date with relevant information.
Responsible: Eirik André Eidså

6.2.4 ECONOMY

The person responsible for economy has to make sure that a budget is created and to monitor our expenses. The economy responsible must ensure that the project follows the budget and is within the economical frames set.
Responsible: Olav Brandt

6.3 TECHNICAL RESPONSIBILITIES

6.3.1 ANALYSIS

The analysis responsible is responsible for that all use cases and classes are analysed, that a functional architecture is developed, and that an analysis document is created and maintained.
Responsible: Marius Haraldseth

6.3.2 DESIGN

The design responsible must ensure that the system is described through classes, objects, state diagrams and activity diagrams, that a GUI is designed, as well as ensuring that a design document is created and maintained.
Responsible: Eirik André Eidså

6.3.3 CODE

The code responsible must ensure that a code standard is developed and adhered to. He will also have a general overview of all technologies and be able to help the team members using them.
Responsible: Arild Oldervoll

6.3.4 TEST

The test responsible has to make sure that a test strategy and specifications are developed and adhered to, and that all tests are performed as specified.
Responsible: Olav Brandt

7 PROJECT CONTROL

7.1 SPRINTS

In Scrum, the iterations are called *Sprint*s, and by standard a Sprint has a duration of 4 weeks, but anything from 1-4 weeks is considered normal. In our project, we have been using Sprints of 4 weeks for the pre-study phase, but after the first presentation, when development starts, we will reduce the sprint duration to three weeks. In Scrum, the Sprints are time-boxed, meaning that if there are remaining tasks in a Sprint, the Sprint will not be extended, but the tasks not done will be dropped and usually moved to the next Sprint. Any features not completed during the Sprint will not be demonstrated during the Sprint review.

7.2 SPRINT OVERVIEW

The amount of hours available in each Sprint is possible to estimate, but at this time it is not possible to estimate accurately what will be done in each Sprint (see the document *Scrum Planning And Product Backlog [1]).*

7.2.1 UNEXPECTED EVENTS

In Scrum we do not take into account the possibility of unexpected problems and delays to occur, and there is no risk analysis of such events before the project starts. Instead of planning on what to do *if* something unexpected would happen, the plan is adjusted only *when* something happens.

In our project, such an unexpected event has occurred. One of our team members has become seriously ill and will spend a lot of time in the hospital and receiving treatment. Scrum handles such unexpected events pretty well. We have not lost any time on creating plans that we now have to change. When we lose most of the velocity produced by a team member, it will only affect the number of tasks we are able to complete in the Sprint we are in when it happens, and it will of course affect the number of User Stories we will be able to complete within a given timeframe. Further on, since we don't know how much he will be able to participate in the project, we have decided to assign him a User Story with a lower priority that is not scheduled for the current Sprint. This is so that he can do as much as he is capable of, without us being dependent on a certain velocity from him.

7.2.2 UPDATED TIMETABLE

Due to this event, we've updated the estimated hours available each Sprint, and our Sprint goals for these Sprints. When estimating hours we have for now estimated 0 hours for our sick team member and a slight increase in hours from the other team members to compensate for some of this. If the ill team member is able to do some work we will only count it as a bonus which will lead to more of the Product Backlog being completed. Table

7.1: Sprint overview gives an overview of the remaining Sprints, the expected hours available and expected Sprint goals for those Sprints.

Sprint #	Start Date	End Date	Estimated hours available	Estimated Sprint Goal
6	18-Jan-2011	08-Feb-2011	220	Analyse the S1000D structure and elements
7	09-Feb-2011	01-Mar-2011	160	Display the data modules in the application
8	02-Mar-2011	23-Mar-2011	180	Define architecture
9	24-Mar-2011	13-Apr-2011	240	Create most important features of the logic engine
10	15-Apr-2011	11-May-2011	230	Create remaining logic engine features
11	11-May-2011	31-May-2011	360	Implement support for external applications
12	01-Jun-2011	09-Jun-2011	200	Final Reports and Presentation

TABLE 7.1: SPRINT OVERVIEW

The development is expected to be finished in the middle of Sprint 11 (around May 17th). The original estimate of 1760 hours of development is downsized to **a total of 1390 hours**. A total of **489 hours** were used during the 5 first Sprints, so the total for the project is expected to be **2079 hours.** This equals **520 hours per group member (with four group members), but since the estimate is based on three persons from middle of Sprint 6, the actual estimate for these three group members is 650 hours.**

7.3 IMPORTANT DATES

During the project there will be four important dates:

- January 12th: The first presentation.
- March 11th: Second presentation sometime during this Sprint.
- June 1st: Delivery of the product.
    - The delivery date is postponed from May 30th because of Eirik's cancer treatment. We will probably not see Eirik again until the presentation day, and therefore the delivery was postponed so we could work with him on the presentation before he had a new round of chemo therapy.

- June 9th: Third and final presentation.

7.4 ACTIVITIES

For project tracking and control, all work done in the project is categorised in the high-level activities listed in, *High-Level Activity Overview* [6]. For a more detailed time tracking, the Sprint Backlog is used to track time on task-level.

7.5 WEEKLY REPORTS

Weekly reports are made for each week in the project. They are sent weekly to our internal mentor, and all the weekly reports will be gathered in the documentation delivered for the presentations. These reports will give an overview of what the group and each group member has been working on for the previous week, what is planned to be done for the next week and the general status of the project.

7.6 OTHER CONTROL DOCUMENTS

7.6.1 DOCUMENT STANDARD

The document standard contains all the rules and practices for Ameo's documentation, such as fonts, styles, use of paragraphs, use of colours, naming of files and more. The document responsible must ensure that all documents that are produced by Ameo follow this standard.

7.6.2 BUDGET

A budget of our expected expenses are documented in, *Budget* [7]. These expenses are to be covered by CORENA, and therefore it is important that we keep track of our expenses and make sure that we stay on budget.

7.6.3 CODE STANDARD DOCUMENT

The Code Standard sets the standard for code writing in Ameo, regarding such things as formatting, naming of variables and functions and so on. The code responsible must ensure that all the code produced by Ameo follow this standard.

7.6.4 TEST REPORTS

The test reports will document that the tasks are completed, but it is also where the tasks are described in detail. The Sprint Backlog only contains a short description/title of the task.

8 REFERENCE(S)

1. *Scrum Planning and Product Backlog*, Ameo, 2010 HiBu.

2. *Prosjektplan*, Bjørnar Lintvedt, Thomas Fiskum Bembridge, Kjetil Skaret, Mads Haugsmoen, Tahreem Butt & Robert Grande (Scrat), 2010 HiBu.

3. *Prosjektplan*, Lone Knutsen, Arne Kristian Åmellem, Zoran Vukobratovic, Ole-Marting Grøtterud, Thomas Transeth (KDA Rivet), 2010 HiBu.

4. *Product Backlog,* Ameo, 2010 HiBu

5. *Document Overview,* Ameo, 2010 HiBu.

6. *High level Activity Overview*, Ameo, 2010 HiBu.

7. *Budget*, Ameo, 2010 HiBu.

8. *S1000D Issue 4.0,* S1000D, http://public.s1000d.org/Downloads/Documents/Issue4.0/s1000d_Issue_4_0_final.zip (last visited 05-Jan-2011).

ANALYSIS DOCUMENT

HIBU STUDENT PROJECT 2011
CORENA S1000D 4.0 PROCESS DATA MODULE RENDERER

____
Arild Oldervoll

____
Marius Haraldseth

____
Eirik André Eidså

____
Olav Brandt

TABLE OF CONTENTS

REVISION HISTORY			
Version Number	**Date**	**Changes**	**Assigned**
1	25-Mar-2011	First Official Version	OBR
2	01-Jun-2011	Updated chapter on Dialogue Handling Added chapter for State Table Added package description Added chapter on Display Package	AOL

	Update of chapter for State Table Updated chapter on Display Package Added subchapter for saving and loading sessions Added subchapter 4.5 XML Handling Added subchapter on Expressions

1 INTRODUCTION

Our assignment to develop a logic engine for support execution of S1000D process Data Modules version 4.0, is given by CORENA Norge A/S. This logic engine can take input from users and/or external applications, such as sensors, to decide its next step in a flow of process Data Modules. It can branch and loop through the selected process Data Module. The logic engine will be an interpreter for process Data Modules. We will make it a stand-alone logic engine in a web based application running on Jetty/Tomcat. The user of the application should be able to move back and forth through the steps using previous and next buttons. The user should be able to save the session to continue at a later time. The logic engine should support the International S1000D Specification Issue 4.0.

1.1 INTENTION OF THIS DOCUMENT

The analysis document will contain documentation describing the background for the decisions we make regarding the analysis, and descriptions of the decisions themselves.

1.2 SYNONYMS, ACRONYMS AND DEFINITIONS

Synonym	Description
Process	Set of instructions and conditions being executed
Data Module	Container of data/information.
process DM	Process Data Module
GWT	Google Web Toolkit
RPC	Remote Procedure Call
MVP	Model-view-presenter
MVC	Model-view-controller

1.3 AUTHOR(S)

Olav Brandt (OBR)

Marius Haraldseth (MHA)

Arild Oldervoll (AOL)

1.4 ASSIGNED

Olav Brandt (OBR)

2 S1000D SPECIFICATION

Our program will be developed according to the S1000D specification. This sets a number of requirements in terms of the features, functionality and structure of our program.

The S1000D specification requires that our application contains a logic engine, which utilises a state table. The primary job of the logic engine is to determine what to display next. It interprets and executes process Data Modules. The state table is a collection of variables and their values. The logic engine must be able to populate, store, retrieve, update and delete state information in the state table. The logic engine must use values from this table in branches, loops and to evaluate expressions used in applicability.

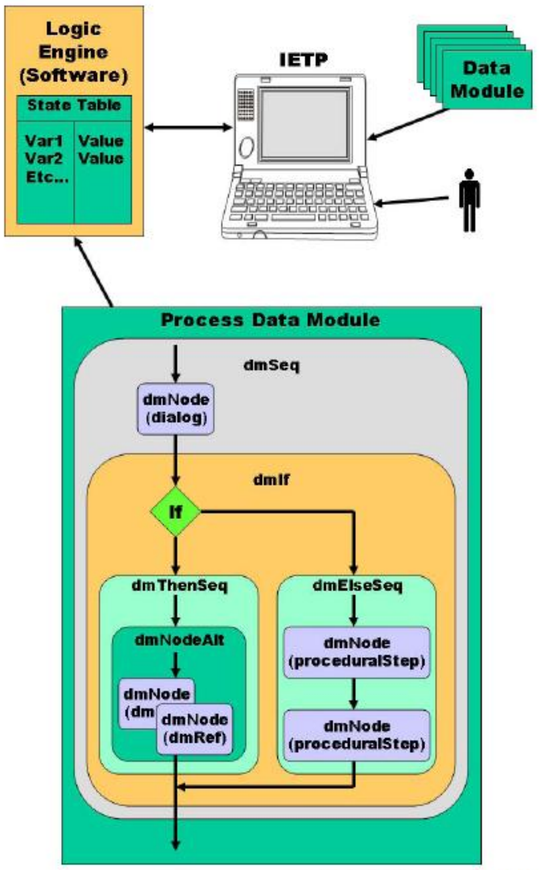


FIGURE 2.2.1 PROCESS DATA MODULE CONCEPTUAL DIAGRAM

The logic engine should be able traverse process Data Module elements, maintain state variables, present dialogues and evaluate expressions. It must traverse the process Data Modules sequentially. It should also be able to communicate with external applications through an interface. Our application will also need to communicate with the CORENA IETP.

The specification also sets requirements for the features of the application’s graphical user interface. Additional specification guidelines for the GUI’s presentation and looks, together with the design of the existing CORENA IETP, are another set of signals we must take into account.

Our program need some form of version management to recognise different versions of the S1000D process DMs, and what to do with them. Different versions may include different elements, require other ways of handling certain elements or actions, or have a different structure.

The logic engine needs to have error detection, record detailed information about the error so it is possible to pinpoint the problem.

We’ve produced a separate set of S1000D documentation with additional analysis and mapping of a process DM’s structure and elements. They can be found under the folder *S1000D pdM*.

3 GWT AND CLIENT/SERVER

We are developing a program that is accessed through a browser; a web application. It is developed using Google Web Toolkit, a set of tools and technologies where the core functionality is the ability to write Java code that is automatically translated to JavaScript to be run in a browser.

The nature of a web application is the two-part structure, consisting of a client and a server. The client is the part that is accessed through a browser by the user of the program and is typically made up by, primarily, a graphical user interface. The server is the backend of the program, and is where the domain/business logic resides.

Because the client and the server are, in essence, two separate entities, communication between them is not as simple as in a standard desktop application. Instead, it must take place through some intermediary protocol or message-handler of some sort.

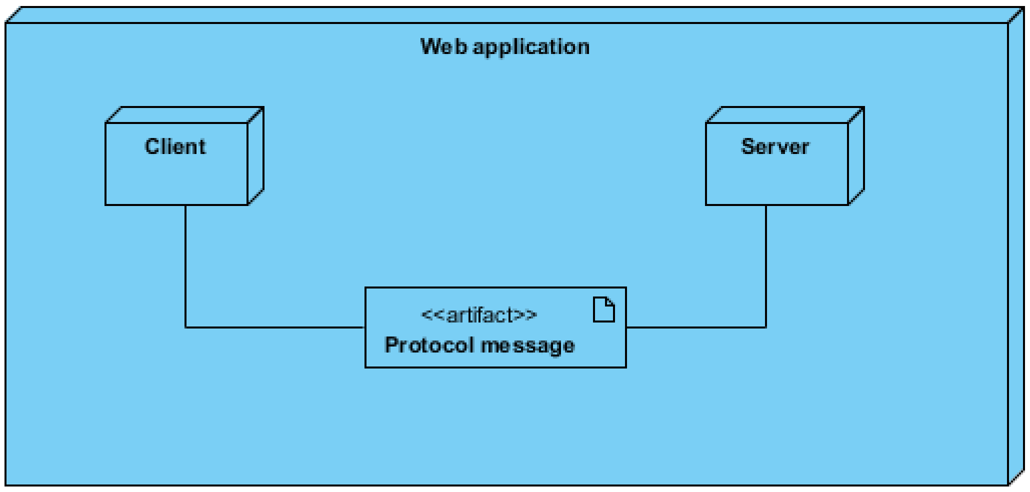


FIGURE 3.1: BASIC STRUCTURE OF A WEB APPLICATION

In our case, GWT provides both the client parts of the application and the means of communication between the client and the server. Our application's client and server will be sending and receiving data through so called RPCs, or Remote Procedure Calls.

The functionality on our server will be usable to our client via *services*. These are defined through a set of interfaces which are situated on the client and implemented by the server. The client can then make an asynchronous call to the server, requesting an operation to be performed and/or a piece of data to be returned.

As noted in figure 3.2, both the client and server are written with Java code, but while the client code is translated to JavaScript, the server code is run as standard Java. The client thus only supports code that is translatable, while the server has no such limitations.

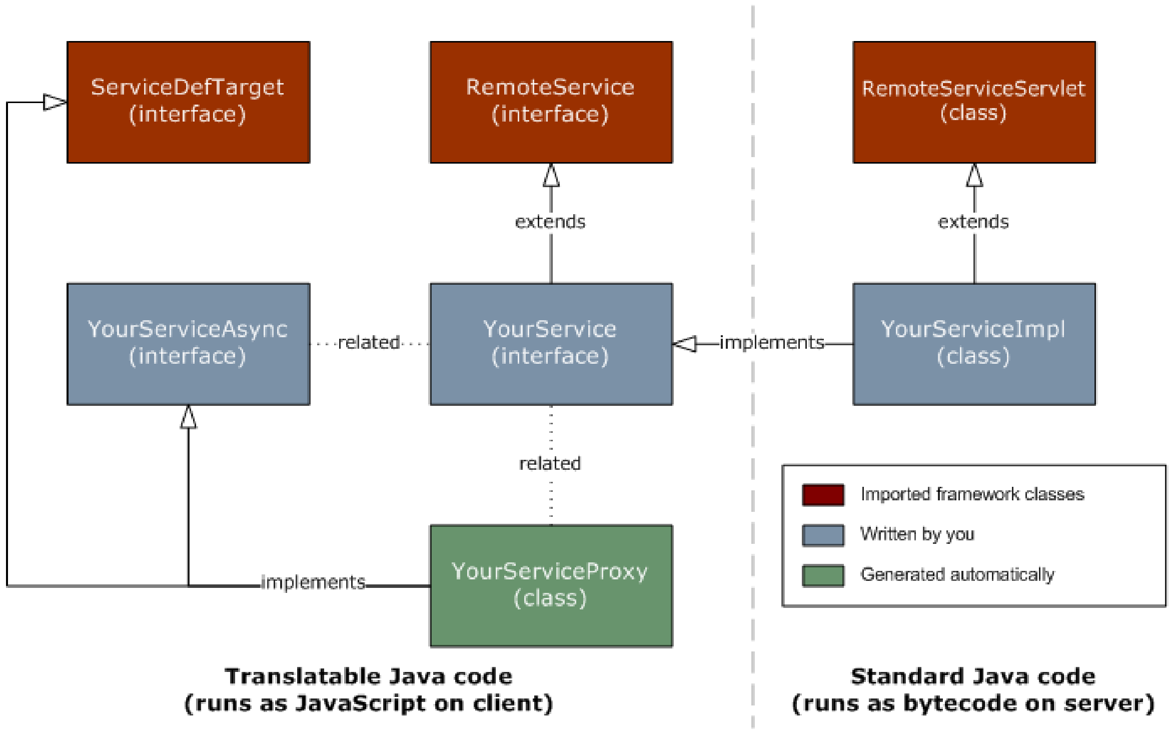

ServiceDefTarget (interface)
RemoteService (interface)
RemoteServiceServlet (class)
extends
extends
YourServiceAsync (interface)
related
YourService (interface)
implements
YourServiceImpl (class)
related
Imported framework classes
Written by you
Generated automatically
implements
YourServiceProxy (class)
Translatable Java code (runs as JavaScript on client)
Standard Java code (runs as bytecode on server)

FIGURE 3.2: GWT'S RPC FRAMEWORK, SHOWING CLIENT/SERVER AND THE SERVICES THAT CONNECT THEM

4 APPLICATION STRUCTURE

During analysis, we've attempted to identify the main components our program is made up by. These components will be broken down into sub-components through a more detailed design process.

Figure 4.1 shows the components we've identified in the first development-oriented Sprint.

- Internal: These components are contained within our actual application.
    - Logic Engine: The functionality related to the logic engine as specified by the S1000D specifications.
    - State Table: Representation of the state of a process DM.
    - Renderer: The component that decides how information is to be displayed in the GUI.
    - GUI: Used to interact with the user.
    - XML Handler: Process Data Modules are represented through XML files. This component handles these files.
    - Communication: For communicating with the CORENA IETP, and possibly external applications.
- External
    - IETP: Interactive Electronic Technical Publication. The IETP our application is interacting with is developed by CORENA. It is used by our application to fetch process DM files and possibly for certain rendering responsibilities.
    - S1000D Version Handler: Some external format, protocol or method used by our program to handle different process DM versions.

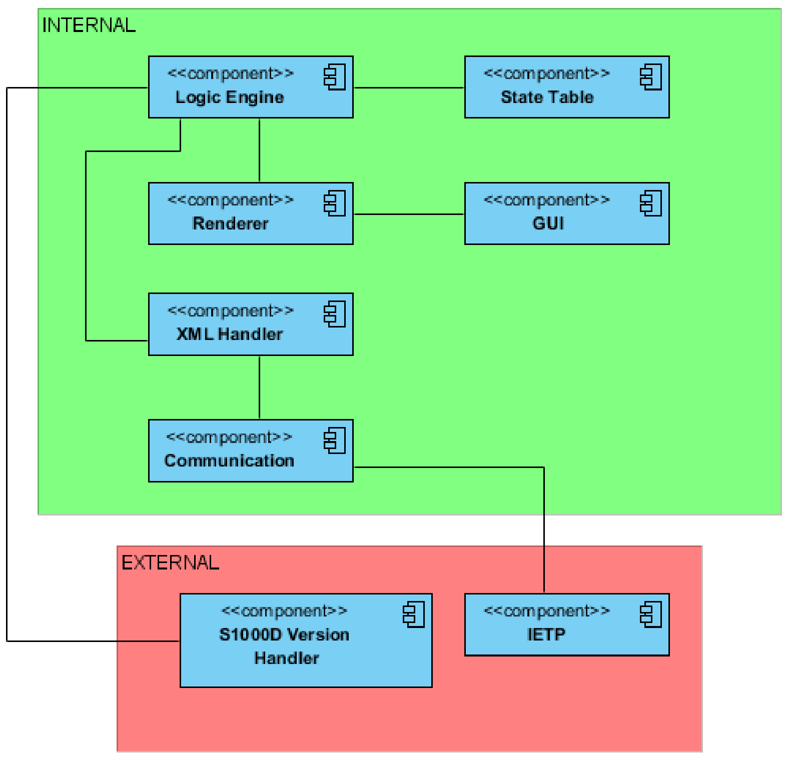

INTERNAL
<<component>> Logic Engine
<<component>> State Table
<<component>> Renderer
<<component>> GUI
<<component>> XML Handler
<<component>> Communication
EXTERNAL
<<component>> S1000D Version Handler
<<component>> IETP

FIGURE 4.1: MAIN COMPONENTS

4.1 SEPARATION OF CONCERNS, MVP

Separation of concerns is a core principle in software development. Separating our program into different parts, each with a distinct role and corresponding responsibilities, allows us to more easily change, refactor or replace software code without affecting the rest of the application.

Separation of concerns is applicable throughout the entire software development process. For the analysis part, we've made a decision on a software architecture that divides a program into three main parts; Model-view-presenter. According to [4], MVP is a "derivative of the model-view-controller software pattern", an architectural pattern that separates the domain/business (*model*), the user interface presentation (*view*) and the input handling (*controller*).

MVP shares the same idea and roles, but some responsibilities are moved around compared to MVC. Most notable, the task of updating the view is moved from the model to the controller, now called a *presenter*.

Figure 4.2 shows a graphical representation of the difference in associations between the software parts of MVC and MVP.

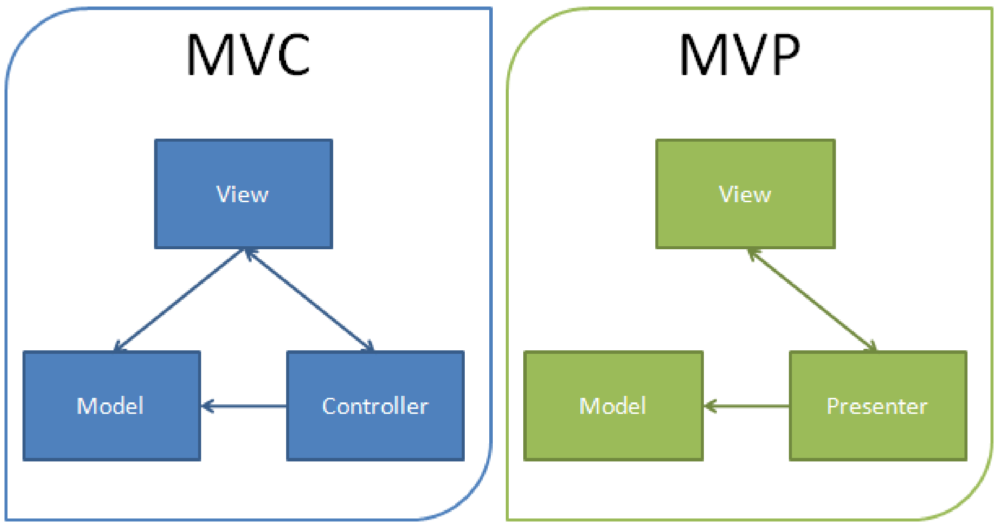


FIGURE 4.2: MVC AND MVP

4.2 PACKAGES

We have identified the main packages that the application will be divided into. Each package has one main task, usually representing a component of the SLRP. All the packages we create belong to the *com.corena.ietp.slrp* domain. The packages are divided so that the content of one package should have minimal effect on any other packages, and for easier testing.

FigurE 4.1 provides an overview of the packages

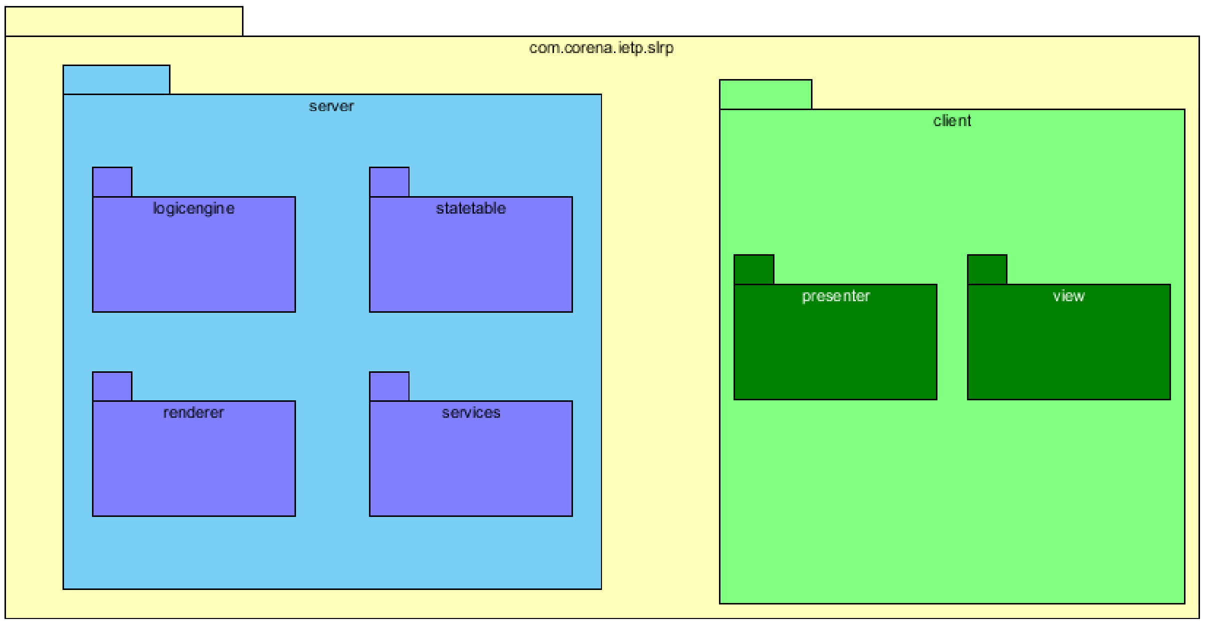


FIGURE 4.3 PACKAGE OVERVIEW

4.3 HIGH LEVEL ARCHITECTURE (HLA)

4.3.1 RENDERING OF VISUAL CONTENT

In accordance with the S1000D specification, the logic engine renders a *display package* for the screen to be displayed. The display software, in this case the client, will handle the display package and display the correct elements on screen. The specification also says that until a screen is processed and any expressions are evaluating to true, the variables for that dmNode should be stored in a separate State Table. This lead to the general HLA shown in FIGURE 4.4 High Level Architecture:

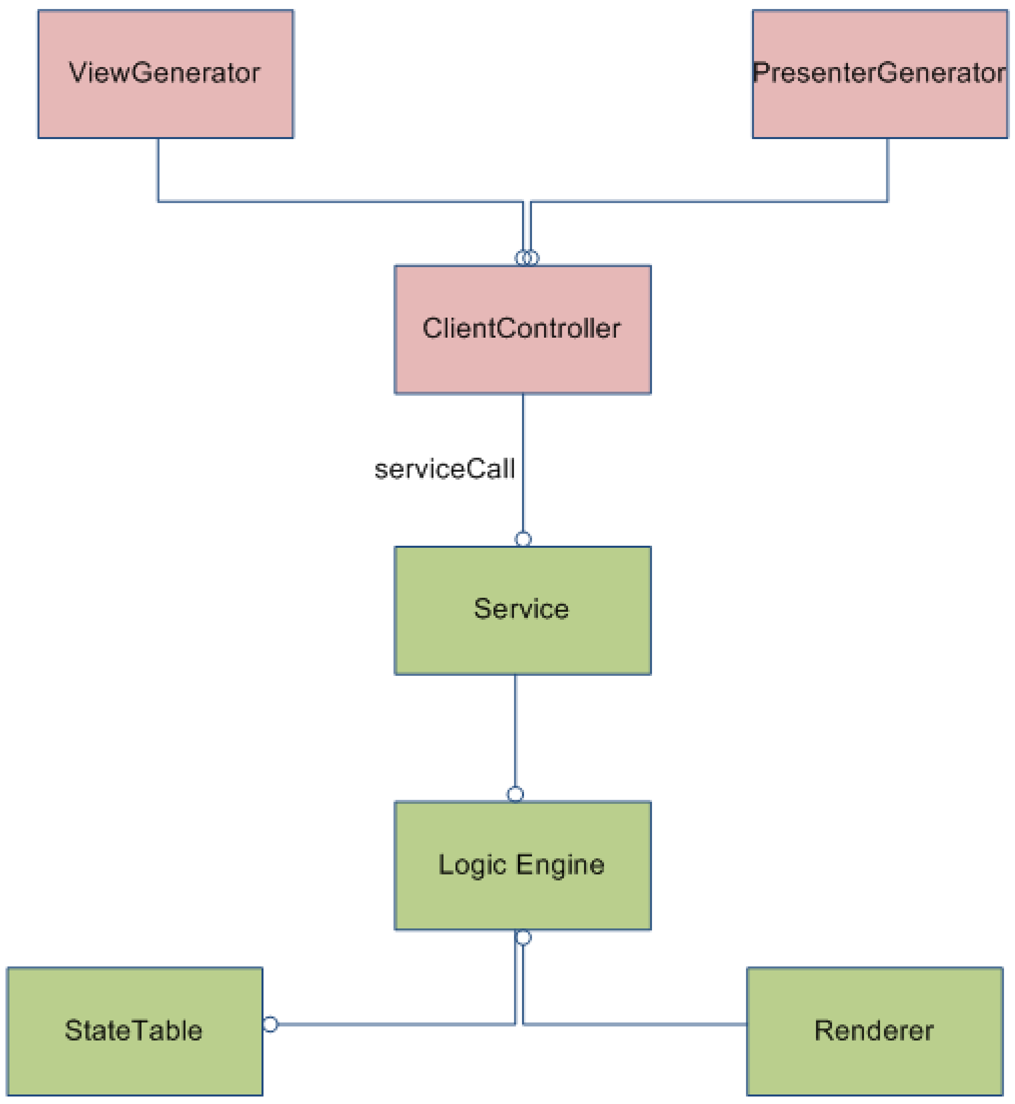


FIGURE 4.4 HIGH LEVEL ARCHITECTURE

The red elements are part of the client, and the green elements are part of the server/backend. The ClientController is an addition to the MVP, and is the one that requests the next Display Package from the server when ready. The logic engine will then process the variables in the temporary State Table and evaluate the expressions to calculate a new

Display Package in the renderer, and then sending this back to the ClientController through the Service Callback. In order to keep it dynamic, the ViewGenerator will then generate the view, and the PresenterGenerator will register listeners for the widgets generated on the ViewGenerator. This will ensure that it's flexible and dynamic, follows the S1000D specifications and that MVP is conserved.

4.4 FUNCTIONALITY

4.4.1 SHOW PDM

A core functionality of our application is to show process DMs in a GUI. The process DM file is located on the CORENA IETP, and needs to be transferred to our program before it is shown. The file is fetched as a result of the process DM being selected by a user in the GUI.

The call originates from the user clicking the Open button and selecting the process DM in a dialogue. The call then travels from the client to the server via a service. The server queries the IETP for the file, which is returned to the client and shown in a content panel.

FigurE 4.5: class diagram for "show pdm"5 shows the functionality's class diagram.

FigurE 4.6 shows the sequence diagram for the functionality.

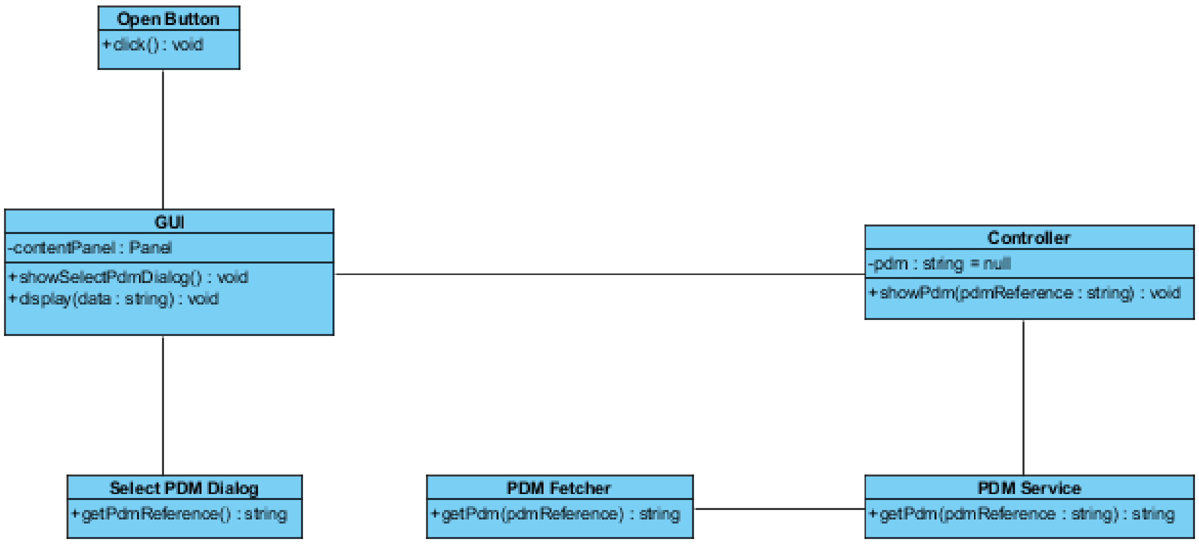

Open Button
+click() : void
GUI
-contentPanel : Panel
+showSelectPdmDialog() : void
+display(data : string) : void
Controller
-pdm : string = null
+showPdm(pdmReference : string) : void
Select PDM Dialog
+getPdmReference() : string
PDM Fetcher
+getPdm(pdmReference) : string
PDM Service
+getPdm(pdmReference : string) : string

FIGURE 4.5: CLASS DIAGRAM FOR "SHOW PDM"

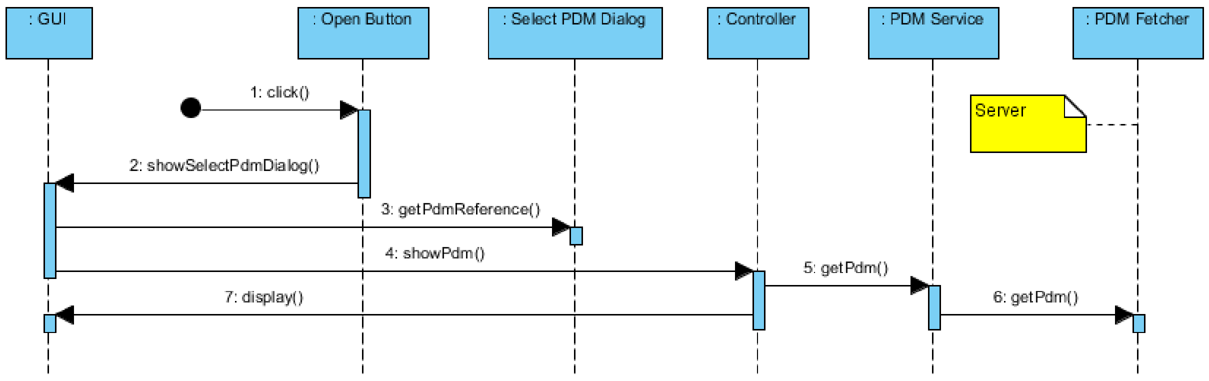


FIGURE 4.6: SEQUENCE DIAGRAM FOR "SHOW PDM"

4.4.2 SAVE AND LOAD SESSIONS

It should be possible to save and load sessions in the application. These features will be implemented in the state table (see chapter 4.7).

When saving the session, a value will be set, and the session info will not be deleted when closing the application. See figure 4.7 for a sequence diagram of the server part.

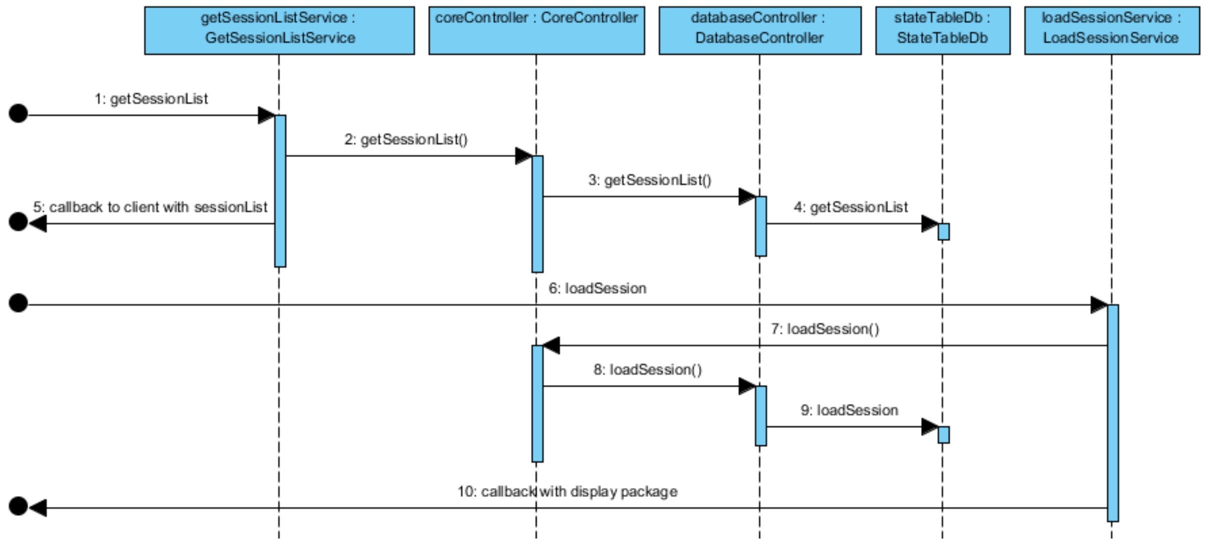

getSessionListService : GetSessionListService
coreController : CoreController
databaseController : DatabaseController
stateTableDb : StateTableDb
loadSessionService : LoadSessionService
1: getSessionList
2: getSessionList()
3: getSessionList()
4: getSessionList
5: callback to client with sessionList
6: loadSession
7: loadSession()
8: loadSession()
9: loadSession
10: callback with display package

FIGURE 4.7 SERVER - LOAD A SESSION

When loading a session, a session list will be loaded from the state table and shown to the user for selection. The user will select a session, and the application will load this session from the state table. See figure 4.8 for an activity diagram.

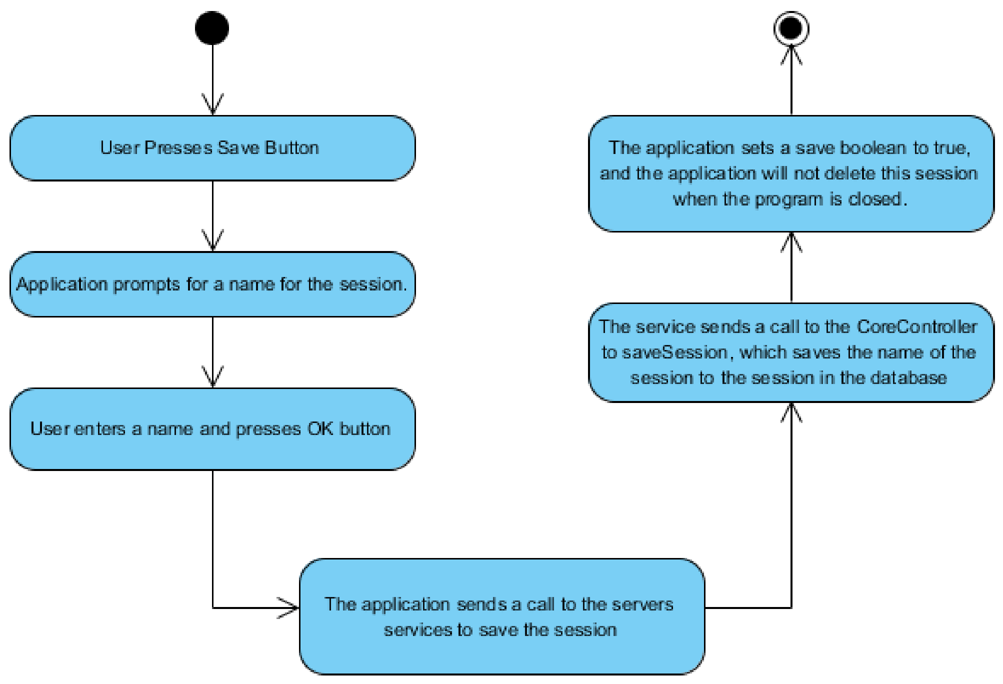

User Presses Save Button
Application prompts for a name for the session.
User enters a name and presses OK button
The application sends a call to the servers services to save the session
The service sends a call to the CoreController to saveSession, which saves the name of the session to the session in the database
The application sets a save boolean to true, and the application will not delete this session when the program is closed.

FIGURE 4.8 SAVE SESSION

4.4.3 EXPRESSIONS

Expressions are what truly separate process Data Modules from other Data Modules. They contain logical operations and functions that need to be evaluated by the Logic Engine according to the spec. Expressions are used for applicability, branching, validating user entry and more.

The functionality of a single <expression> is decided by the state of its child elements. An <expression> element may contain one of the following:

- Four types of “operator” (Boolean, number, set, string)
- Five types of “function” (Boolean, number, set string, substring)
- Six types of “value” (Boolean, string, real, integer, set, no value)
- A variable reference

The operators and functions are further represented by several dozen “operations” and “actions”, several of which have multiple parameter and return types.

When the Logic Engine encounters an <expression> element while traversing the process Data Module, it is sent to an Expression Evaluator for evaluation. The Expression Evaluator checks the type of data the expression contains, and performs the appropriate action. The result is returned to the Logic Engine, which then executes the next step based on the result.

Figures 4.9 and 4.10 show a class diagram and sequence diagram for evaluating expressions.

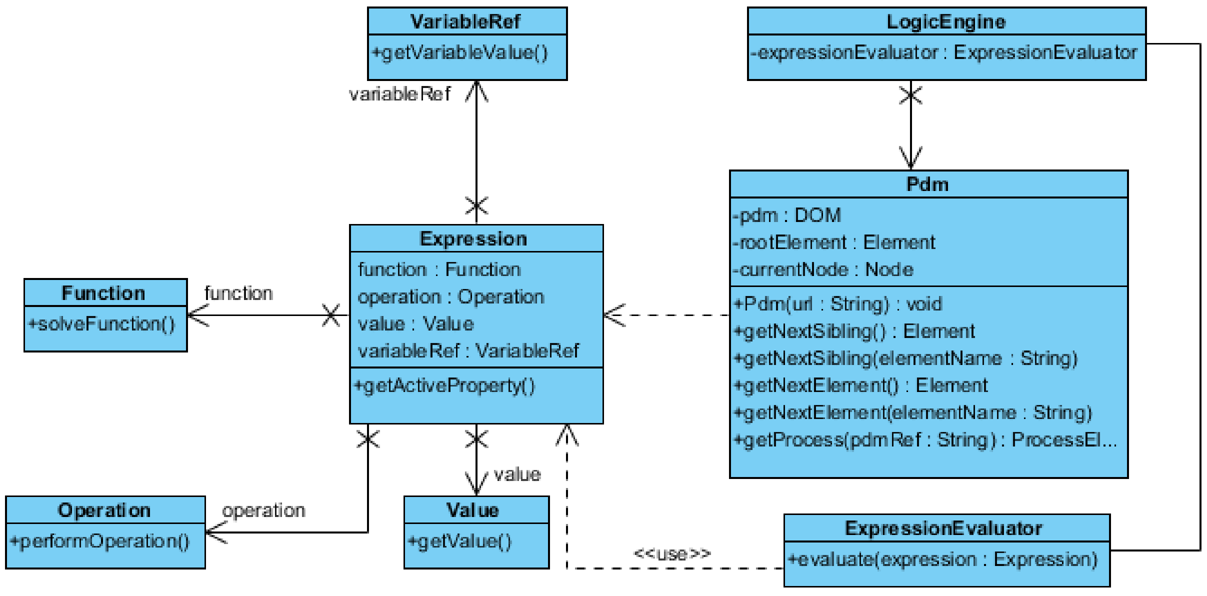


FIGURE 4.9 CLASS DIAGRAM FOR EVALUATING EXPRESSIONS

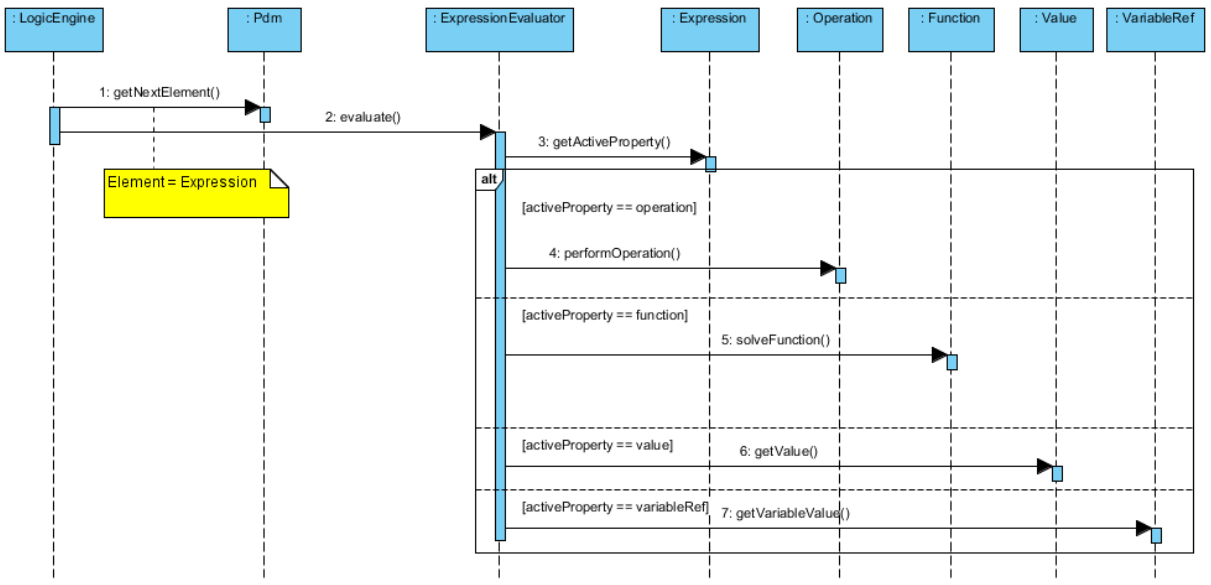


FIGURE 4.10 SEQUENCE DIAGRAM FOR EVALUATING EXPRESSIONS

4.5 XML HANDLING

The process DM is represented in an XML format when received from the IETP. To use the data in this file further in our application it must be parsed into a format that the application can handle in a simpler manner than reading the file from start to finish every time it needs new data from the process DM. The XML handler should separate the concern of the pDM format so that the rest of the application will have a unified representation of the data even though the S1000D specification might change with new versions.

4.6 DISPLAY PACKAGE

When encountering elements in the process DM that is displayable, the Logic Engine will use the Renderer to generate a *display package* to be sent to the client. The display package contains the relevant data found in the element - that is, only the data that will be displayed to the user. Other, server-side data is evaluated in the Logic Engine and is not something the client needs or should have access to.

There are two main categories of displayable data/elements; dialogues and collections of figures, tables and multimedia objects, either directly in a dmNode or as a number of procedural steps. Dialogues are standard pop-up dialogues whose content is decided “on the fly” by the Logic Engine, i.e. the content is completely dynamic. A procedural step can contain a number of displayable data, including text, figures and tables. A list of procedural steps can be followed by an optional embedded dialogue.

Figure 4.11 shows a class diagram for display package handling

Figure 4.12 shows a sequence diagram for display package handling

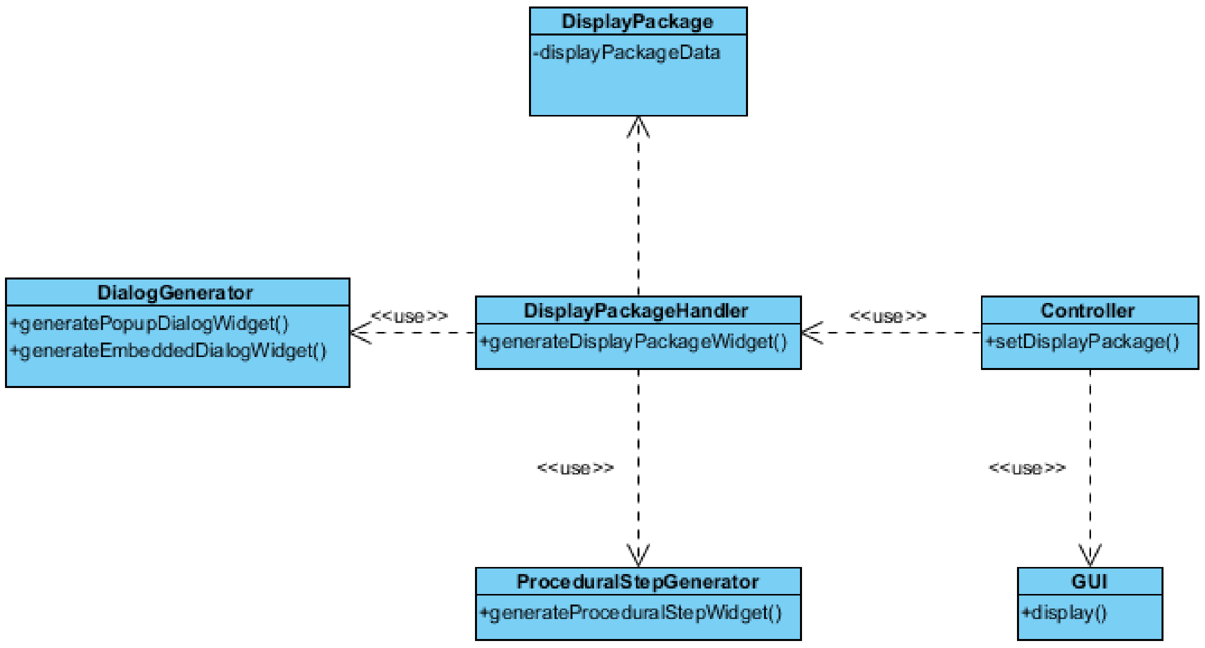
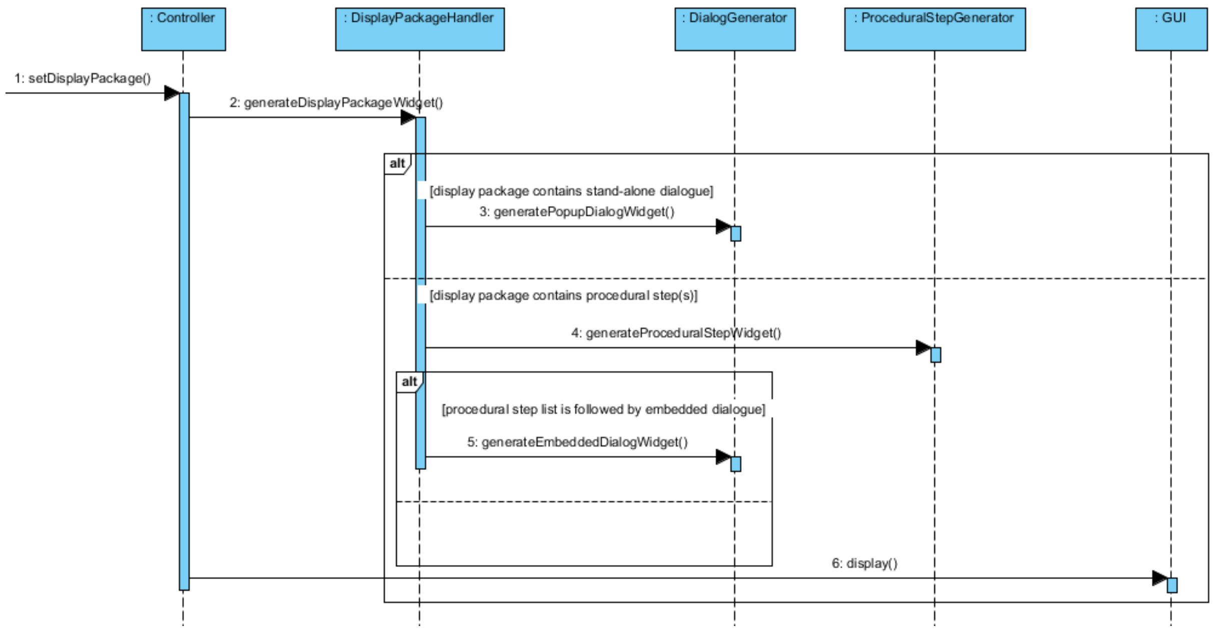


FIGURE 4.11 CLASS DIAGRAM FOR DISPLAY PACKAGE HANDLING



: Controller
: DisplayPackageHandler
: DialogGenerator
: ProceduralStepGenerator
: GUI
1: setDisplayPackage()
2: generateDisplayPackageWidget()
alt
[display package contains stand-alone dialogue]
3: generatePopupDialogWidget()
[display package contains procedural step(s)]
4: generateProceduralStepWidget()
alt
[procedural step list is followed by embedded dialogue]
5: generateEmbeddedDialogWidget()
6: display()

FIGURE 4.12 SEQUENCE DIAGRAM FOR DISPLAY PACKAGE HANDLING

4.7 DIALOGUE HANDLING

When the Logic Engine encounters a dialogue type element (<dialog>, <message>), it will 1) create a temporary state table for the dialogue, and 2) pass the contents of the element and its child elements to the Renderer. The Renderer will interpret the elements and create a display package that is sent to the client. When a user submits input from the dialogue, the event and the input data are sent via a service call to the server.

The server will fill the dialogue state table with the data and the Logic Engine will attempt to validate it. If the data validates, the Logic Engine moves on to the next step in the process DM. If not, an error message is generated, sent to the client and displayed.

Figure 4.13 shows an activity diagram for the dialogue handling.

Figure 4.14 shows a component diagram for the dialogue handling.

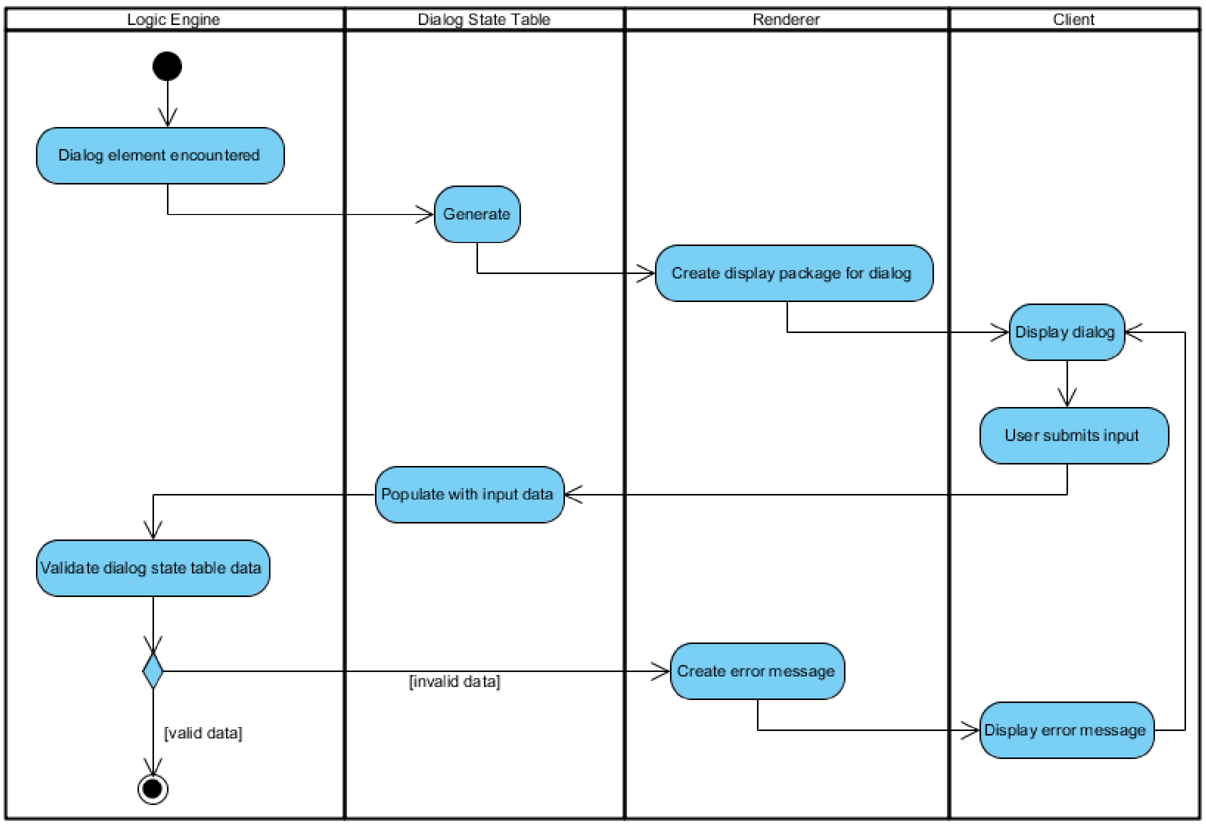


FIGURE 4.13 ACTIVITY DIAGRAM FOR DIALOGUE HANDLING

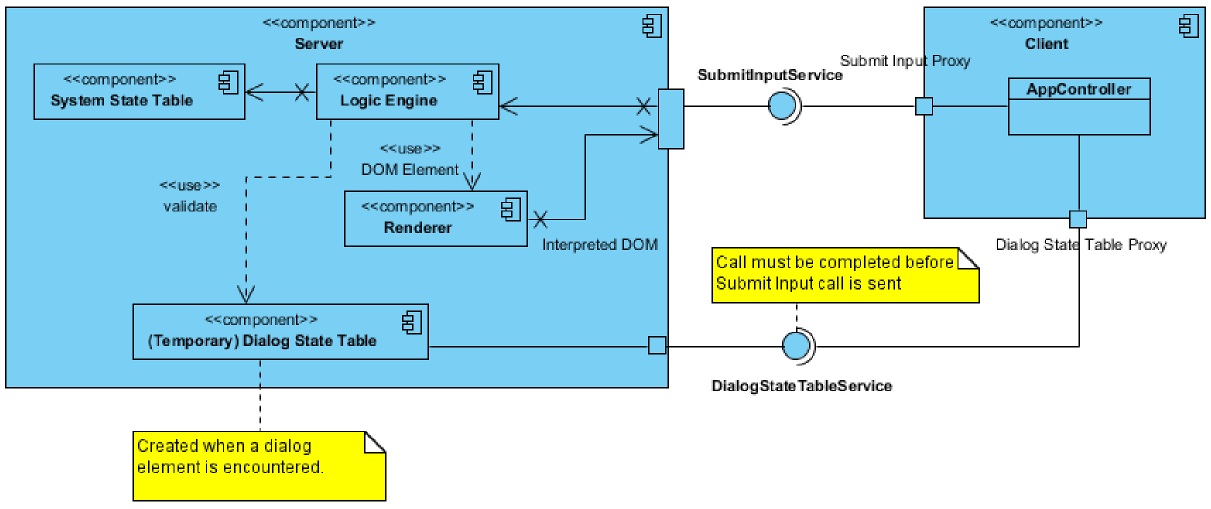


FIGURE 4.14 COMPONENT DIAGRAM FOR DIALOGUE HANDLING

4.8 STATE TABLE

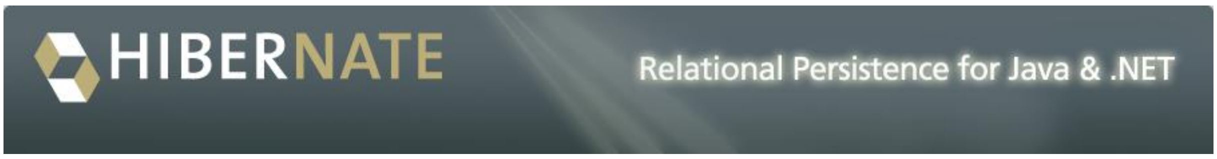
We need a state table for this application. It is a requirement that this should be persistent. Persistency means that it should be stored outside of the application, so if the program should crash for example, the state table will still be extant.

To fulfil this requirement we will use a database for the state table.

In this state table it should be possibilities to:

- Store a new session.
- Store a new variable.
    - Temporary and not temporary
    - A variable can have several values
- Update variables.
- Store a new state.
- Retrieve sessions, variables and states.
- Delete state table sessions and states.

For communication with our database we will use the object-relational mapping tool Hibernate. Hibernate is a free open source tool. This tool makes it possible for us to work with objects and not tables.



We will refer to two types of session.

- *State table session*, this will always be referred to as the state table session. It is a run/session in the application
- Hibernate *Session* object; this will be referred to as Session object. This is an object which the Java objects gets an association with when they are persisted (see chapter 4.7.1). This object is created every time we have an interaction with the database.

4.8.1 HIBERNATE OBJECT LIFE CYCLE

The Hibernate object has a lifecycle as shown in figure 4.15 below:

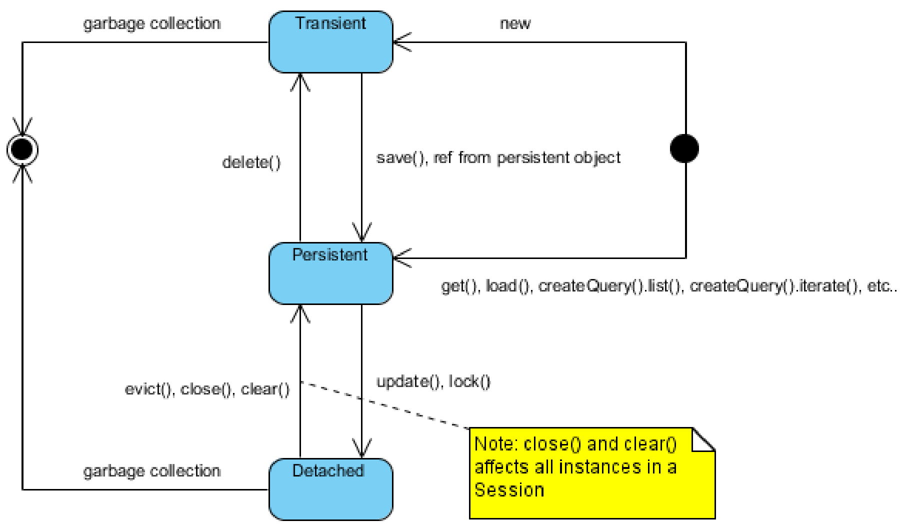

Transient
garbage collection
new
delete()
save(), ref from persistent object
Persistent
get(), load(), createQuery().list(), createQuery().iterate(), etc..
evict(), close(), clear()
update(), lock()
Note: close() and clear() affects all instances in a Session
garbage collection
Detached

FIGURE 4.15 THE HIBERNATE OBJECT LIFECYCLE

Transient objects are not yet saved to a row in the database. They have no association with the database, they act as normal Java objects. There is no connection between the transaction and these objects. These objects can be turned into persistent objects with the *save* method call of the Session object. This is the Hibernate *Session*. It can also be persisted by adding a reference from another persistent object to this object.

Persistent objects have an association with the database and are always associated with a persistence manager, i.e. a *Session* object. The object does not have to be saved to the database yet, but it will have a primary key value set anyway. When the *delete* method of the *Session* is called on a persistent object, the object will be removed from the database, but it will still be available as a regular Java object.

Detached objects are objects that were persistent, but no longer have a connection to a *Session* object. The object is now stale. This object can be persisted again.

4.8.2 APPLICATION'S STATE TABLE STRUCTURE

The class diagram for the state table is shown in figure 4.16 below:

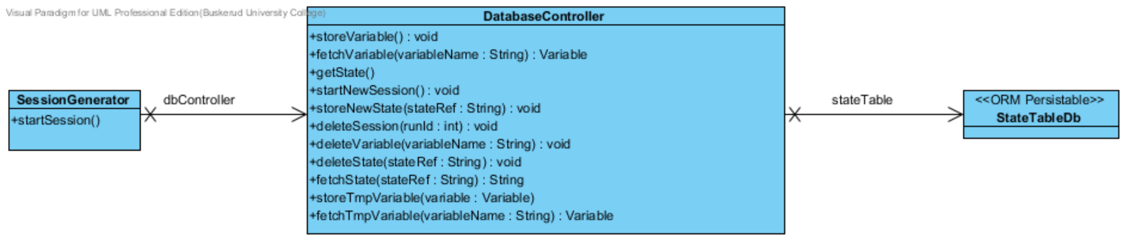

Visual Paradigm for UML Professional Edition(Buskerud University College)
SessionGenerator
+startSession()
dbController
DatabaseController
+storeVariable() : void
+fetchVariable(variableName : String) : Variable
+getState()
+startNewSession() : void
+storeNewState(stateRef : String) : void
+deleteSession(runId : int) : void
+deleteVariable(variableName : String) : void
+deleteState(stateRef : String) : void
+fetchState(stateRef : String) : String
+storeTmpVariable(variable : Variable)
+fetchTmpVariable(variableName : String) : Variable
stateTable
<<ORM Persistable>>
StateTableDb

FIGURE 4.16 ANALYSIS OF STATE TABLE CONTROL AND CONNECTION

This class diagram represents the state table as a persistent database, and that we use a controller to control every query to the database. The queries are done automatically by Hibernate. We treat the tables as objects in the application. The session generator is just a class which contains everything that needs to be done when starting a new session.

The start-up of a new session will work as shown in figure 4.17 below. This sequence diagram shows that it will store a session, a state and the pre-set variables from the pDM:

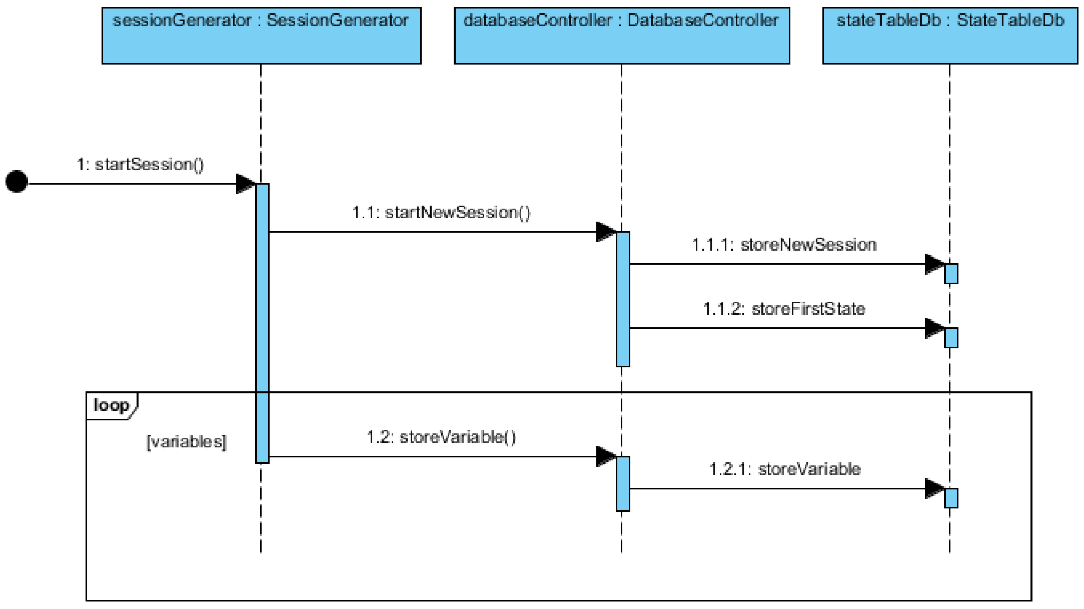


FIGURE 4.17 CREATING A NEW SESSION IN THE STATE TABLE

To delete a state table session will work as shown in the sequence diagram in figure 4.18 below:

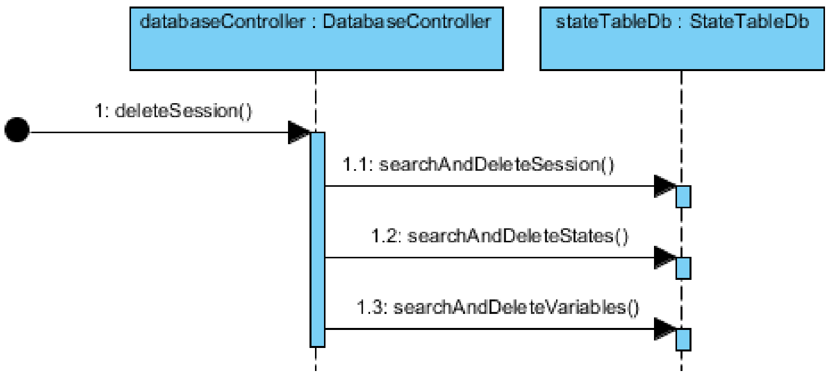

databaseController : DatabaseController
stateTableDb : StateTableDb
1: deleteSession()
1.1: searchAndDeleteSession()
1.2: searchAndDeleteStates()
1.3: searchAndDeleteVariables()

FIGURE 4.18 DELETE A STATE TABLE SESSION

When a state table session is deleted, its states and variables also have to be deleted. This will happen automatically when we delete the state table session.

The next two figures show how to delete a state and a variable, figure 4.19 and 4.20.

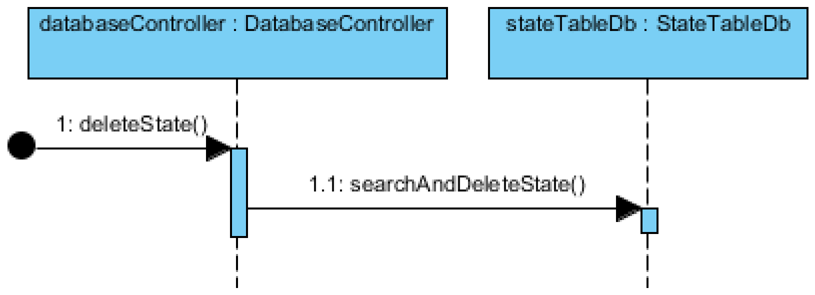

databaseController : DatabaseController
stateTableDb : StateTableDb
1: deleteState()
1.1: searchAndDeleteState()

FIGURE 4.19 DELETE STATE FROM THE STATE TABLE

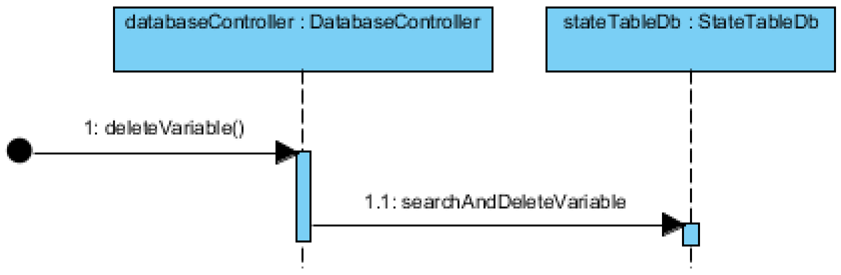

databaseController : DatabaseController
stateTableDb : StateTableDb
1: deleteVariable()
1.1: searchAndDeleteVariable

FIGURE 4.20 DELETE VARIABLE FROM THE STATE TABLE

The methods for fetching states, fetching variables and updating variables are almost identical.

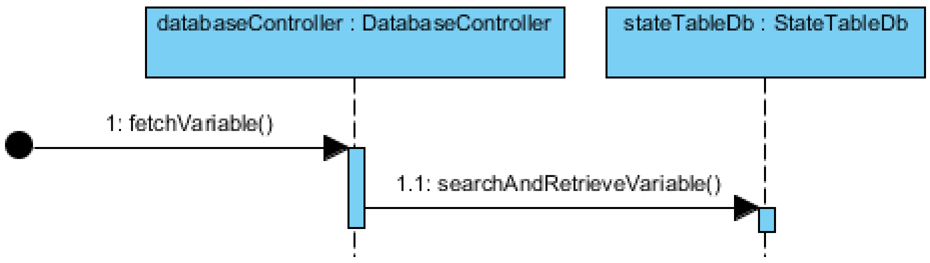

databaseController : DatabaseController
stateTableDb : StateTableDb
1: fetchVariable()
1.1: searchAndRetrieveVariable()

FIGURE 4.21 FETCH VARIABLE FROM THE STATE TABLE

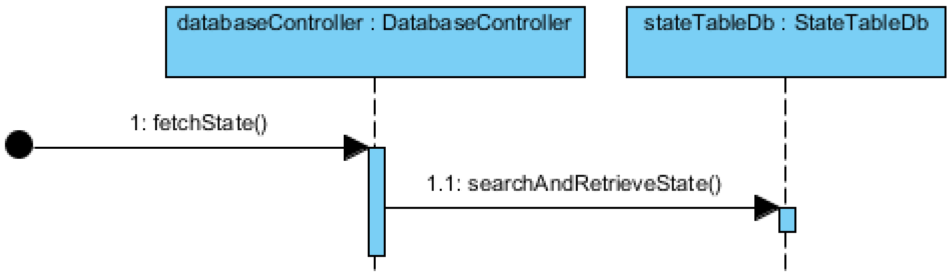

databaseController : DatabaseController
stateTableDb : StateTableDb
1: fetchState()
1.1: searchAndRetrieveState()

FIGURE 4.22 FETCH STATE FROM THE STATE TABLE

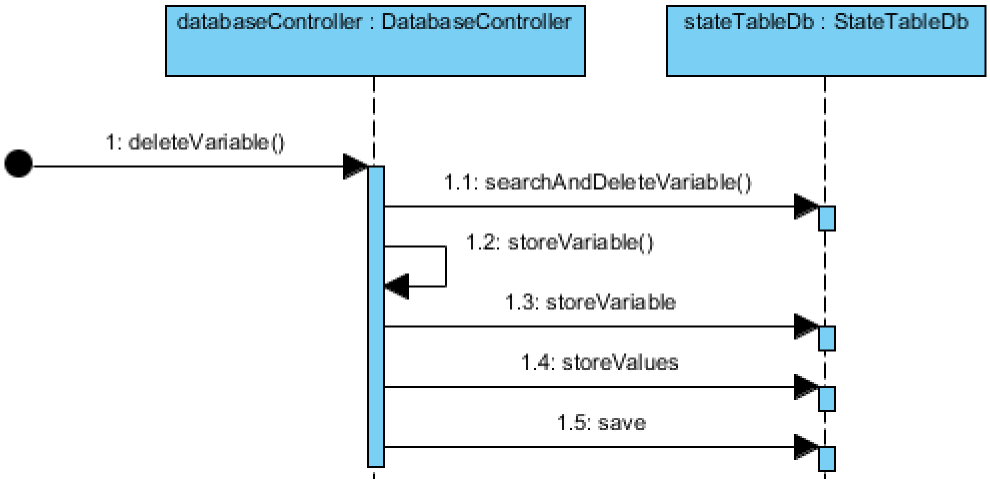

databaseController : DatabaseController
stateTableDb : StateTableDb
1: deleteVariable()
1.1: searchAndDeleteVariable()
1.2: storeVariable()
1.3: storeVariable
1.4: storeValues
1.5: save

FIGURE 4.23 UPDATE A VARIABLE IN THE STATE TABLE

5 REFERENCE(S)

1. *International Specification for technical publications utilizing a common source database S1000D*, Technical Publications Specification Maintenance Group (TPSMG), 01-Aug-2008 (Issue 4.0)

2. *Communicate with a Server*, Google code, http://code.google.com/webtoolkit/doc/latest/DevGuideServerCommunication.html (last visited 06-Mar-2011)

3. *What is Model View Presenter*, WeAsk, http://www.weask.us/entry/model-view-presenter (last visited 06-Mar-2011)

4. *Model-view-presenter*, Wikipedia, http://en.wikipedia.org/wiki/Model_View_Presenter (last visited 06-Mar-2011)

5. *Hibernate,* JBoss Community Hibernate, http://www.hibernate.org (last visited 27-Apr-2011)

6. *Hibernate (Java),* Wikipedia, http://en.wikipedia.org/wiki/Hibernate_%28Java%29 (last visited 27-Apri-2011)

7. *Getting Started with Hibernate (revision 1.4)*, Alan P. Sexton (Published 23-Jan-2006), cs.bham.ac.uk, http://www.cs.bham.ac.uk/~aps/syllabi/2005_2006/issws/h03/hibernate.html (last visited 09-May-2011)

DESIGN DOCUMENT

HIBU STUDENT PROJECT 2011

CORENA S1000D 4.0 PROCESS DATA MODULE RENDERER

Arild Oldervoll

Marius Haraldseth

Eirik André Eidså

Olav Brandt

TABLE OF CONTENTS

REVISION HISTORY			
Version Number	**Date**	**Changes**	**Assigned**
1	07-Mar-2011	First final version	MHA
2	01-Jun-2011	Second Official Verision - Added chapters for: o Caution and Warnings o Saving and Loading sessions o Data Binding subchapter o State Table Database o MVP subchapter - Updated diagrams	MHA

1 INTRODUCTION

Our assignment to develop a logic engine for support execution of S1000D process Data Modules version 4.0, is given by CORENA Norge A/S. This logic engine can take input from users and/or external applications, such as sensors, to decide its next step in a flow of process Data Modules. It can branch and loop through the selected process Data Module. The logic engine will be an interpreter for process Data Modules. We will make it a stand-alone logic engine in a web based application running on Jetty/Tomcat. The user of the application should be able to move back and forth through the steps using previous and next buttons. The user should be able to save the session to continue at a later time. The logic engine should support the International S1000D Specification Issue 4.0.

1.1 INTENTION OF THIS DOCUMENT

The Design Document will contain a closer, in depth overview of the application we are creating, S1000D Logic Engine and Renderer for Process Data Modules (SLRP). It is important to notice that this document only covers the main architecture and design of the parts that has been developed so far.

1.2 SYNONYMS, ACRONYMS AND DEFINITIONS

Synonym	Description
Process	Set of instructions and conditions being executed
Data Module	Container of data/information.
process DM	Process Data Module
MVC	Model-View-Controller
MVP	Model-View-Presenter
GUI	Graphical User Interface
DB	Database
ORM	Object-Relational Mapping
RPC	Remote Procedure Call
SAX	Simple API for XML
DOM	Data Object Model
XML	Extensible Markup Language

1.3 AUTHOR(S)

Marius Haraldseth (MHA)

Arild Oldervoll (AOL)

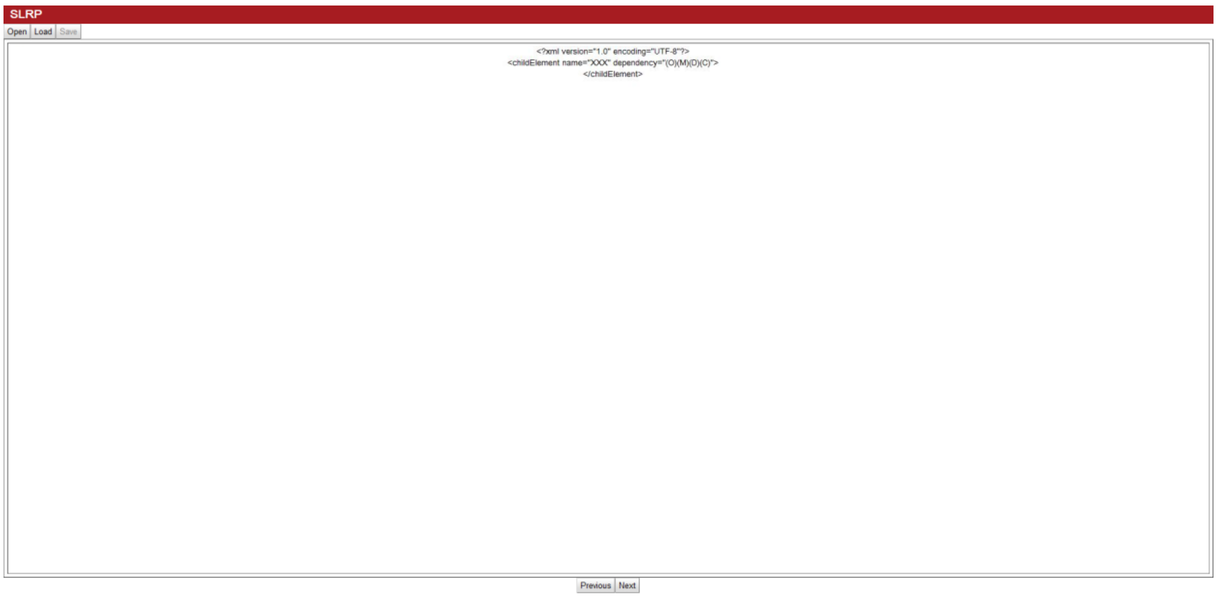
Olav Brandt (OBR)

1.4 ASSIGNED

Arild Oldervoll (AOL)

2 LAYOUT / GUI / VIEW

We have made a decision for the user interface in our application. It is an easy and intuitive layout. In the guidelines from the S1000D specification it is said that Next and Previous buttons should be on top of the content panel, but we have made a decision to put them in the bottom, as we see this as a more logical solution. Our content panel will be a scrollable panel, so the Next and Previous buttons will always be visible.



```
<?xml version="1.0" encoding="UTF-8"?>
<childElement name="XXX" dependency="(O)(M)(D)(C)">
</childElement>
```

FIGURE 2.1 THE VIEW FOR OUR APPLICATION - SLRP

3 SYSTEM STRUCTURE

Our application is a part of a larger system. It is accessed through a web browser on a user's computer, and needs to communicate with the CORENA IETP for certain functionalities, such as a document service.

Figure 3.1.1 shows a deployment diagram for the system our application is a part of.

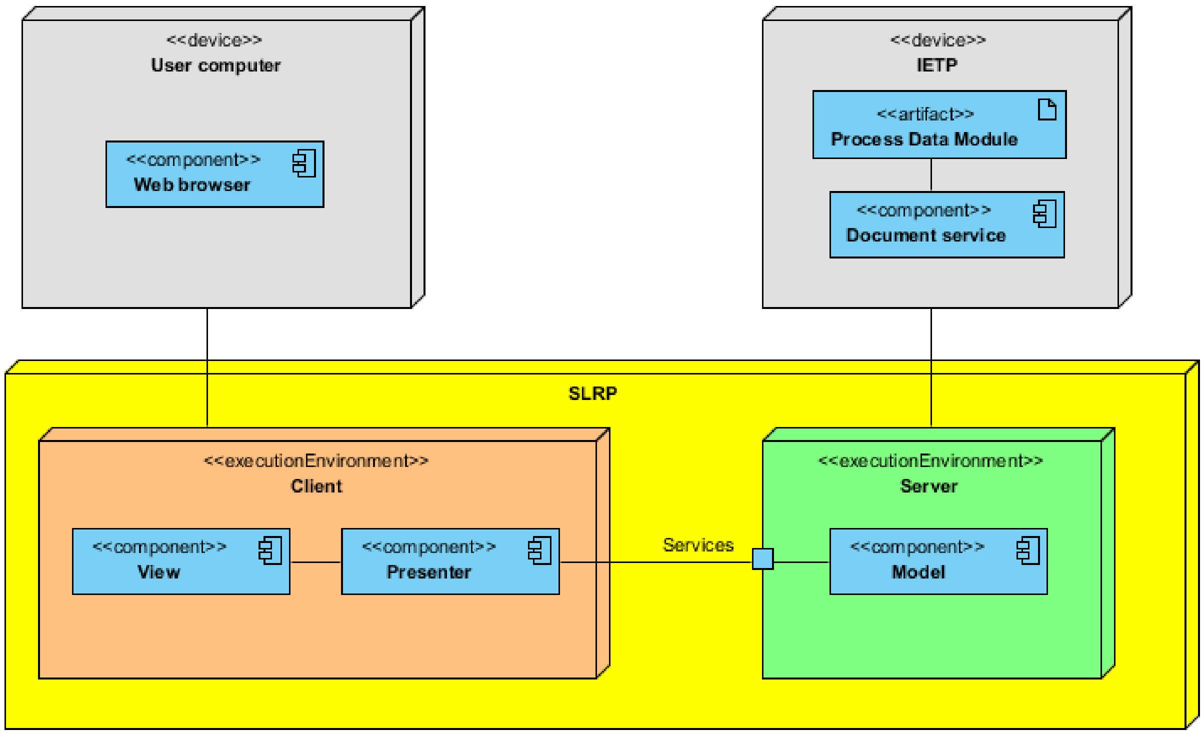


FIGURE 3.1 DEPLOYMENT DIAGRAM FOR THE SYSTEM

4 MODEL-VIEW-PRESENTER

This is the main design pattern for our application SLRP. We will use the model-view-presenter (MVP) pattern. One of the reasons we will use MVP is that it will make testing of the system a lot easier as we split most of the functionality away from GWT. This gives us the opportunity to test all functionality without having to use hosted mode in GWT.

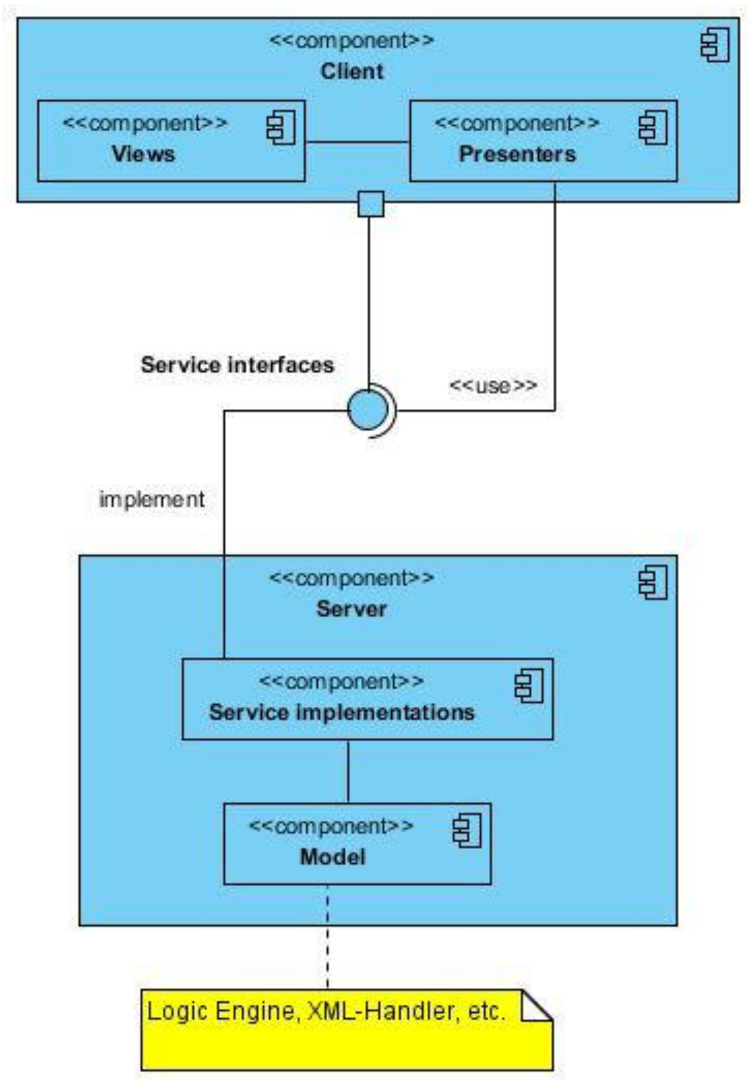


FigurE 4.1 Model-View-Presenter and client/server for SLRP

4.1 MODEL

Our model will be located at the server component. The model will contain the logic engine, the state-table, an XML handler etc. The model is the part which will process and interpret the in the application. The model will be the part which implements our business logic and objects.

4.2 VIEW

An application can have several views. Our application will contain one primary view, which will contain the content panel of our application, the upper button panel with Open, Load and Save buttons and the button panel at the bottom with Previous and Next buttons. The view contains all of the UI components to make up the application.

The view will implement an interface defined in the presenter. It will not fire any events of its own.

4.3 PRESENTER

The presenter part of our application will control what will be displayed in the view, and create the content with the help of functionality in the model. The presenter will prepare and render everything that will be shown in the view.

The presenter will handle, and register all buttons located in the views, and it will present data to our content panel view. It will listen for DOM events, such as ClickHandlers, it will listen on an event bus which we will develop and it will call services.

Our presenter will use an event bus to communicate with services. We will define application events which will be extending GwtEvent. The presenter will then add handlers to listen for events. To do this we will move the ClickHandler away from the view, and handle them in the presenter. An example on how to do this:

display.getOpenButton().addClickHandler()…

4.4 EXTENSION

MVP is a broad and general architecture. For our specific use of it, we will be following the structure described in [5]. In addition to the primary three components of MVP, a fourth is added. This component is responsible for client-side logic that is not limited to one Presenter. It is represented by a class, *AppController*.

Tasks for the new component include the handling of an Event Bus for application-wide events and generating Presenter & View objects.

5 FETCH PDM

The Show PDM functionality described in the analysis document has been split into two separate and distinct functionalities. One of these is dubbed “Fetch PDM”, and describes the way our program fetches a process DM file from the CORENA IETP.

Figure 5.1 shows the functionality’s class diagram.

Figure 5.2 shows the sequence diagram for the functionality.

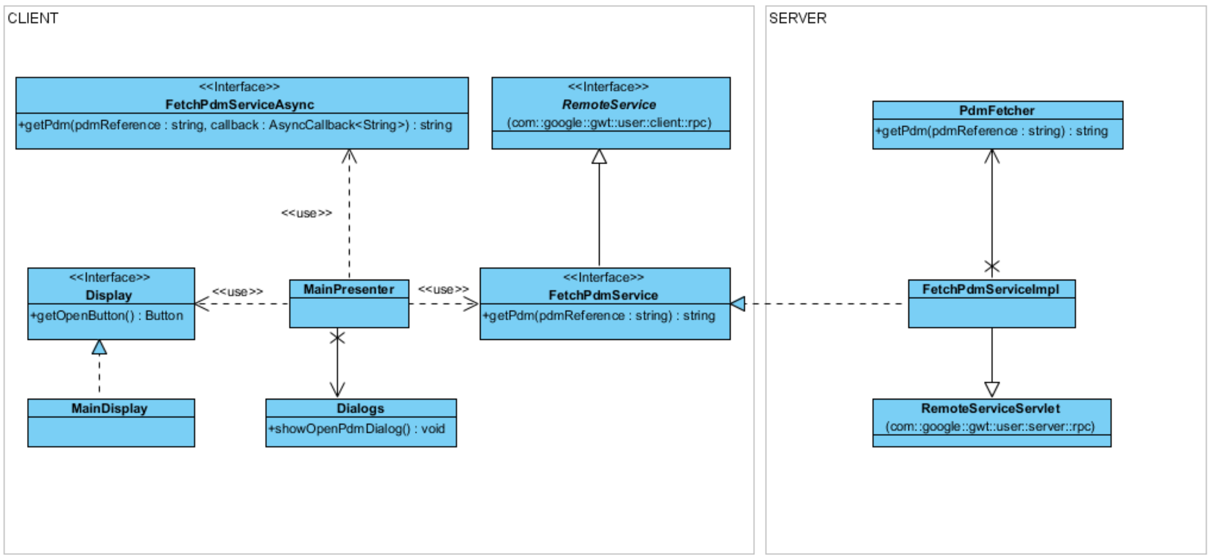

CLIENT
SERVER
<<Interface>> FetchPdmServiceAsync
+getPdm(pdmReference : string, callback : AsyncCallback<String>) : string
<<Interface>> RemoteService (com::google::gwt::user::client::rpc)
PdmFetcher
+getPdm(pdmReference : string) : string
<<use>>
<<Interface>> Display
+getOpenButton() : Button
<<use>>
MainPresenter
<<use>>
<<Interface>> FetchPdmService
+getPdm(pdmReference : string) : string
FetchPdmServiceImpl
MainDisplay
Dialogs
+showOpenPdmDialog() : void
RemoteServiceServlet (com::google::gwt::user::server::rpc)

FIGURE 5.1 CLASS DIAGRAM FOR “FETCH PDM”

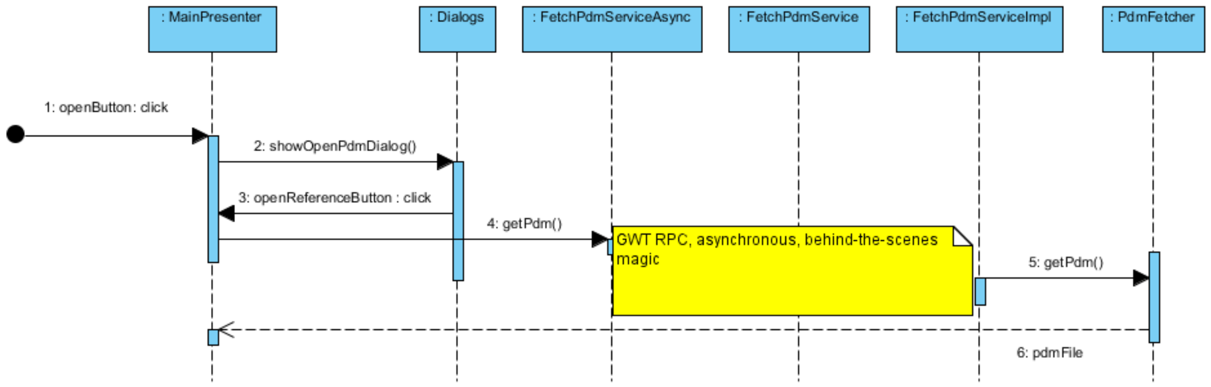


FIGURE 5.2 SEQUENCE DIAGRAM FOR “FETCH PDM”

5.1 PDM PARSING

The data received from the IETP is in an XML format, and to be usable by the rest of the application it must be transferred to a different format that is easier for the application to work with. There are three main options commonly used for parsing and representation of XML data that we have considered to use, these are SAX, DOM and data binding.

5.1.1 SAX

SAX, Simple API for XML, is probably the least suitable parser for our application. It parses the XML from start to end with no possibilities to navigate back and forth through the file.

5.1.2 DOM

DOM, Data Object Model, is a more appropriate choice than SAX for our application, it creates an object for each node or element and a navigable object tree. Each node object knows about its parents, children and siblings and its attributes.

5.1.3 DATA BINDING

XML data binding provides the same features as DOM, but instead of generic node objects, each XML element is mapped to custom Java objects. The advantage is that these objects also can contain business logic and methods related to that exact object, which is useful for our application. Therefore this is the method we have decided on using, and we are using the library XStream for data binding. With XStream, most of the mapping is done automatically and only requires manual mapping in special cases. It also creates a full object graph as in DOM.

Data binding requires that a class is created for each element that is supposed to be mapped. The classes contain the legal attributes and child elements stored as variables, as well as getter and setter methods for each of these. These classes can be found in the package *com.corena.ietp.slrp.server.pdm.elements* as sub-packages. These classes are mostly a representation of elements specified in the S1000D specification, as documented by AMEO in the “S1000D PDM” documents. The class *PdmElementGenerator* is where the actual data binding takes place as shown in Figure 5.3.

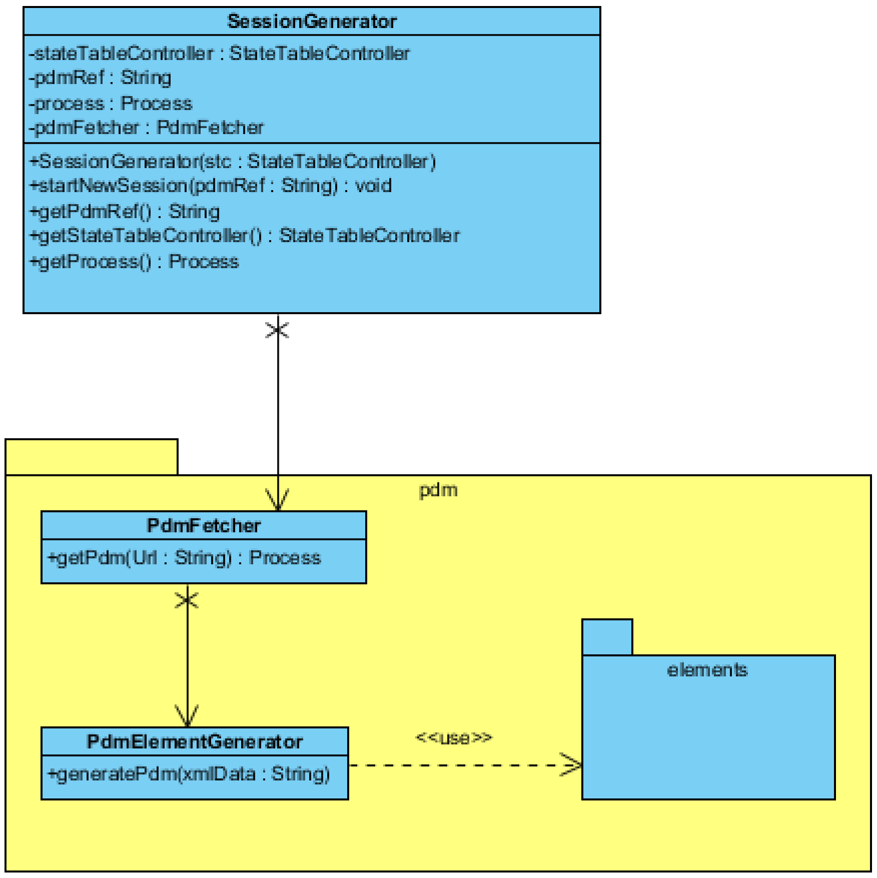

SessionGenerator
-stateTableController : StateTableController
-pdmRef : String
-process : Process
-pdmFetcher : PdmFetcher
+SessionGenerator(stc : StateTableController)
+startNewSession(pdmRef : String) : void
+getPdmRef() : String
+getStateTableController() : StateTableController
+getProcess() : Process
pdm
PdmFetcher
+getPdm(Url : String) : Process
elements
<<use>>
PdmElementGenerator
+generatePdm(xmlData : String)

FIGURE 5.3 PDM FETCHING AND DATA BINDING

6 CLIENT AND SERVER

6.1 CLIENT

Our client component is the application frontend and will hold the view and presenter parts of the application. This is the mainly where GWT will be used in our application and the source code will be compiled into JavaScript. The main client components are shown in Figure 6.1

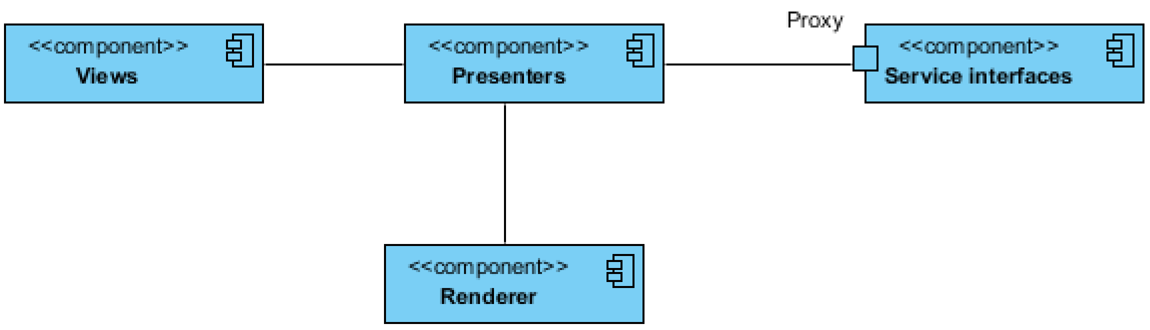


FIGURE 6.1 Client components of SLRP

6.2 SERVER

Our server component will be the component containing the model. Most of the functionality will be contained here. The model, or server if you'd like contains the logic engine, which is our main and most important functionality of the application. Other parts of the model are the state table, the XML handler and the communication component.

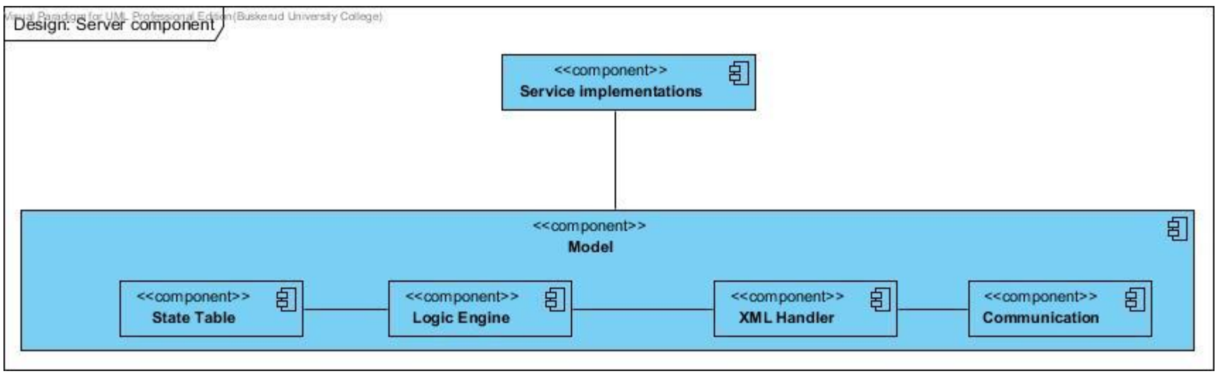


FigurE 6.2 Server Component of SLRP

6.3 CLIENT / SERVER / SERVICE

The client will have two service interfaces, a normal interface, and an asynchronous interface which it will use to call the service implementation located in the model at the server component. The server implements the normal interface located in the client, and the client will use this interface. The client will receive a callback from the server with the result from the service call.

6.3.1 COMMAND PATTERN

We will use RPC calls to the server services. These services will need three classes each (xxx = name for the service):

Located at the client:

- Service Interface (xxxService)
    - Annotate with: @RemoteServiceRelativePath("name")
- Async version of the interface (xxxServiceAsync)

Located at the server:

- Server implementation (xxxServiceImpl)

This pattern is called the Command Pattern. Since the call is asynchronous, you will receive a callback from the service implementation with the result.

7 DATA OBJECT MODEL (DOM)

Java offers mainly two different ways of parsing XML documents and storing them in memory; DOM (Data Object Model) or SAX (Simple API for XML). We have chosen DOM for our application. SAX is faster in use than DOM, but it does not organise or model the data parsed, so we would have to create our own data structure. With DOM the data is organised in a tree structure organised by the elements in the XML file, a Document Object Model. This is slower than SAX and takes more memory, but it allows traversing of the XML document forward and backwards.

If DOM turns out to be too slow for our application when we start working on large data files, it is always possible to change to SAX later on, but this will be more time consuming than using DOM, as we would have to create our own data structure.

8 DATABASE SYSTEM

For the state table in SLRP we must use a persistent database system. We have decided to use the Hibernate library for Java with the Apache Derby Database.

8.1 HIBERNATE

We will use Hibernate so we can treat our database in an object-oriented way. Hibernate is an object-relational mapping (ORM) library for the Java programming language. It provides a framework for mapping an object-oriented domain model to a traditional relational database. We use Hibernate together with the Apache Derby database to gain persistence in our system.

There are several ways to map Java classes to database tables, either by using annotations or an XML mapping file. We will use the XML mapping file. These mapping files' extension is *.hbm.xml*, and they are stored together with the classes that represent the mapping files' tables.

Other than the mapping files, we have a configuration file, which is known by its *.cfg.xml* extension. This file is placed in the root of the *src* folder of the project. The configuration file contains mappings for the drivers and connection for the database as well as a lot of other options, both mandatory and optional.

Hibernate3 require these libraries in the build path (the versions we have used):

- Hibernate3.jar
- antlr-2.7.6.jar
- commons-collections-3.1.jar
- dom4j-1.6.1.jar
- javaassist-3.12.0.GA.jar
- jta-1.1.jar
- slf4j-api-1.6.1.jar
- slf4j-simple-1.6.1.jar
- hibernate-jpa-2.0-api-1.0.0.Final.jar
- hibernate-validator-4.0.0.Beta2.jar
- validation-api-1.0.CR3.jar

We have used Hibernate version 3.6.3 Final Distribution.

To use Hibernate we have created a standard *HibernateUtil* class which generates a *SessionFactory* for communicating with the database. The *SessionFactory* shall only be created once and later on utilized in the communication with the database. For each time we then communicate with the database we create a *Session* which we have to close at the end of the communication.

What happens when we perform a save query to the database is that we first create a *transient* object, which we makes *persistent* with:

- *session.save(object);*

Now the object is persistent, but it is not saved until the query is committed. To commit we use this statement:

- *session.getTransaction().commit();*

The object(s) is/are now saved as row(s) in the database, and persistence is obtained. Diagrams which present this are shown in figure 8.3 and 8.4 in chapter 8.4.1.

Be aware that we in this chapter are talking about two different kinds of sessions. The first one, which is mentioned above is a Hibernate session, which is created every time we interact with the database. The second is one *run* or *session* in the state table, which represents one run with the program when a user opens a new manual. It can be saved, and continued by another user at a later time.

8.1.1 DEFININTIONS

Transient object – The object is not represented in the database.

Persistent object – The object is represented in the database.

Detached object – The object has been persistent, but is not anymore. It is in a stale state.

8.2 APACHE DERBY

We will use Apache Derby as our database system since we only need a small embedded database in our application. Derby is a small and easy-to-use full featured relational SQL database. It is also free under the Apache license. Derby is portable and 100% Java and it is only 2,5 MB.

We use a Derby client database. To use this we need one library in the build path:

- derbyclient.jar

8.3 STATE TABLE DATABASE MODEL

SLRP's state table database model:

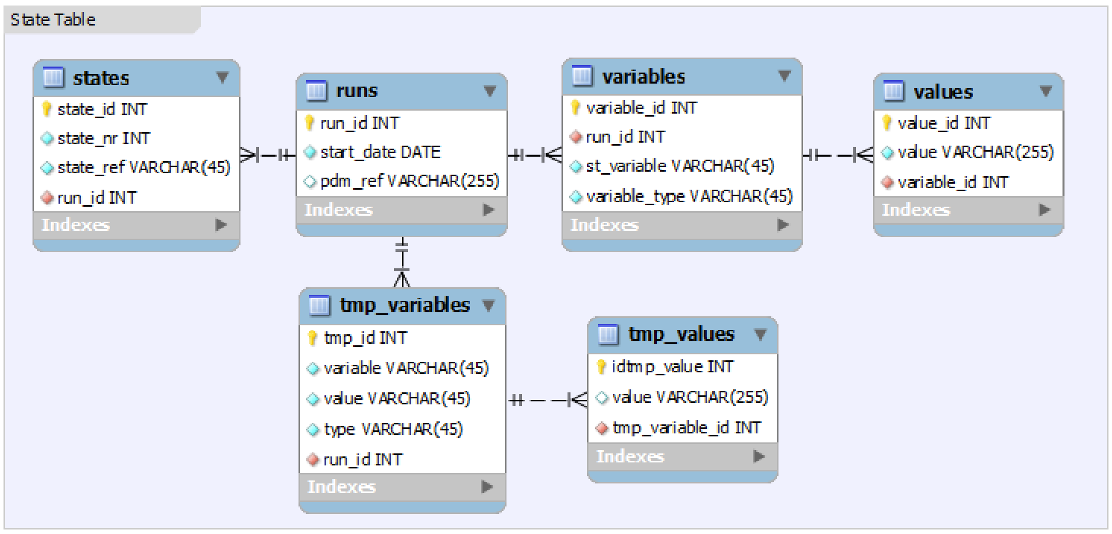


Figure 8.1 State Table Database Model

8.3.1 STATE TABLE

These are the State Table tables for the SLRP application which holds the data for each session.

The *session's* table is the table containing the identification number of a session, and its start date. The *session_id* is the primary key for this table.

The *states* table is the table holding the history and reference to each state, or *<dmNode>* in a session. The *state_id* is this table's key.

The *variables* table is the table containing all the variables, with connection to their values. The variables are global in our program, but the variable name will be used for several sessions, and therefore we have a *variable_id* as a unique and primary key.

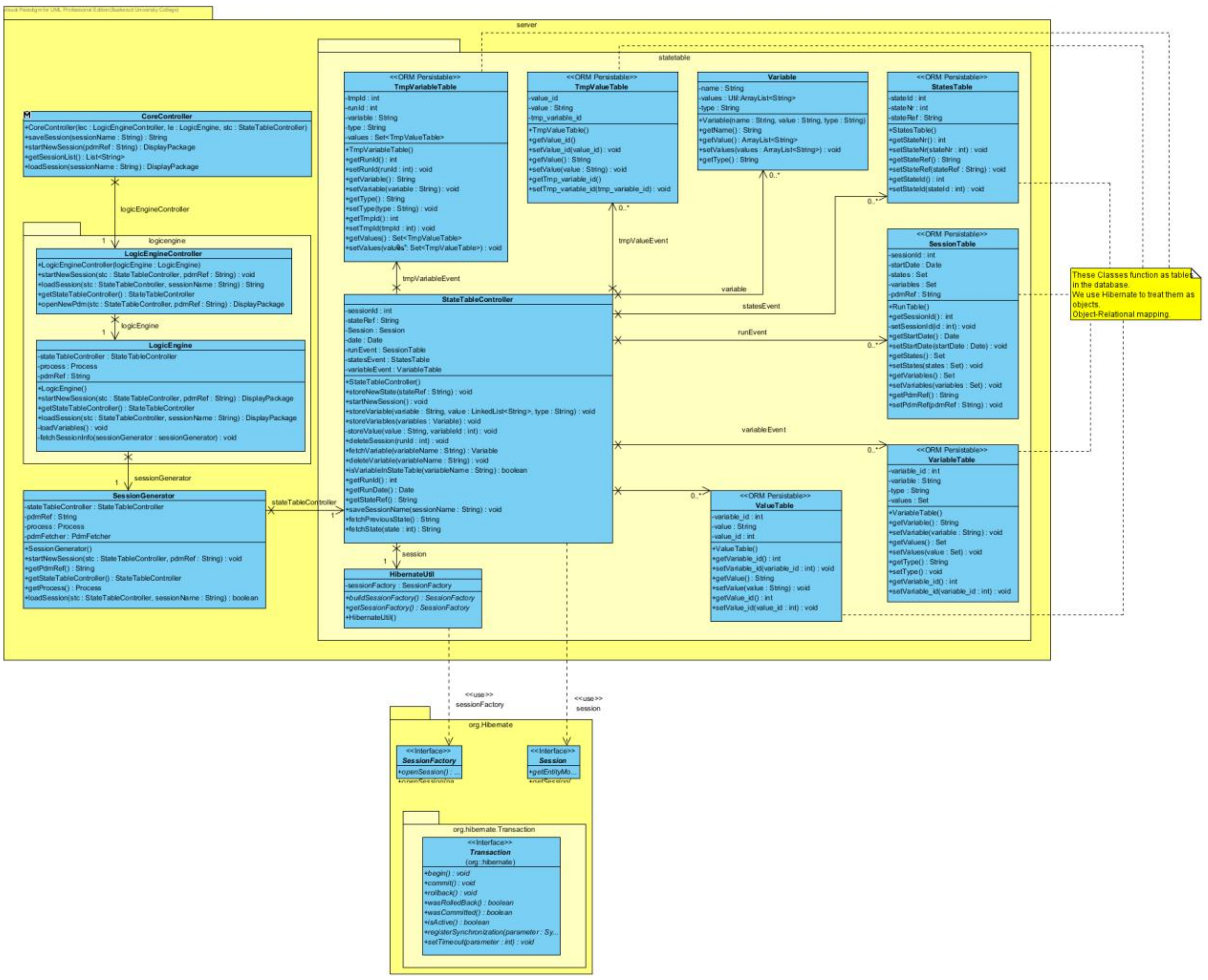
The *values* table is where the values for the variables will be stored. The connection to the variable is in the *variable_id.* The primary key is the *value_id.*

The *tmp_variables* table is where the temporary variables which dialogues will use are saved until they are done and validated. When they are validated, the values will be stored in the State Table tables. Connection to the session is through the *session_id.* The primary key is the *tmp_id.*

The *tmp_values* table is the table which stores the values for the temporary variables.

8.4 STATE TABLE APPLICATION STRUCTURE

In SLRP our database and Hibernate system will be represented as figure 8.2.



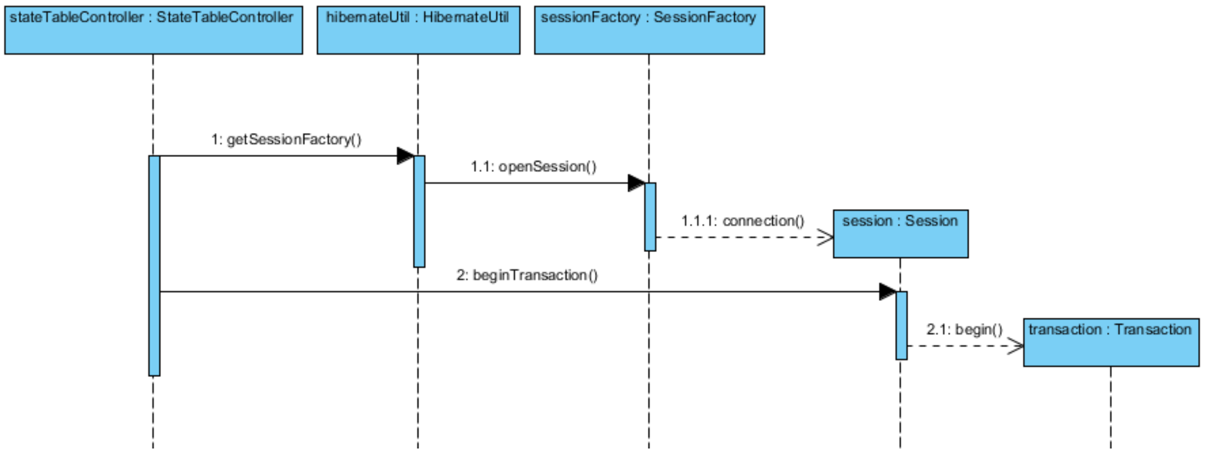
server
statetable
CoreController
logicEngineController
logicengine
LogicEngineController
logicEngine
LogicEngine
sessionGenerator
SessionGenerator
stateTableController
<<ORM Persistable>> TmpVariableTable
<<ORM Persistable>> TmpValueTable
Variable
<<ORM Persistable>> StatesTable
<<ORM Persistable>> SessionTable
<<ORM Persistable>> VariableTable
<<ORM Persistable>> ValueTable
StateTableController
HibernateUtil
tmpVariableEvent
tmpValueEvent
variable
statesEvent
runEvent
variableEvent
session
These Classes function as tables in the database. We use Hibernate to treat them as objects. Object-Relational mapping.
<<use>> sessionFactory
<<use>> session
org.Hibernate
<<Interface>> SessionFactory
<<Interface>> Session
org.hibernate.Transaction
<<Interface>> Transaction (org::hibernate)
+begin() : void
+commit() : void
+rollback() : void
+wasRolledBack() : boolean
+wasCommitted() : boolean
+isActive() : boolean
+setTimeout(parameter : int) : void

FIGURE 8.2 STATE TABLE SYSTEM IN SLRP

The upper package in this diagram is our application. The lower package is the org.Hibernate package which we use for our mapping with the database. This is the complete system for our state table. We have a *StateTableController* which is the core which controls everything, and the *HibernateUtil* which is the standard static singleton pattern that generates the *SessionFactory.* The rest of the classes are object-relational mapping (ORM) tables, represented as objects/classes in Java.

8.4.1 OPEN AND CLOSE HIBERNATE SESSION

This is the method to start a new session with Hibernate against the database. This method will be used every time we communicate with the database, and returns a session object. See the diagram below:



stateTableController : StateTableController
hibernateUtil : HibernateUtil
sessionFactory : SessionFactory
1: getSessionFactory()
1.1: openSession()
1.1.1: connection()
session : Session
2: beginTransaction()
2.1: begin()
transaction : Transaction

FIGURE 8.3 START HIBERNATE SESSION

We need to save (persist) and commit the session before we close it each time, as shown in figure 8.4.

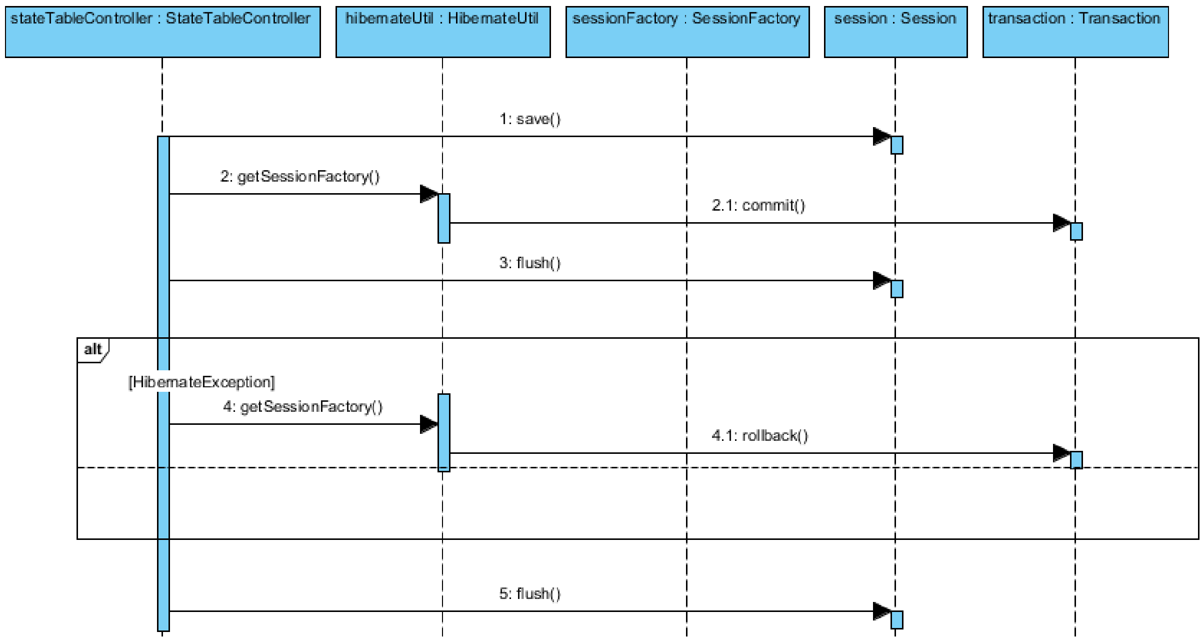
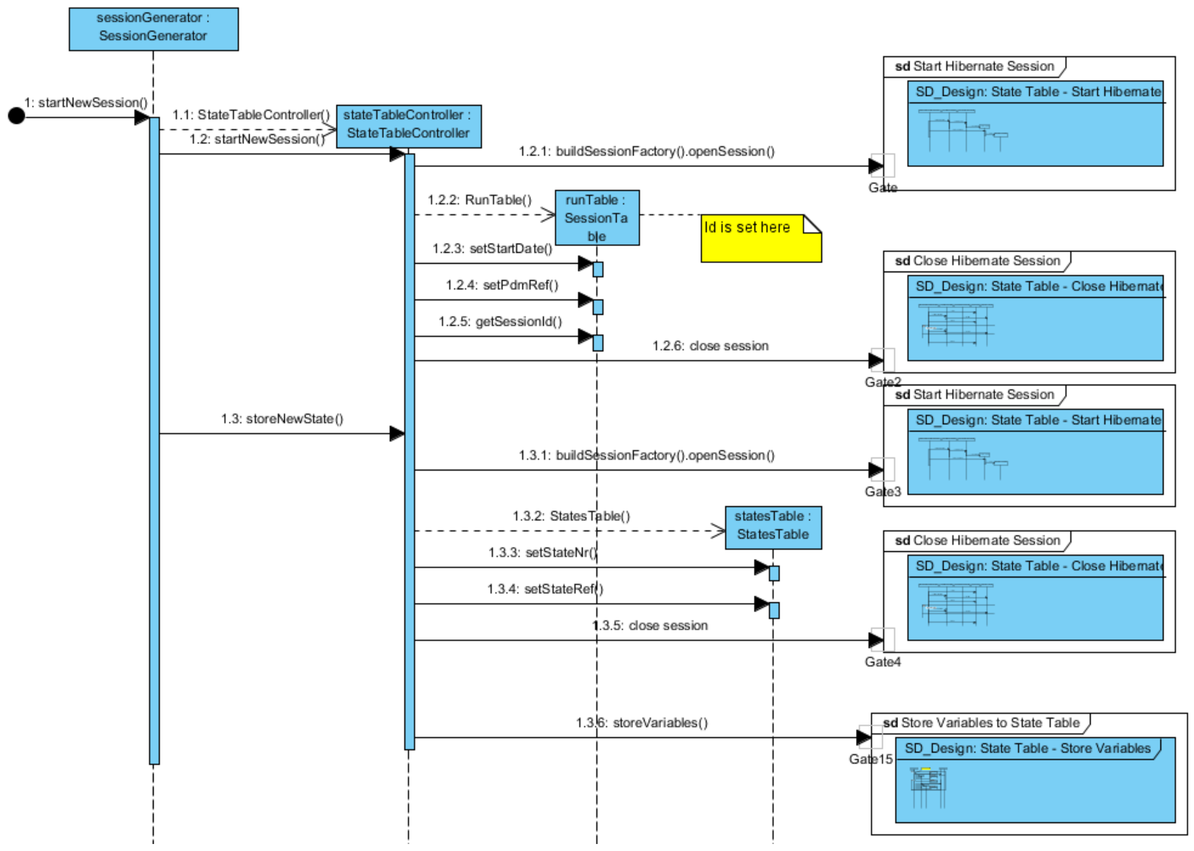


FIGURE 8.4 CLOSE HIBERNATE SESSION

8.4.2 START A NEW STATE TABLE SESSION AND SAVING TO STATE TABLE

When a new session is started in the database, we first insert a row into the run table, then we insert the first state, and finally we insert pre-set variables, if there are any. See the diagram in figure 8.5.


sessionGenerator : SessionGenerator
1: startNewSession()
1.1: StateTableController()
1.2: startNewSession()
stateTableController : StateTableController
1.2.1: buildSessionFactory().openSession()
1.2.2: RunTable()
runTable : SessionTable
Id is set here
1.2.3: setStartDate()
1.2.4: setPdmRef()
1.2.5: getSessionId()
1.2.6: close session
1.3: storeNewState()
1.3.1: buildSessionFactory().openSession()
1.3.2: StatesTable()
statesTable : StatesTable
1.3.3: setStateNr()
1.3.4: setStateRef()
1.3.5: close session
1.3.6: storeVariables()
sd Start Hibernate Session
SD_Design: State Table - Start Hibernate
Gate
sd Close Hibernate Session
SD_Design: State Table - Close Hibernate
Gate2
sd Start Hibernate Session
SD_Design: State Table - Start Hibernate
Gate3
sd Close Hibernate Session
SD_Design: State Table - Close Hibernate
Gate4
sd Store Variables to State Table
SD_Design: State Table - Store Variables
Gate15

FIGUR 8.5 STARTING A NEW STATE TABLE SESSION

The methods used to store variables and states are the same methods as is being used to perform these operations as single instructions when the program is running. There is one exception, and that is when we want to store several variables, we have added another method, see figures 8.6 and 8.7.

The methods for saving variables and values to the state table contain one parameter; *isTmpVariable* or *isTmpValue.* This is a Boolean parameter which tells the methods if they should store to the temporary tables or the regular state table tables. A value of *true* saves to the temporary, and *false* to the other.

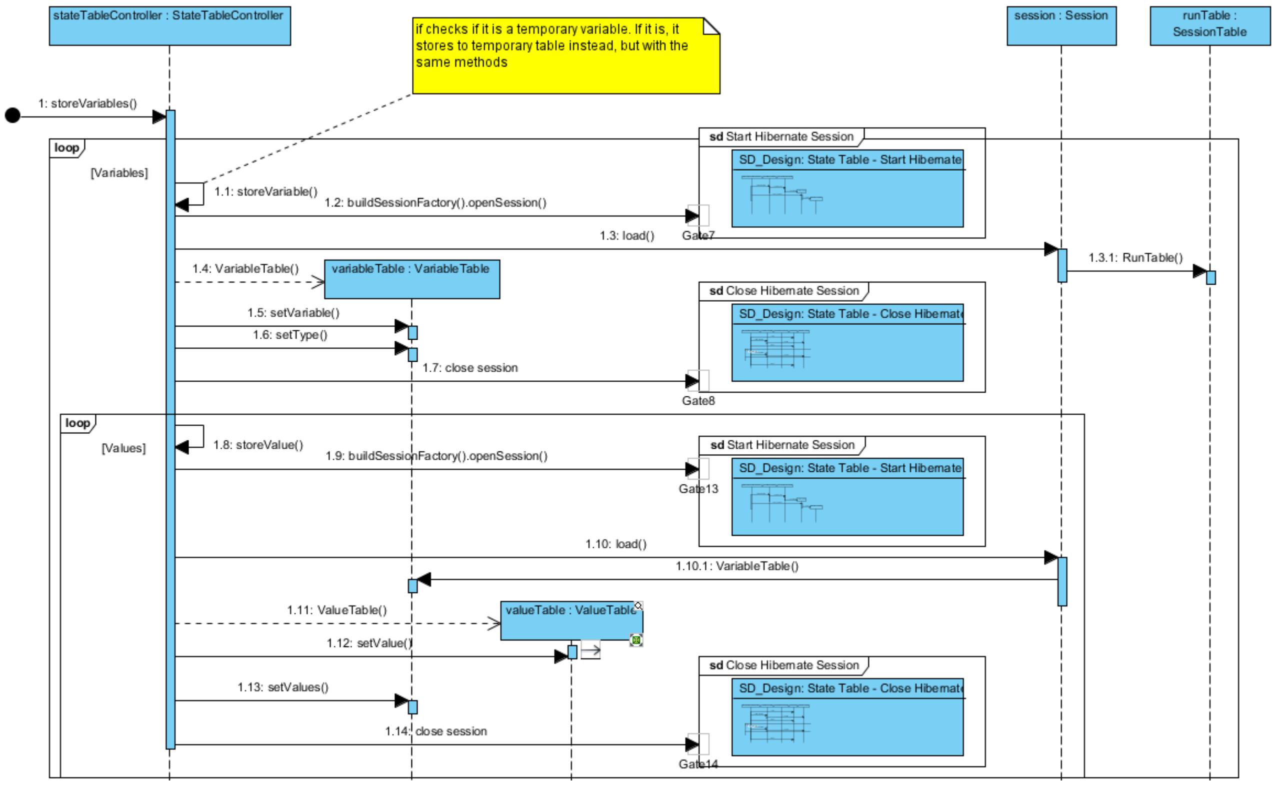
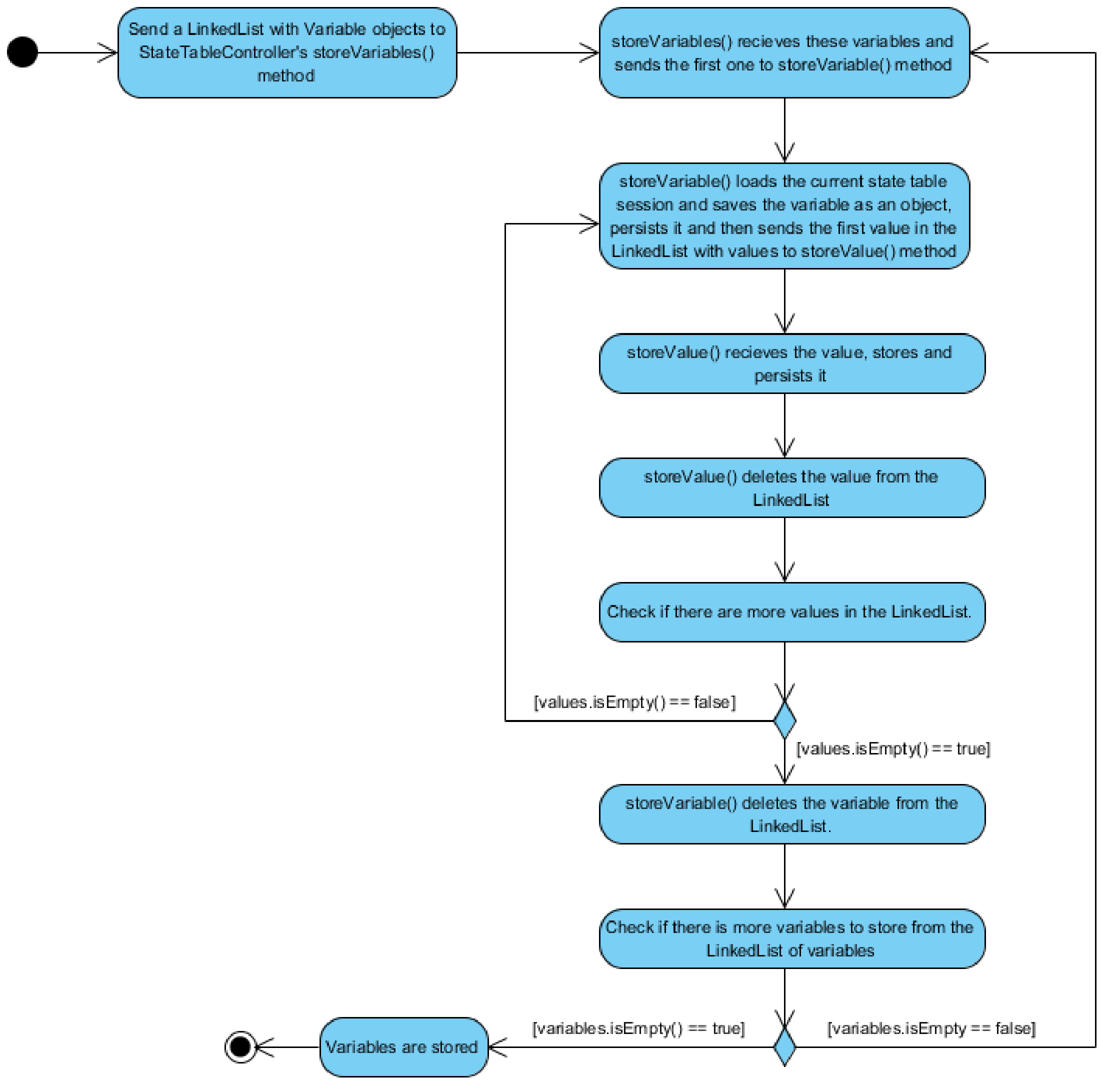

stateTableController : StateTableController
if checks if it is a temporary variable. If it is, it stores to temporary table instead, but with the same methods
session : Session
runTable : SessionTable
1: storeVariables()
loop
[Variables]
1.1: storeVariable()
1.2: buildSessionFactory().openSession()
sd Start Hibernate Session
SD_Design: State Table - Start Hibernate
Gate7
1.3: load()
1.3.1: RunTable()
1.4: VariableTable()
variableTable : VariableTable
1.5: setVariable()
1.6: setType()
sd Close Hibernate Session
SD_Design: State Table - Close Hibernate
1.7: close session
Gate8
loop
[Values]
1.8: storeValue()
1.9: buildSessionFactory().openSession()
sd Start Hibernate Session
SD_Design: State Table - Start Hibernate
Gate13
1.10: load()
1.10.1: VariableTable()
1.11: ValueTable()
valueTable : ValueTable
1.12: setValue()
1.13: setValues()
sd Close Hibernate Session
SD_Design: State Table - Close Hibernate
1.14: close session
Gate14

FIGURE 8.6 STORE VARIABLES TO STATE TABLE

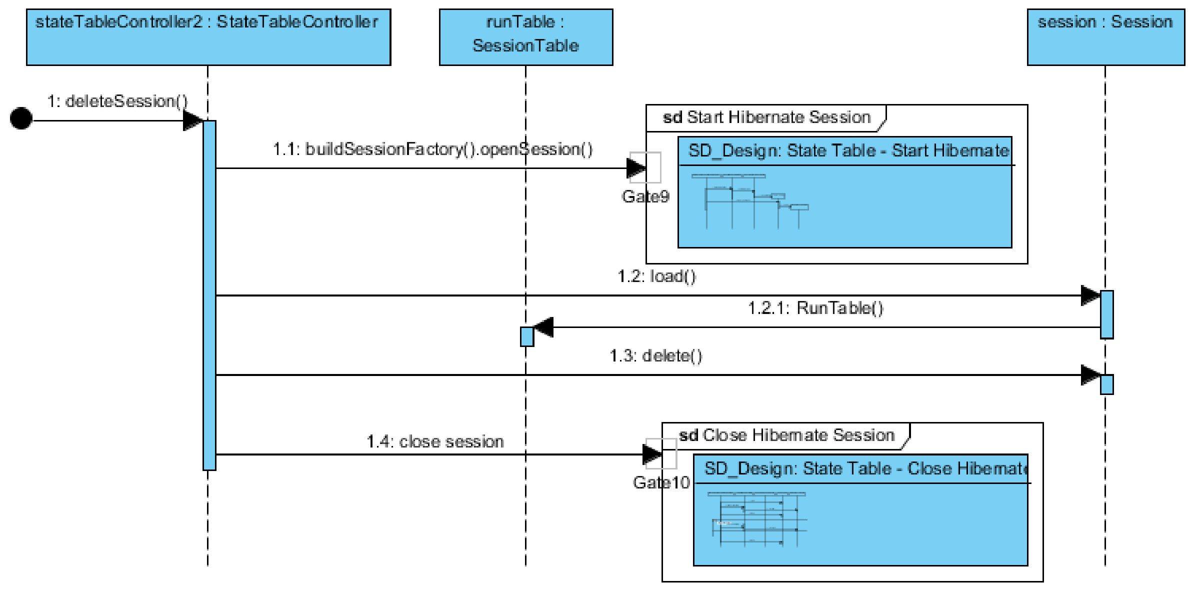
The only real difference is that the method *storeVariables()* receives a linked list with variables objects which is being inserted into state table one by one with the *storeVariable()* method. The *storeVariable()* method uses the *storeValue()* method to store the values of the variable to state table.


Send a LinkedList with Variable objects to StateTableController's storeVariables() method
storeVariables() recieves these variables and sends the first one to storeVariable() method
storeVariable() loads the current state table session and saves the variable as an object, persists it and then sends the first value in the LinkedList with values to storeValue() method
storeValue() recieves the value, stores and persists it
storeValue() deletes the value from the LinkedList
Check if there are more values in the LinkedList.
[values.isEmpty() == false]
[values.isEmpty() == true]
storeVariable() deletes the variable from the LinkedList.
Check if there is more variables to store from the LinkedList of variables
[variables.isEmpty() == true]
[variables.isEmpty == false]
Variables are stored

FIGUR 8.7 STORE VARIABLES TO THE STATE TABLE

8.4.3 DELETE STATE TABLE SESSION

To delete a state table session, every state and variable belonging to that specific session also have to be deleted. The variables and states are take care of by Hibernate. We just have to delete the session object/row. The sequence diagram for deleting a state table session is shown in figure 8.8.

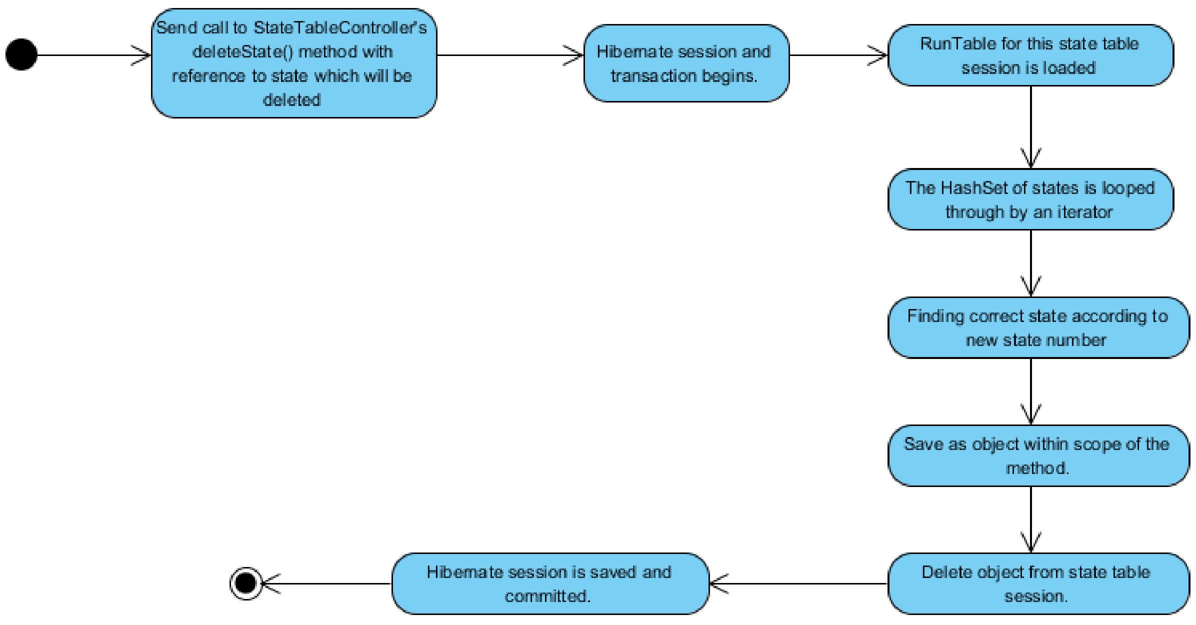


stateTableController2 : StateTableController
runTable : SessionTable
session : Session
1: deleteSession()
1.1: buildSessionFactory().openSession()
sd Start Hibernate Session
SD_Design: State Table - Start Hibernate
Gate9
1.2: load()
1.2.1: RunTable()
1.3: delete()
1.4: close session
sd Close Hibernate Session
SD_Design: State Table - Close Hibernate
Gate10

FIGURE 8.8 DELETE A STATE TABLE SESSION

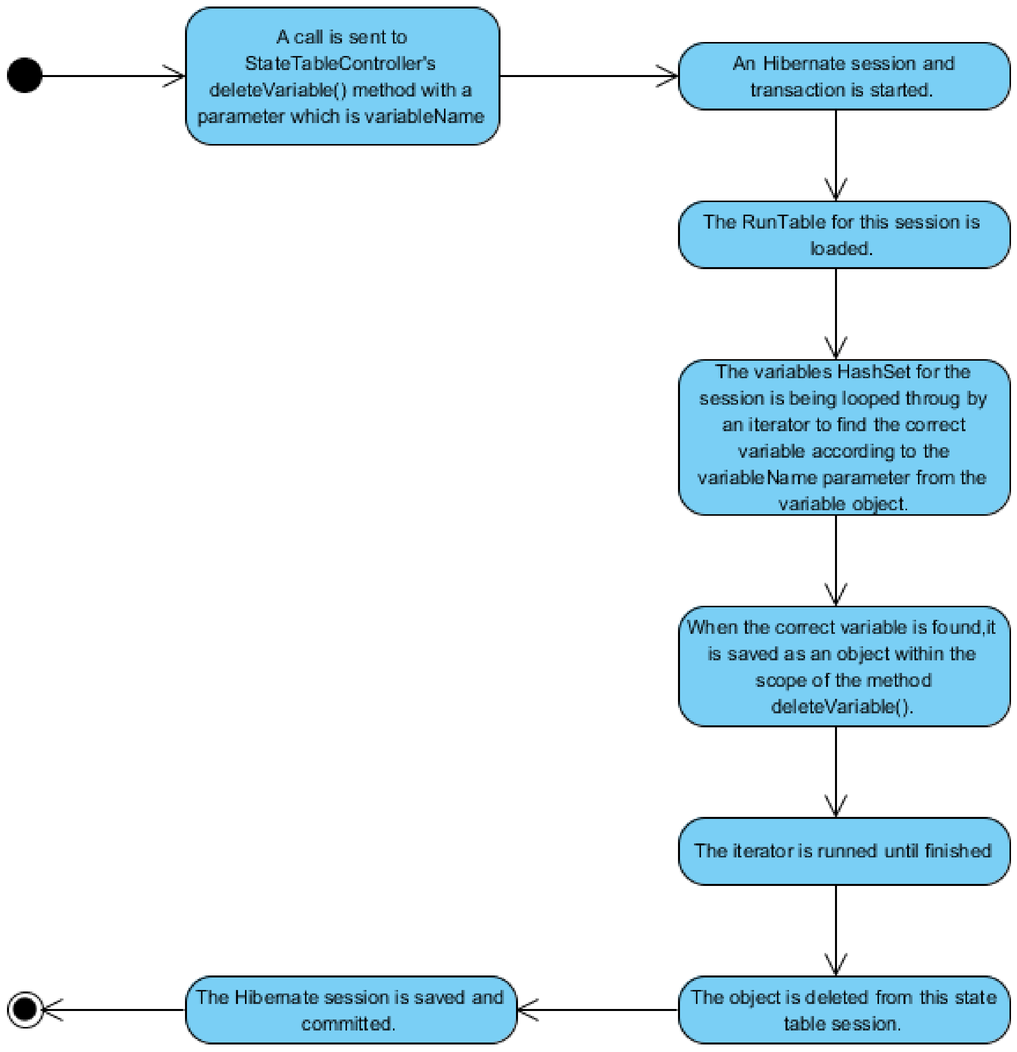
8.4.4 DELETE STATES AND VARIABLES

To delete states and variables we need to enter the state table session and iterate through them to find the one we are looking for. Then we delete this object from the Hibernate Session object. See activity diagrams 8.9 and 8.10.



Send call to StateTableController's deleteState() method with reference to state which will be deleted
Hibernate session and transaction begins.
RunTable for this state table session is loaded
The HashSet of states is looped through by an iterator
Finding correct state according to new state number
Save as object within scope of the method.
Delete object from state table session.
Hibernate session is saved and committed.

FIGURE 8.9 DELETE A STATE FROM THE STATE TABLE



A call is sent to StateTableController's deleteVariable() method with a parameter which is variableName
An Hibernate session and transaction is started.
The RunTable for this session is loaded.
The variables HashSet for the session is being looped throug by an iterator to find the correct variable according to the variableName parameter from the variable object.
When the correct variable is found,it is saved as an object within the scope of the method deleteVariable().
The iterator is runned until finished
The object is deleted from this state table session.
The Hibernate session is saved and committed.

FIGURE 8.10 DELETE A VARIABLE FROM THE STATE TABLE

8.4.5 FETCH STATES FROM THE STATE TABLE

To fetch a state from the state table there are two options.

- Fetch the previous state.
- Fetch a state by a number.

Both of these methods use the *fetchState()* method, which fetches a state by using the number of the state. The difference is that to fetch the previous state we use a method *fetchPreviousState()* before the *fetchState()* method. What this extra method does, is to first delete the current state, and then decrement the *stateNr* for the *StateTableController.* Then it sends a call to *fetchState()* with the new *stateNr* as parameter. See figure 8.11. The activity diagram in this figure shows both of these methods.

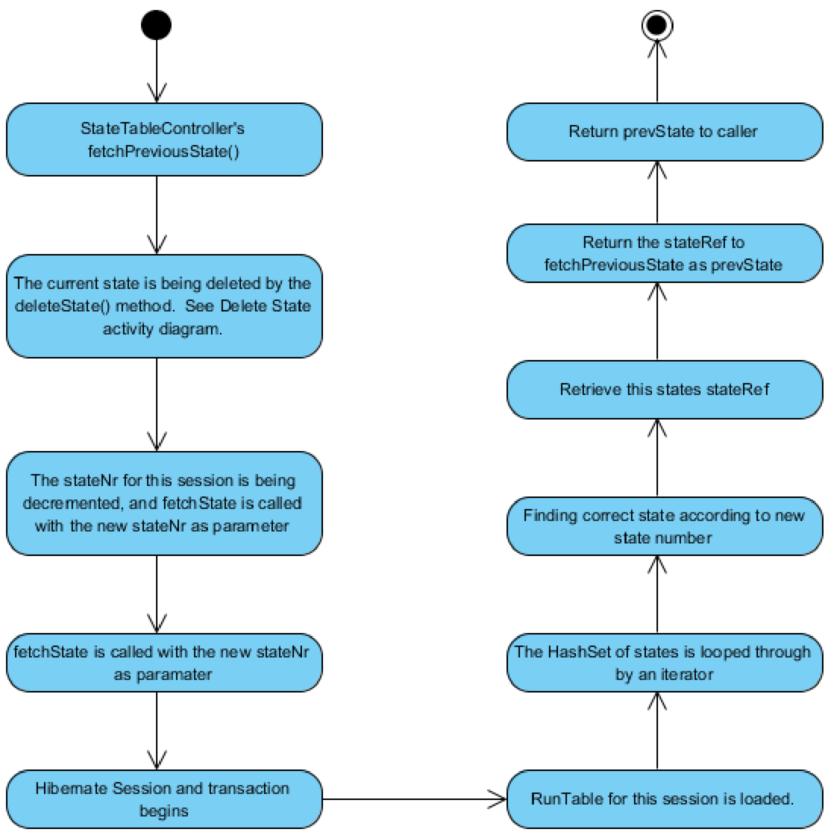

StateTableController's fetchPreviousState()
The current state is being deleted by the deleteState() method. See Delete State activity diagram.
The stateNr for this session is being decremented, and fetchState is called with the new stateNr as parameter
fetchState is called with the new stateNr as paramater
Hibernate Session and transaction begins
RunTable for this session is loaded.
The HashSet of states is looped through by an iterator
Finding correct state according to new state number
Retrieve this states stateRef
Return the stateRef to fetchPreviousState as prevState
Return prevState to caller

FIGURE 8.11 FETCH THE PREVIOUS STATE FROM THE STATE TABLE

8.4.6 FETCH VARIABLES FROM THE STATE TABLE

To fetch variables from the state table we must load the current state table session object and iterate though it's variables in the *Set*. When we find the variable we are looking for, we save its values to a LinkedList and store a new *Variable* object in the application with these values, the variable name and type. See figure 8.12.

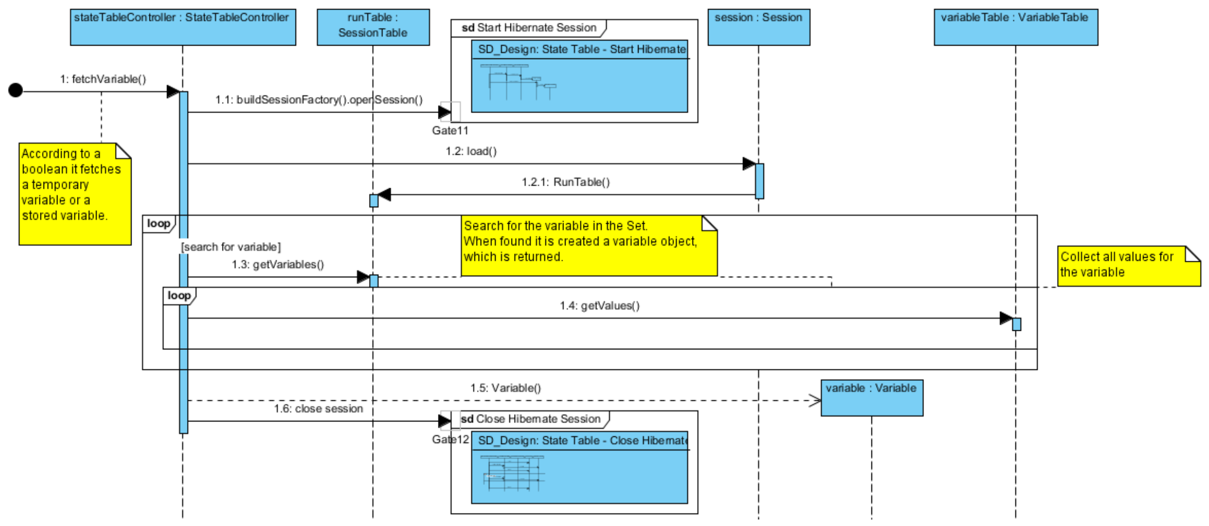

stateTableController : StateTableController
runTable : SessionTable
sd Start Hibernate Session
SD_Design: State Table - Start Hibernate
session : Session
variableTable : VariableTable
1: fetchVariable()
1.1: buildSessionFactory().openSession()
Gate11
1.2: load()
1.2.1: RunTable()
According to a boolean it fetches a temporary variable or a stored variable.
loop
[search for variable]
1.3: getVariables()
Search for the variable in the Set. When found it is created a variable object, which is returned.
Collect all values for the variable
loop
1.4: getValues()
1.5: Variable()
variable : Variable
1.6: close session
sd Close Hibernate Session
Gate12
SD_Design: State Table - Close Hibernate

FIGURE 8.12 FETCH A VARIABLE FROM THE STATE TABLE

9 SAVING AND LOADING SESSIONS

SLRP will have functionality for saving a session, and continuing where you left off.

9.1 SAVE SESSION

Once the session is saved, all the work you do afterwards is also automatically saved. When we save a session, a Boolean value *isSaved* in the *StateTableController* is set to true, and a name for the session is saved in the state table. The process is showed in figure 9.1 below.

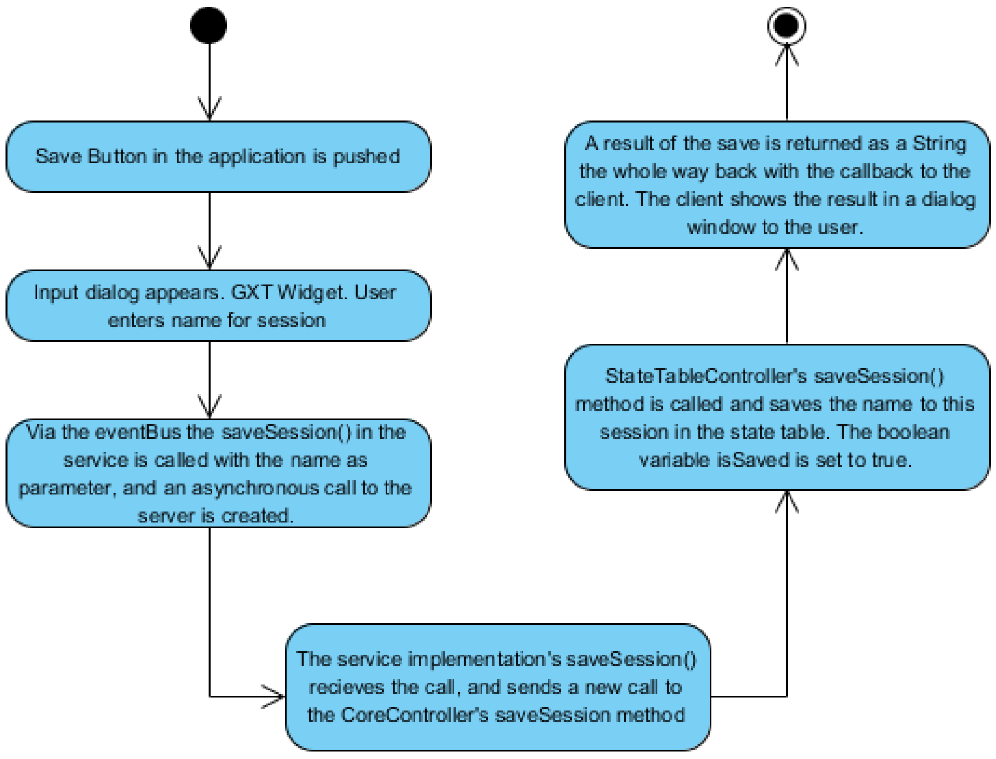
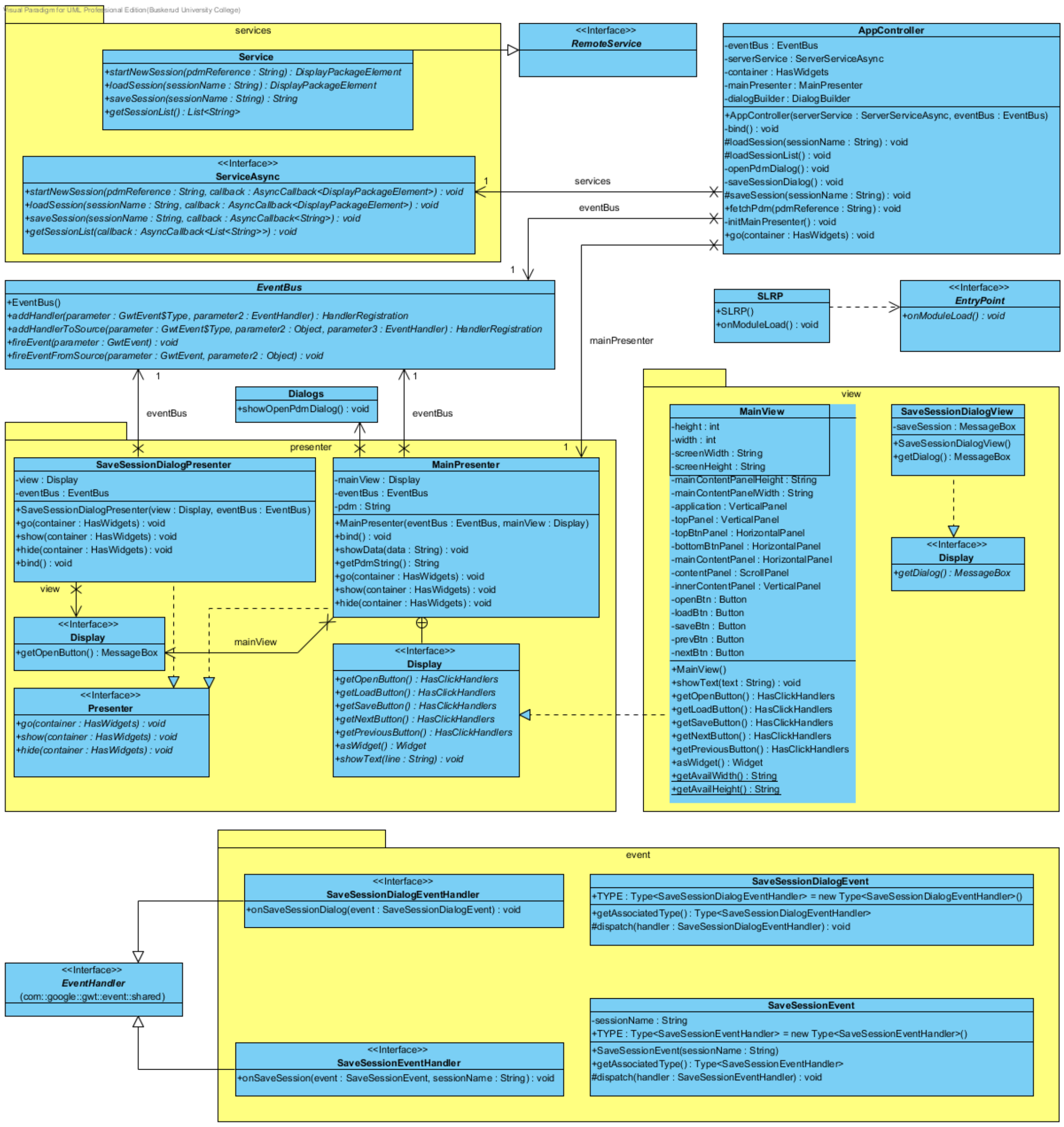

Save Button in the application is pushed
Input dialog appears. GXT Widget. User enters name for session
Via the eventBus the saveSession() in the service is called with the name as parameter, and an asynchronous call to the server is created.
The service implementation's saveSession() recieves the call, and sends a new call to the CoreController's saveSession method
StateTableController's saveSession() method is called and saves the name to this session in the state table. The boolean variable isSaved is set to true.
A result of the save is returned as a String the whole way back with the callback to the client. The client shows the result in a dialog window to the user.

FIGURE 9.1 SAVE SESSION

The client side of this feature is developed as shown in the class diagram in figure 9.2 below:



Visual Paradigm for UML Professional Edition(Buskerud University College)
services
Service
+startNewSession(pdmReference : String) : DisplayPackageElement
+loadSession(sessionName : String) : DisplayPackageElement
+saveSession(sessionName : String) : String
+getSessionList() : List<String>
<<Interface>>
ServiceAsync
+startNewSession(pdmReference : String, callback : AsyncCallback<DisplayPackageElement>) : void
+loadSession(sessionName : String, callback : AsyncCallback<DisplayPackageElement>) : void
+saveSession(sessionName : String, callback : AsyncCallback<String>) : void
+getSessionList(callback : AsyncCallback<List<String>>) : void
<<Interface>>
RemoteService
AppController
-eventBus : EventBus
-serverService : ServerServiceAsync
-container : HasWidgets
-mainPresenter : MainPresenter
-dialogBuilder : DialogBuilder
+AppController(serverService : ServerServiceAsync, eventBus : EventBus)
-bind() : void
#loadSession(sessionName : String) : void
#loadSessionList() : void
-openPdmDialog() : void
-saveSessionDialog() : void
#saveSession(sessionName : String) : void
+fetchPdm(pdmReference : String) : void
-initMainPresenter() : void
+go(container : HasWidgets) : void
services
eventBus
mainPresenter
1
1
EventBus
+EventBus()
+addHandler(parameter : GwtEvent$Type, parameter2 : EventHandler) : HandlerRegistration
+addHandlerToSource(parameter : GwtEvent$Type, parameter2 : Object, parameter3 : EventHandler) : HandlerRegistration
+fireEvent(parameter : GwtEvent) : void
+fireEventFromSource(parameter : GwtEvent, parameter2 : Object) : void
SLRP
+SLRP()
+onModuleLoad() : void
<<Interface>>
EntryPoint
+onModuleLoad() : void
Dialogs
+showOpenPdmDialog() : void
eventBus
eventBus
1
1
1
presenter
SaveSessionDialogPresenter
-view : Display
-eventBus : EventBus
+SaveSessionDialogPresenter(view : Display, eventBus : EventBus)
+go(container : HasWidgets) : void
+show(container : HasWidgets) : void
+hide(container : HasWidgets) : void
+bind() : void
MainPresenter
-mainView : Display
-eventBus : EventBus
-pdm : String
+MainPresenter(eventBus : EventBus, mainView : Display)
+bind() : void
+showData(data : String) : void
+getPdmString() : String
+go(container : HasWidgets) : void
+show(container : HasWidgets) : void
+hide(container : HasWidgets) : void
view
mainView
<<Interface>>
Display
+getOpenButton() : MessageBox
<<Interface>>
Presenter
+go(container : HasWidgets) : void
+show(container : HasWidgets) : void
+hide(container : HasWidgets) : void
<<Interface>>
Display
+getOpenButton() : HasClickHandlers
+getLoadButton() : HasClickHandlers
+getSaveButton() : HasClickHandlers
+getNextButton() : HasClickHandlers
+getPreviousButton() : HasClickHandlers
+asWidget() : Widget
+showText(line : String) : void
view
MainView
-height : int
-width : int
-screenWidth : String
-screenHeight : String
-mainContentPanelHeight : String
-mainContentPanelWidth : String
-application : VerticalPanel
-topPanel : VerticalPanel
-topBtnPanel : HorizontalPanel
-bottomBtnPanel : HorizontalPanel
-mainContentPanel : HorizontalPanel
-contentPanel : ScrollPanel
-innerContentPanel : VerticalPanel
-openBtn : Button
-loadBtn : Button
-saveBtn : Button
-prevBtn : Button
-nextBtn : Button
+MainView()
+showText(text : String) : void
+getOpenButton() : HasClickHandlers
+getLoadButton() : HasClickHandlers
+getSaveButton() : HasClickHandlers
+getNextButton() : HasClickHandlers
+getPreviousButton() : HasClickHandlers
+asWidget() : Widget
+getAvailWidth() : String
+getAvailHeight() : String
SaveSessionDialogView
-saveSession : MessageBox
+SaveSessionDialogView()
+getDialog() : MessageBox
<<Interface>>
Display
+getDialog() : MessageBox
event
<<Interface>>
SaveSessionDialogEventHandler
+onSaveSessionDialog(event : SaveSessionDialogEvent) : void
SaveSessionDialogEvent
+TYPE : Type<SaveSessionDialogEventHandler> = new Type<SaveSessionDialogEventHandler>()
+getAssociatedType() : Type<SaveSessionDialogEventHandler>
#dispatch(handler : SaveSessionDialogEventHandler) : void
<<Interface>>
EventHandler
(com::google::gwt::event::shared)
SaveSessionEvent
-sessionName : String
+TYPE : Type<SaveSessionEventHandler> = new Type<SaveSessionEventHandler>()
+SaveSessionEvent(sessionName : String)
+getAssociatedType() : Type<SaveSessionEventHandler>
#dispatch(handler : SaveSessionEventHandler) : void
<<Interface>>
SaveSessionEventHandler
+onSaveSession(event : SaveSessionEvent, sessionName : String) : void

FIGURE 9.213 CLIENT - SAVE SESSION

The client consists of an *EventBus* where events are fired when a user pushes a button in the application. In this diagram the events and event handlers for saving is shown, but it is exactly the same principles for loading a session and opening a new session. In this diagram the views and presenters required for saving a session are displayed, and as with the event and event handlers, it is the same principle for the other functionality. They all follow the MVP architecture design with an additional *AppController* as the client's controller (see chapter 4). The *Service* package contains the classes which send the asynchronous remote procedure calls (RPC) to the server. GWT handles this automatically, and sends a callback with the response.

The *SLRP* class is the entry point of this application.

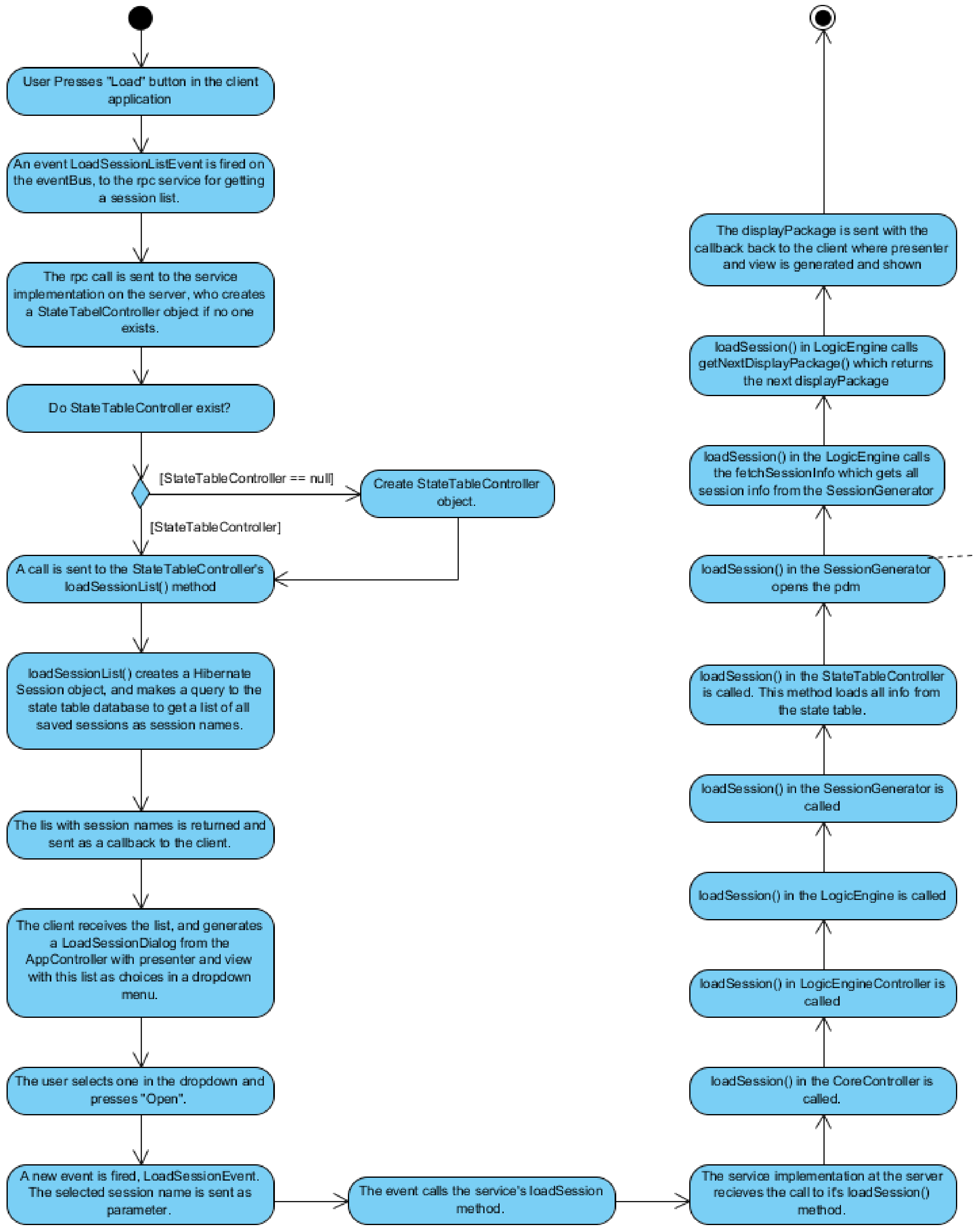
9.2 LOAD SESSION

If you want to continue a previously saved session you can load it by clicking the load button in the application. An event is then fired on the *EventBus*, calling the remote procedure call (RPC) service's *getSessionList()* method. An asynchronous call is now sent to the server's service implementation. From *CoreController,* a list is fetched from the state table, and returned with the callback to the client.

In the client, a dialogue is created with a presenter and a view, showing the list returned with the callback. The user is prompted to select one and press the "*Open*" button. A new event is now fired on the *EventBus* to the RPC service's *loadSession()* with the selected session name as an argument. This method sends an asynchronous callback to the server.

In the server's service implementation the call is received in the *loadSession()* method. This method forwards the call with *sessionName* and *stateTableController* as arguments through *CoreController* which saves the *stateTableController* object and next through *LogicEngineController* and *LogicEngine*. *LogicEngine* creates a *SessionGenerator* object which only lives in the scope of this method. Here the *loadSession()* method is run, and the session info is loaded from the state table and the process Data Module is loaded from CORENA IETP.

Back in the *LogicEngine* the process Data Module is sent to the *Renderer* for getting the next *DisplayPackage.* When this package is fetched, it is returned with the callback to the client, where view and presenter are being generated. The current step of the session is then displayed. In figure 9.2, the process is described.



User Presses "Load" button in the client application
An event LoadSessionListEvent is fired on the eventBus, to the rpc service for getting a session list.
The rpc call is sent to the service implementation on the server, who creates a StateTabelController object if no one exists.
Do StateTableController exist?
[StateTableController == null]
Create StateTableController object.
[StateTableController]
A call is sent to the StateTableController's loadSessionList() method
loadSessionList() creates a Hibernate Session object, and makes a query to the state table database to get a list of all saved sessions as session names.
The lis with session names is returned and sent as a callback to the client.
The client receives the list, and generates a LoadSessionDialog from the AppController with presenter and view with this list as choices in a dropdown menu.
The user selects one in the dropdown and presses "Open".
A new event is fired, LoadSessionEvent. The selected session name is sent as parameter.
The event calls the service's loadSession method.
The service implementation at the server recieves the call to it's loadSession() method.
loadSession() in the CoreController is called.
loadSession() in LogicEngineController is called
loadSession() in the LogicEngine is called
loadSession() in the SessionGenerator is called
loadSession() in the StateTableController is called. This method loads all info from the state table.
loadSession() in the SessionGenerator opens the pdm
loadSession() in the LogicEngine calls the fetchSessionInfo which gets all session info from the SessionGenerator
loadSession() in LogicEngine calls getNextDisplayPackage() which returns the next displayPackage
The displayPackage is sent with the callback back to the client where presenter and view is generated and shown

FIGURE 9.3 LOAD SESSION

10 DISPLAY PACKAGE HANDLING

10.1 WIDGET BUILDING AND <TYPE, BUILDER> MAPPING

For the transition between the display package data and the client-side widgets generated from it, we had a goal to avoid a constant “check type and then act” design (and code) mentality. However, because the widgets are client-specific and not something the model (which generates the display package) should know or care about, we weren’t able to implement a direct interface where you could say displayPackageElement.buildWidget(). *Some* type checking was required.

The answer we came up with was to “automate” the type checking by using a Map that maps between a Type and a Builder. The Type is an enumeration describing data elements implementing a specific interface (like DisplayPackageElement). The Builder is an interface (like WidgetBuilder) implemented by classes building a Widget or some other object used by the client. The mapping is handled by an abstract convenience wrapper class using generics; TypeController, which is extended by specific implementations (like WidgetController).

This allows us to directly build the client representation of a server data object, without having to check its type and call the correct method in the correct class.

10.2 DIALOGUES

Dialogues are one of the kinds of display package data, and are built using the TypeController<Type, Builder> structure/framework. The dialogue building implementation of this structure/framework consists of:

- The DisplayPackageElement interface and the type of its implementers (e.g., DialogGroup or Menu).
- The WidgetBuilder interface with method *buildWidget(DisplayPackageElement element)*
- The TypeController extension; WidgetController. WidgetController also implements the WidgetBuilder interface and uses the buildWidget-method to send the DisplayPackageElement to the correct WidgetBuilder.

The <dialog>-element itself is a DisplayPackageElement with a WidgetBuilder implementation, DialogBuilder. However, in order to keep the client as simple as possible, dialogues are never children, only top-level elements. This means they won’t be automatically built by other WidgetBuilder implementations, instead they’ll always be built directly by the display package handling software.

The TypeController<Type, Builder> structure/framework is especially useful for building dialogues, as the <dialog>-element consists of complex children that themselves may consist of complex children (and so on). By having these implement the DisplayPackageElement interface, a parent class may simply iterate over them, calling WidgetController's buildWidget-method on each of them, without having to do any manual type checking –and-treatment. This way, when a new DisplayPackageElement and its builder are implemented, the elements that can contain them do not need to be changed.

Figure 10.1 shows a class diagram of the dialogue building framework/structure, without the child elements.

Figure 10.2 shows an activity diagram that explains the process of building dialogues.

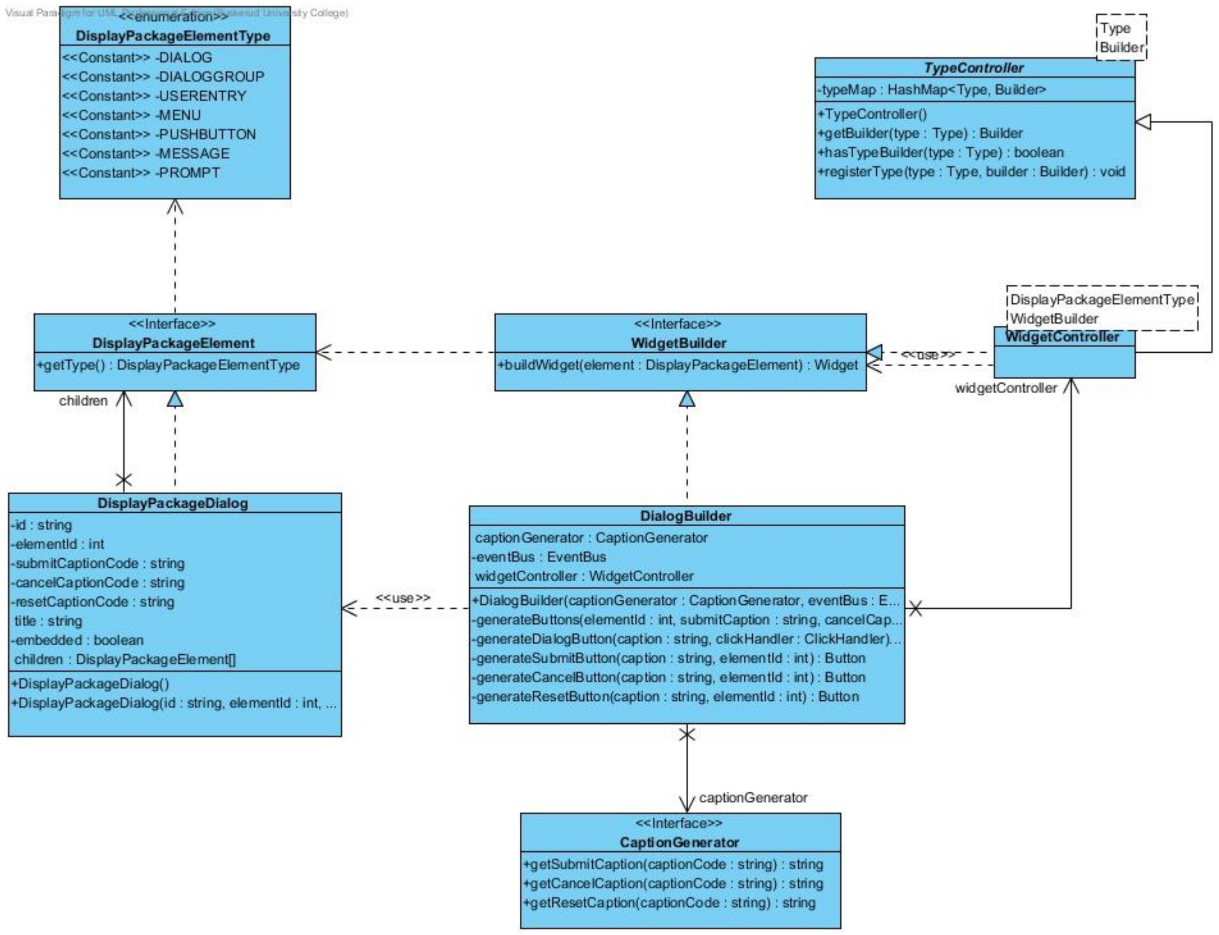

Visual Paradigm for UML ... University College)
<<enumeration>>
DisplayPackageElementType
<<Constant>> -DIALOG
<<Constant>> -DIALOGGROUP
<<Constant>> -USERENTRY
<<Constant>> -MENU
<<Constant>> -PUSHBUTTON
<<Constant>> -MESSAGE
<<Constant>> -PROMPT
Type
Builder
TypeController
-typeMap : HashMap<Type, Builder>
+TypeController()
+getBuilder(type : Type) : Builder
+hasTypeBuilder(type : Type) : boolean
+registerType(type : Type, builder : Builder) : void
DisplayPackageElementType
WidgetBuilder
WidgetController
<<Interface>>
DisplayPackageElement
+getType() : DisplayPackageElementType
<<Interface>>
WidgetBuilder
+buildWidget(element : DisplayPackageElement) : Widget
<<use>>
widgetController
children
DisplayPackageDialog
-id : string
-elementId : int
-submitCaptionCode : string
-cancelCaptionCode : string
-resetCaptionCode : string
title : string
-embedded : boolean
children : DisplayPackageElement[]
+DisplayPackageDialog()
+DisplayPackageDialog(id : string, elementId : int, ...
<<use>>
DialogBuilder
captionGenerator : CaptionGenerator
-eventBus : EventBus
widgetController : WidgetController
+DialogBuilder(captionGenerator : CaptionGenerator, eventBus : E...
-generateButtons(elementId : int, submitCaption : string, cancelCap...
-generateDialogButton(caption : string, clickHandler : ClickHandler)...
-generateSubmitButton(caption : string, elementId : int) : Button
-generateCancelButton(caption : string, elementId : int) : Button
-generateResetButton(caption : string, elementId : int) : Button
captionGenerator
<<Interface>>
CaptionGenerator
+getSubmitCaption(captionCode : string) : string
+getCancelCaption(captionCode : string) : string
+getResetCaption(captionCode : string) : string

FIGURE 10.1 CLASS DIAGRAM SHOWING THE DIALOGUE-BUILDING FRAMEWORK/STRUCTURE

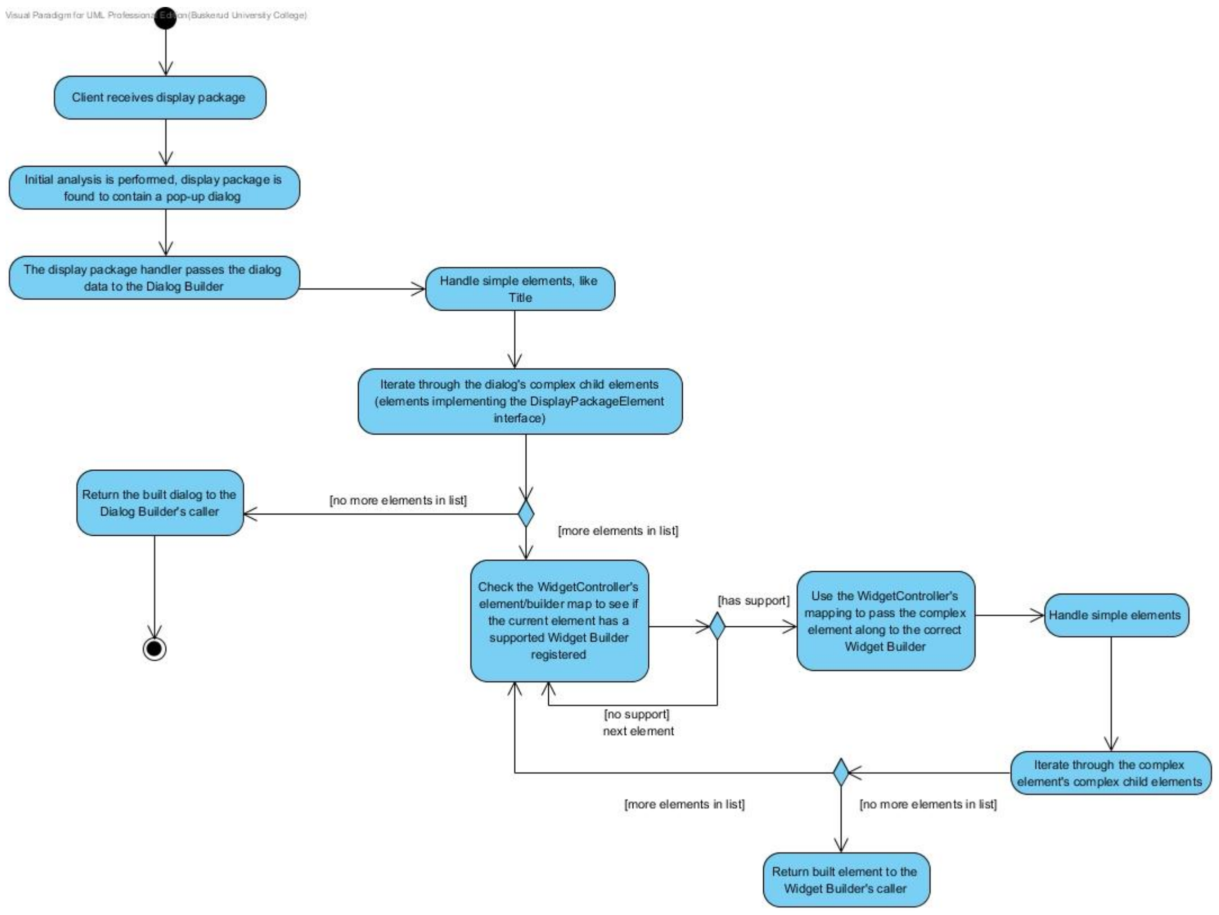

Visual Paradigm for UML Professional Edition(Buskerud University College)
Client receives display package
Initial analysis is performed, display package is found to contain a pop-up dialog
The display package handler passes the dialog data to the Dialog Builder
Handle simple elements, like Title
Iterate through the dialog's complex child elements (elements implementing the DisplayPackageElement interface)
[no more elements in list]
Return the built dialog to the Dialog Builder's caller
[more elements in list]
Check the WidgetController's element/builder map to see if the current element has a supported Widget Builder registered
[has support]
Use the WidgetController's mapping to pass the complex element along to the correct Widget Builder
Handle simple elements
[no support] next element
Iterate through the complex element's complex child elements
[more elements in list]
[no more elements in list]
Return built element to the Widget Builder's caller

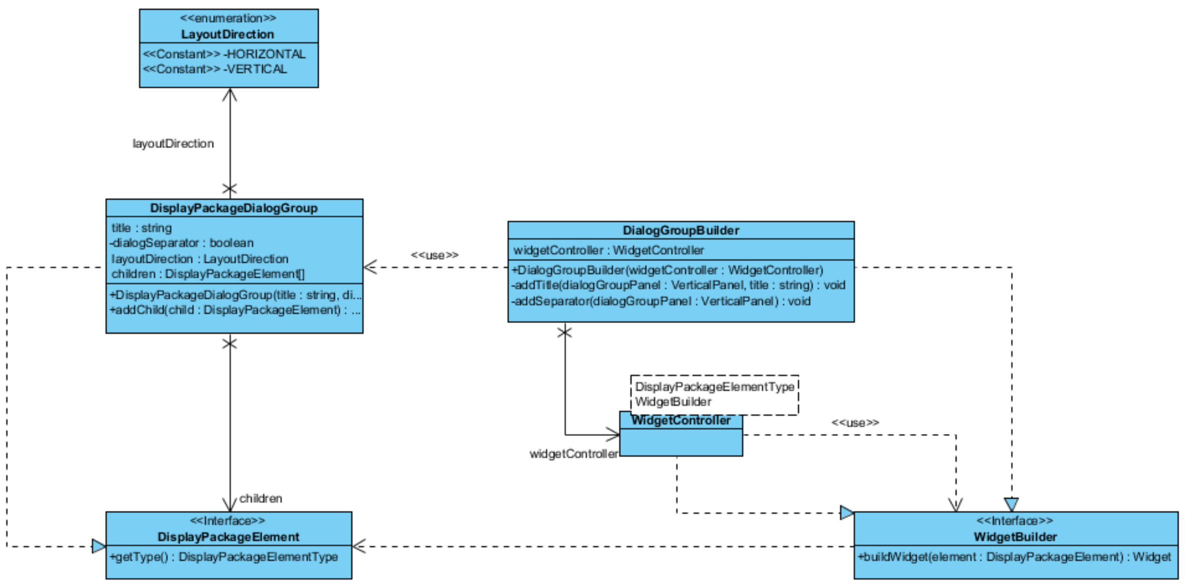
FIGURE 10.2 ACTIVITY DIAGRAM FOR DIALOGUE BUILDING

10.3 DIALOGUE CHILDREN

10.3.1 <dialogGroup>

A Dialogue Group is a layout element, used to toggle the direction its children are laid out in. If the parent of the dialogue group is vertically oriented, the dialogue group's layout is horizontal, and vice versa. It may contain the same child-elements as the <dialog>-element.

Figure 10.3 shows a class diagram for the building of a dialog group widget.



<<enumeration>>
LayoutDirection
<<Constant>> -HORIZONTAL
<<Constant>> -VERTICAL
layoutDirection
DisplayPackageDialogGroup
title : string
-dialogSeparator : boolean
layoutDirection : LayoutDirection
children : DisplayPackageElement[]
+DisplayPackageDialogGroup(title : string, di...
+addChild(child : DisplayPackageElement) : ...
<<use>>
DialogGroupBuilder
widgetController : WidgetController
+DialogGroupBuilder(widgetController : WidgetController)
-addTitle(dialogGroupPanel : VerticalPanel, title : string) : void
-addSeparator(dialogGroupPanel : VerticalPanel) : void
DisplayPackageElementType
WidgetBuilder
WidgetController
<<use>>
widgetController
children
<<Interface>>
DisplayPackageElement
+getType() : DisplayPackageElementType
<<Interface>>
WidgetBuilder
+buildWidget(element : DisplayPackageElement) : Widget

FIGURE 10.3 CLASS DIAGRAM FOR THE BUILDING OF DIALOG GROUPS

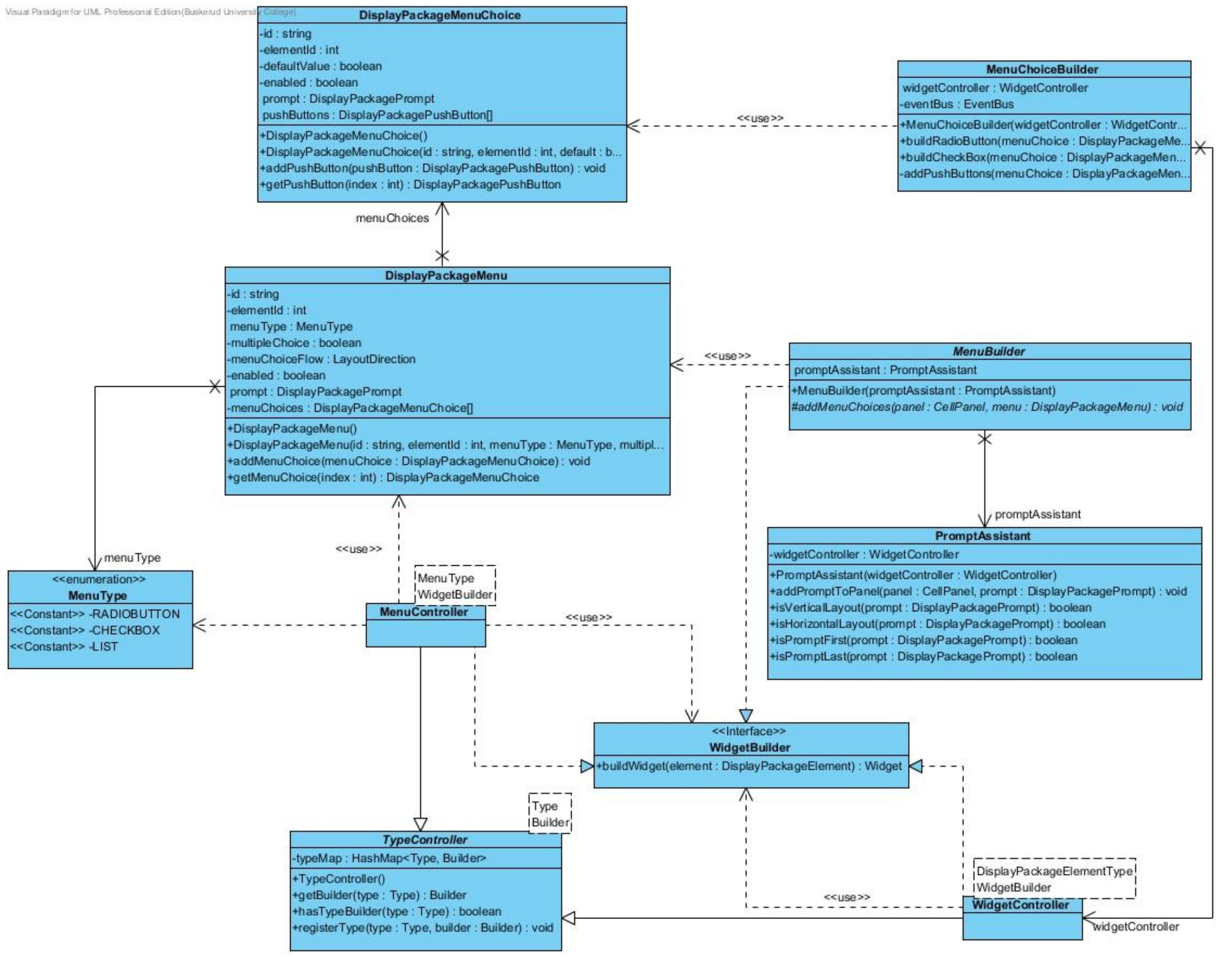
10.3.2 <menu>

A menu is one of the two elements used to obtain input from a user. It may be of three types; a radiobutton menu, a checkbox menu or a list menu. A list menu may be single- or multiple-choice. A menu is made up of one or more <menuChoice>-elements, which make up the entries in the menu.

The TypeController<Type, Builder> structure/framework is used to build menu widgets, with MenuType as the key and WidgetBuilder as the value. The common builder functionality of the menu types is located in an abstract superclass, MenuBuilder, which implements the WidgetBuilder interface. This class is extended by each of the builders, which implement an abstract method *addMenuChoices*.

Radiobutton- and checkbox-menus are made up of separate menu choice widgets which do not share a common parent. The builders for these menus use an additional class to build the menu choice widgets, MenuChoiceBuilder. The list menu, on the other hand, is one single "parent widget", whose children are simply list-entries with labels and values.

Figure 10.4 shows a class diagram for the building of menus.



Visual Paradigm for UML Professional Edition(Buskerud University College)
DisplayPackageMenuChoice
-id : string
-elementId : int
-defaultValue : boolean
-enabled : boolean
prompt : DisplayPackagePrompt
pushButtons : DisplayPackagePushButton[]
+DisplayPackageMenuChoice()
+DisplayPackageMenuChoice(id : string, elementId : int, default : b...
+addPushButton(pushButton : DisplayPackagePushButton) : void
+getPushButton(index : int) : DisplayPackagePushButton
MenuChoiceBuilder
widgetController : WidgetController
-eventBus : EventBus
+MenuChoiceBuilder(widgetController : WidgetContr...
+buildRadioButton(menuChoice : DisplayPackageMe...
+buildCheckBox(menuChoice : DisplayPackageMen...
-addPushButtons(menuChoice : DisplayPackageMen...
<<use>>
menuChoices
DisplayPackageMenu
-id : string
-elementId : int
menuType : MenuType
-multipleChoice : boolean
-menuChoiceFlow : LayoutDirection
-enabled : boolean
prompt : DisplayPackagePrompt
-menuChoices : DisplayPackageMenuChoice[]
+DisplayPackageMenu()
+DisplayPackageMenu(id : string, elementId : int, menuType : MenuType, multipl...
+addMenuChoice(menuChoice : DisplayPackageMenuChoice) : void
+getMenuChoice(index : int) : DisplayPackageMenuChoice
<<use>>
MenuBuilder
promptAssistant : PromptAssistant
+MenuBuilder(promptAssistant : PromptAssistant)
#addMenuChoices(panel : CellPanel, menu : DisplayPackageMenu) : void
promptAssistant
PromptAssistant
-widgetController : WidgetController
+PromptAssistant(widgetController : WidgetController)
+addPromptToPanel(panel : CellPanel, prompt : DisplayPackagePrompt) : void
+isVerticalLayout(prompt : DisplayPackagePrompt) : boolean
+isHorizontalLayout(prompt : DisplayPackagePrompt) : boolean
+isPromptFirst(prompt : DisplayPackagePrompt) : boolean
+isPromptLast(prompt : DisplayPackagePrompt) : boolean
menuType
<<enumeration>>
MenuType
<<Constant>> -RADIOBUTTON
<<Constant>> -CHECKBOX
<<Constant>> -LIST
<<use>>
MenuType
WidgetBuilder
MenuController
<<use>>
<<Interface>>
WidgetBuilder
+buildWidget(element : DisplayPackageElement) : Widget
Type
Builder
TypeController
-typeMap : HashMap<Type, Builder>
+TypeController()
+getBuilder(type : Type) : Builder
+hasTypeBuilder(type : Type) : boolean
+registerType(type : Type, builder : Builder) : void
DisplayPackageElementType
WidgetBuilder
<<use>>
WidgetController
widgetController

FIGURE 10.4 CLASS DIAGRAM FOR THE BUILDING OF MENUS

10.3.3 <userEntry>

A user entry is the other element used to obtain input from a user and is represented by a text input field. The user entry has a reference to a variable value where the data in the text field is saved. The variable value is sent to the server via a SubmitEvent when it is updated.

Figure 10.5 shows a class diagram for the building of user entries.

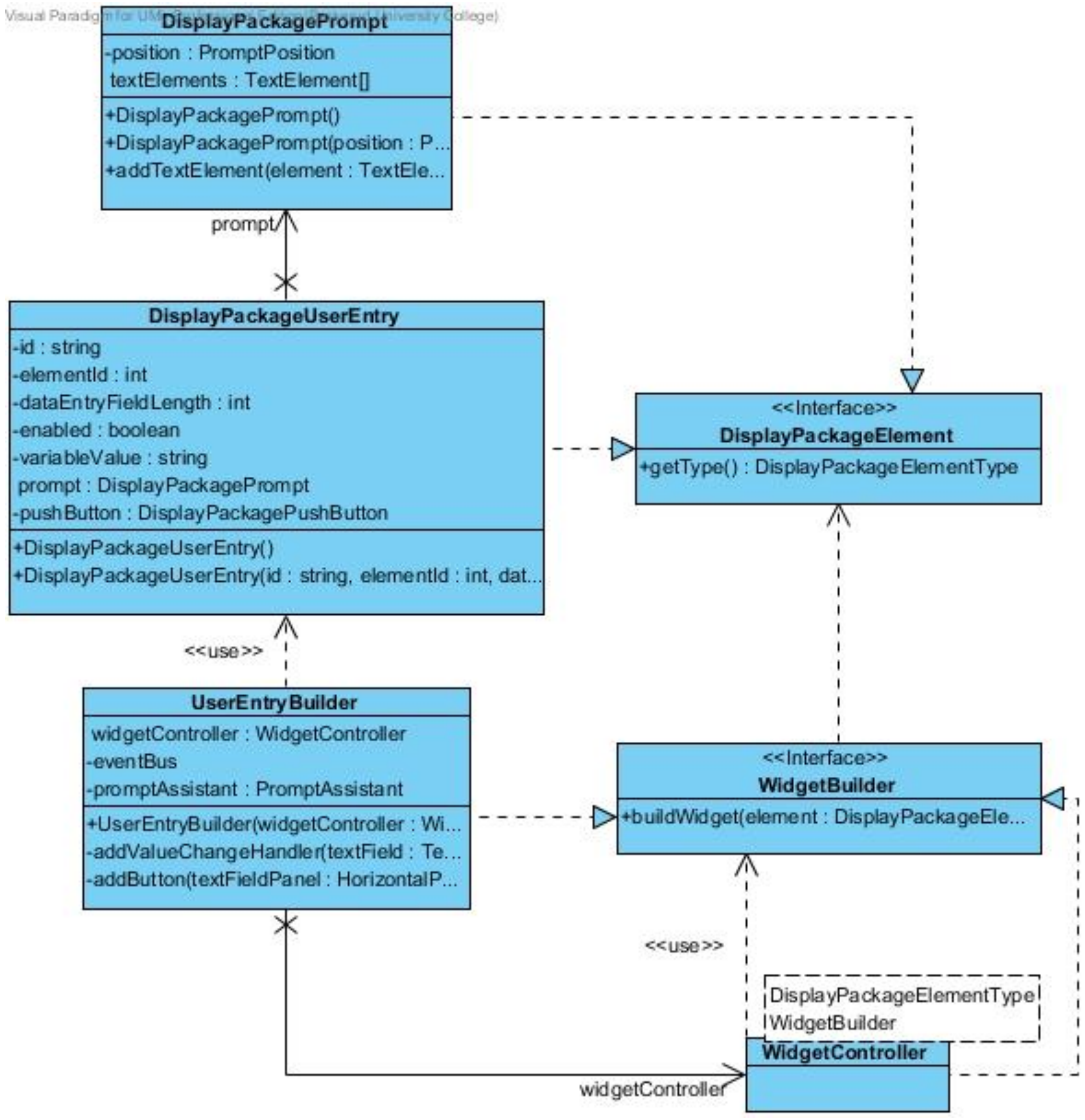

Visual Paradigm for UML Standard Edition(Ameo University College)
DisplayPackagePrompt
-position : PromptPosition
textElements : TextElement[]
+DisplayPackagePrompt()
+DisplayPackagePrompt(position : P...
+addTextElement(element : TextEle...
prompt
DisplayPackageUserEntry
-id : string
-elementId : int
-dataEntryFieldLength : int
-enabled : boolean
-variableValue : string
prompt : DisplayPackagePrompt
-pushButton : DisplayPackagePushButton
+DisplayPackageUserEntry()
+DisplayPackageUserEntry(id : string, elementId : int, dat...
<<Interface>>
DisplayPackageElement
+getType() : DisplayPackageElementType
<<use>>
UserEntryBuilder
widgetController : WidgetController
-eventBus
-promptAssistant : PromptAssistant
+UserEntryBuilder(widgetController : Wi...
-addValueChangeHandler(textField : Te...
-addButton(textFieldPanel : HorizontalP...
<<Interface>>
WidgetBuilder
+buildWidget(element : DisplayPackageEle...
<<use>>
DisplayPackageElementType
WidgetBuilder
WidgetController
widgetController

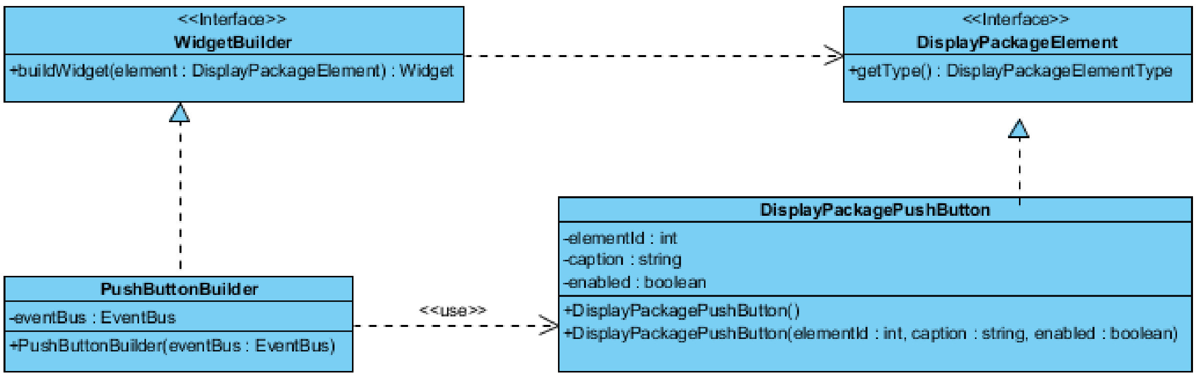
FIGURE 10.5 CLASS DIAGRAM FOR THE BUILDING OF USER ENTRIES

10.3.4 <pushButton>

A push button is used to launch a so-called associated action; a new dialogue or an external application. This is something that occurs on the server, so the display package / client version is of limited complexity.

The buttons used for submitting/cancelling/resetting a dialogue are **not** <pushButton>-elements.

Figure 10.6 shows a class diagram for the building of push buttons.



<<Interface>>
WidgetBuilder
+buildWidget(element : DisplayPackageElement) : Widget
<<Interface>>
DisplayPackageElement
+getType() : DisplayPackageElementType
DisplayPackagePushButton
-elementId : int
-caption : string
-enabled : boolean
+DisplayPackagePushButton()
+DisplayPackagePushButton(elementId : int, caption : string, enabled : boolean)
PushButtonBuilder
-eventBus : EventBus
+PushButtonBuilder(eventBus : EventBus)
<<use>>

FIGURE 10.6 CLASS DIAGRAM FOR BUILDING PUSH BUTTONS

10.4 PROMPTS AND MESSAGES

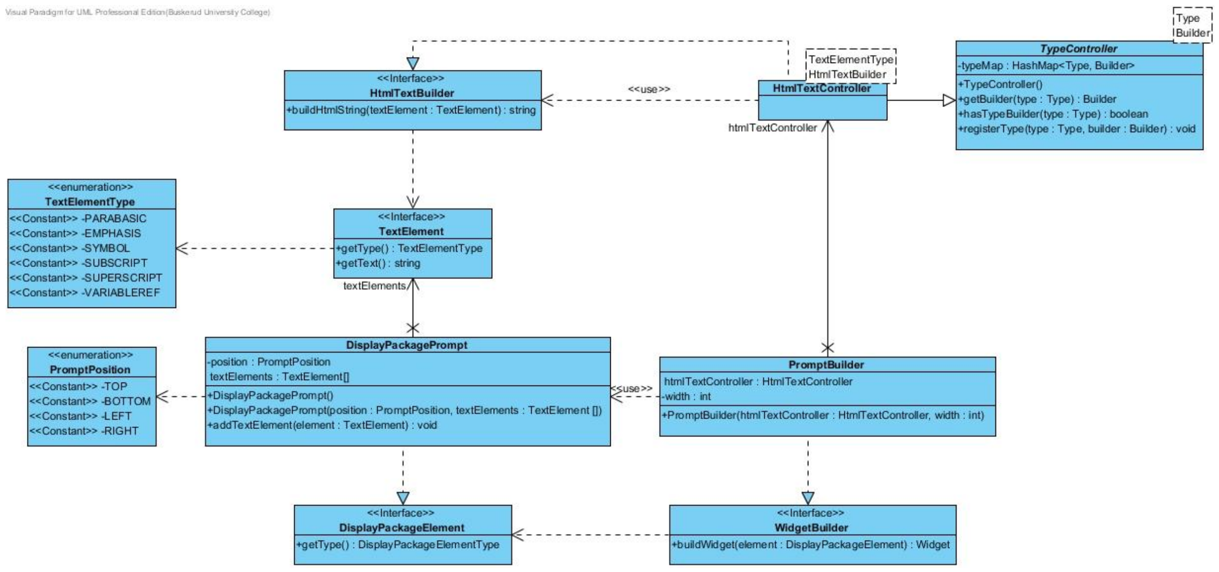
The S1000D specification uses the <prompt>-element to present text information to the user. A prompt is made up of several different text elements, and can be positioned relative to its parent (which is always another display package element). The ability to position the prompt affects the layout of the parent and the order of its children. This means the builders of certain parent elements also has to treat the <prompt> element when building their widgets.

The text elements in a prompt are built individually using the TypeController<Type, Builder> structure/framework. An interface, TextElement is implemented by all text element data classes and their type is used as the key in the map. An HtmlTextBuilder interface is implemented by the text element builders and the value in the map.

A <message>-element is a (relatively) simple element used to display messages to the user. It may be in the form of a pop-up dialogue, or embedded in some other element.

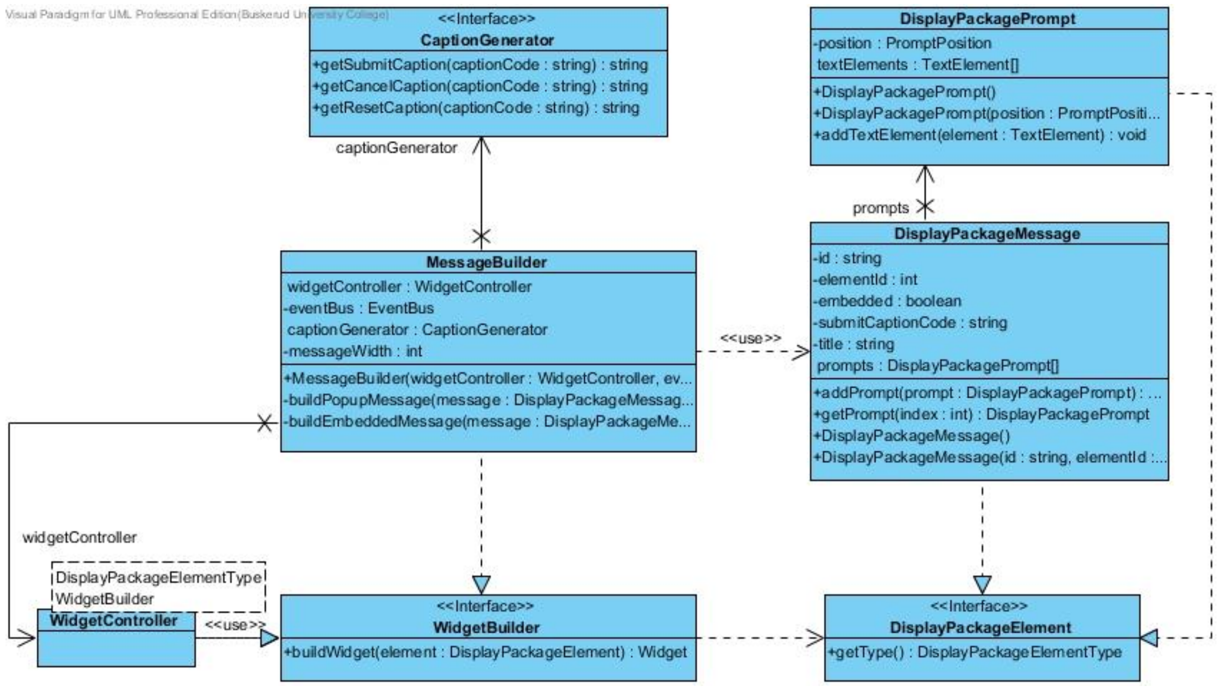
Figure 10.7 shows the class diagram for building prompts, including text elements.

Figure 10.8 shows the class diagram for building messages.



Visual Paradigm for UML Professional Edition(Buskerud University College)
<<Interface>>
HtmlTextBuilder
+buildHtmlString(textElement : TextElement) : string
<<use>>
TextElementType
HtmlTextBuilder
HtmlTextController
htmlTextController
Type
Builder
TypeController
-typeMap : HashMap<Type, Builder>
+TypeController()
+getBuilder(type : Type) : Builder
+hasTypeBuilder(type : Type) : boolean
+registerType(type : Type, builder : Builder) : void
<<enumeration>>
TextElementType
<<Constant>> -PARABASIC
<<Constant>> -EMPHASIS
<<Constant>> -SYMBOL
<<Constant>> -SUBSCRIPT
<<Constant>> -SUPERSCRIPT
<<Constant>> -VARIABLEREF
<<Interface>>
TextElement
+getType() : TextElementType
+getText() : string
textElements
<<enumeration>>
PromptPosition
<<Constant>> -TOP
<<Constant>> -BOTTOM
<<Constant>> -LEFT
<<Constant>> -RIGHT
DisplayPackagePrompt
-position : PromptPosition
textElements : TextElement[]
+DisplayPackagePrompt()
+DisplayPackagePrompt(position : PromptPosition, textElements : TextElement [])
+addTextElement(element : TextElement) : void
<<use>>
PromptBuilder
htmlTextController : HtmlTextController
-width : int
+PromptBuilder(htmlTextController : HtmlTextController, width : int)
<<Interface>>
DisplayPackageElement
+getType() : DisplayPackageElementType
<<Interface>>
WidgetBuilder
+buildWidget(element : DisplayPackageElement) : Widget

FIGURE 10.7 CLASS DIAGRAM SHOWING THE BUILDING OF PROMPTS



Visual Paradigm for UML Professional Edition(Buskerud University College)
<<Interface>>
CaptionGenerator
+getSubmitCaption(captionCode : string) : string
+getCancelCaption(captionCode : string) : string
+getResetCaption(captionCode : string) : string
captionGenerator
DisplayPackagePrompt
-position : PromptPosition
textElements : TextElement[]
+DisplayPackagePrompt()
+DisplayPackagePrompt(position : PromptPositi...
+addTextElement(element : TextElement) : void
prompts
MessageBuilder
widgetController : WidgetController
-eventBus : EventBus
captionGenerator : CaptionGenerator
-messageWidth : int
+MessageBuilder(widgetController : WidgetController, ev...
-buildPopupMessage(message : DisplayPackageMessag...
-buildEmbeddedMessage(message : DisplayPackageMe...
<<use>>
DisplayPackageMessage
-id : string
-elementId : int
-embedded : boolean
-submitCaptionCode : string
-title : string
prompts : DisplayPackagePrompt[]
+addPrompt(prompt : DisplayPackagePrompt) : ...
+getPrompt(index : int) : DisplayPackagePrompt
+DisplayPackageMessage()
+DisplayPackageMessage(id : string, elementId :...
widgetController
DisplayPackageElementType
WidgetBuilder
WidgetController
<<use>>
<<Interface>>
WidgetBuilder
+buildWidget(element : DisplayPackageElement) : Widget
<<Interface>>
DisplayPackageElement
+getType() : DisplayPackageElementType

FIGURE 10.8 CLASS DIAGRAM FOR BUILDING MESSAGES

10.5 SUBMITTING INPUT

The way we submit user input (which originates in the client) to the model/server for evaluation is an important part in our quest to keep the client simple and free of model logic. The client should know as little as possible about what it is sending, but it must know *something* in order to send it.

We have defined three submission scenarios in addition to the one defined in the specification.

- *Associated action*. Clicking a widget whose display package element has an associated action attached to it submits the element id to the model, which executes the action. The associated action may be the only response to a submission, or one of several.
- *Menu*. Clicking a menu entry submits the element id(s) to the model which, depending on the menu type, overwrites, adds or removes the element id(s) from the menu state.
- *User Entry*. A user entry sets a variable value which, together with the element id, is submitted when the user entry text field is updated and stored in the model/server.
- *Dialogues*. Clicking the "Submit" button in a dialogue (or a message) submits the element id to the model/server, where all the assertions, validations and other expressions defined by its child elements are evaluated.

10.6 WARNINGS AND CAUTIONS

Warnings and Cautions, with the elements <warning>, <caution> and <warningAndCautionPara> are supported as display package- and text elements. Building them on the client was complicated by the <warningAndCautionPara> element differing from other text elements. Instead of separating pure text into its own element, like <paraBasic> in <prompt>, <warningAndCautionPara> puts its pure text directly in the element (as a value) – similar to HTML.

To make this work with the framework already in place, <warningAndCautionPara> is represented as both a text element containing pure text, and a Display Package element holding the list of child elements (including the text element version of itself).

Furthermore, a lot of functionality was shared between <warning> and <caution>, and has been abstracted into superclasses. Both the data classes for <warning> and <caution>, as well as their builders, are extending a class containing common functionality and/or data.

Figure 10.9 shows a class diagram for warnings and cautions, minus the text elements.

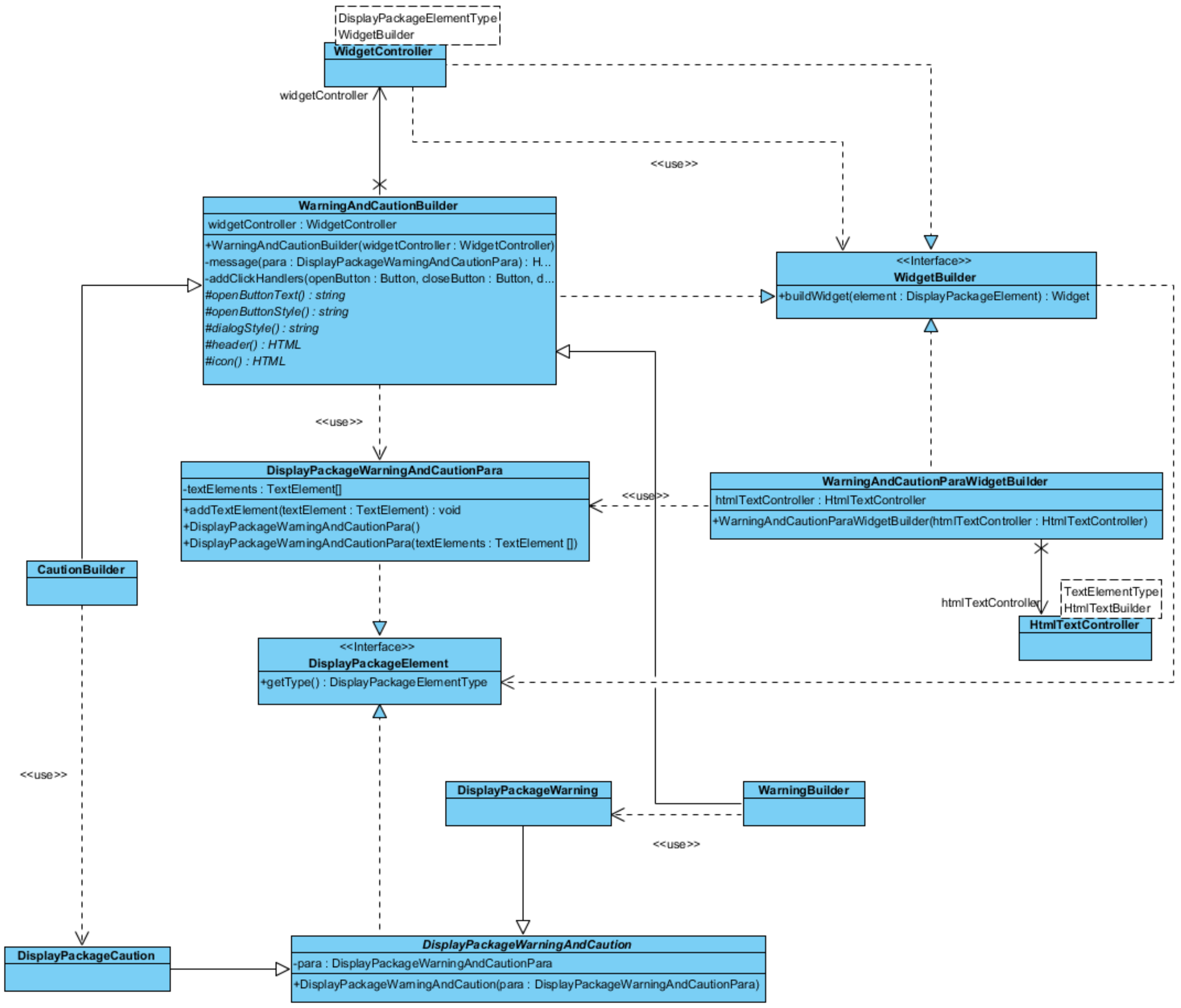

DisplayPackageElementType
WidgetBuilder
WidgetController
widgetController
<<use>>
WarningAndCautionBuilder
widgetController : WidgetController
+WarningAndCautionBuilder(widgetController : WidgetController)
-message(para : DisplayPackageWarningAndCautionPara) : H...
-addClickHandlers(openButton : Button, closeButton : Button, d...
#openButtonText() : string
#openButtonStyle() : string
#dialogStyle() : string
#header() : HTML
#icon() : HTML
<<Interface>>
WidgetBuilder
+buildWidget(element : DisplayPackageElement) : Widget
<<use>>
DisplayPackageWarningAndCautionPara
-textElements : TextElement[]
+addTextElement(textElement : TextElement) : void
+DisplayPackageWarningAndCautionPara()
+DisplayPackageWarningAndCautionPara(textElements : TextElement [])
<<use>>
WarningAndCautionParaWidgetBuilder
htmlTextController : HtmlTextController
+WarningAndCautionParaWidgetBuilder(htmlTextController : HtmlTextController)
CautionBuilder
htmlTextController
TextElementType
HtmlTextBuilder
HtmlTextController
<<Interface>>
DisplayPackageElement
+getType() : DisplayPackageElementType
<<use>>
DisplayPackageWarning
WarningBuilder
<<use>>
DisplayPackageCaution
DisplayPackageWarningAndCaution
-para : DisplayPackageWarningAndCautionPara
+DisplayPackageWarningAndCaution(para : DisplayPackageWarningAndCautionPara)

FIGURE 10.9 CLASS DIAGRAM FOR WARNINGS AND CAUTIONS

11 EXPRESSIONS

11.1 BACKGROUND

As mentioned in the Analysis document, expressions contain a variety of operations, functions and values. Supporting all of these *out of the box* was deemed unneeded and impractical. Instead, we've focused on coming up with a design of the surrounding framework that allowed for a dynamic approach to adding support for new operations, functions, or values.

The primary source of challenge when designing and developing support for expressions has been the dilemma of *type safety* versus a goal of keeping everything dynamic and generic. Or more directly, we wanted to do things as generic as possible, but we didn't want the application to get blown up by a run time exception because of some mistaken assumption or spelling error.

Type safety could easily be obtained by checking the content of the expression compared to the possible child elements (operator, function, value, variable reference), then, if applicable, checking the operator for all possible operations, or the function for every type of action – and finally, checking for the specified values' types before passing them to the correct method. This, however, would lead to several dozen if-else sentences in at least three layers of nested statements, which is the very opposite of *dynamic*.

The amount of checking required to "easily" obtain type safety is also why finding a dynamic and generic solution has been such a challenge. The specification defines operators with different operations and functions with different actions – which in turn have different return types, different parameter types and different amounts of parameters. Several of the operations and actions also have multiple signatures, and in one case, more than one return type within the same signature. A *one size fits all* solution was hardly apparent.

11.2 SOLUTION

The solution came through Java's *generics*, which is not something we've had any kind of significant experience with prior to this project. Thankfully, we have had some lectures on the C++ equivalent; templates, which helped with our basic approach and understanding.

Through several rounds of experimentation and prototyping with generics, we believe we've found a solution that allows us to avoid type checking whenever we use a value, and at the same time maintain an acceptable level of type safety.

Because an expression only contains one operation/function/value/variable-reference, we can compress the values of these elements into one single field in the Expression-class during the data binding process. This field is named *evaluation*. It contains the name of the operation or action as a String, and has an interface counter-part to which it is mapped.

Using a Map, we map between this field plus the corresponding method's signature (that is, the types of its parameter(s), and the number of parameter(s)) – and an implementation of the generic Evaluation<R, P> interface which defines the return and parameter types, and a *getResult*-method. The map is wrapped in an EvaluationController for convenience and to separate the implementation from the class' clients.

In effect, this gives us a kind of *function pointer*, where the Expression class which is being evaluated simply calls the EvaluationController's *getResult*-method with its evaluation-signature (after first checking that it is supported), which is then passed on to the correct Evaluation-implementation. These implementations are only called from EvaluationController, and only if they are registered with the current signature. Thus, unless the mapping itself is erroneous, no wrongful casting should occur, and type safety is maintained.

Adding support for new operations or functions is done by creating a class implementing the Evaluation-interface, and registering it, together with its signature, in EvaluationController.

As a side note, the EvaluationController is currently a static class (or, it has all-static fields and methods). This is because of some data binding limitations, and not a design necessity beyond this limitation.

Figure 11.1 shows a class diagram for the expression evaluation, without any Evaluation-implementations.

Figure 11.2 shows an activity diagram that describes the expression evaluation process.

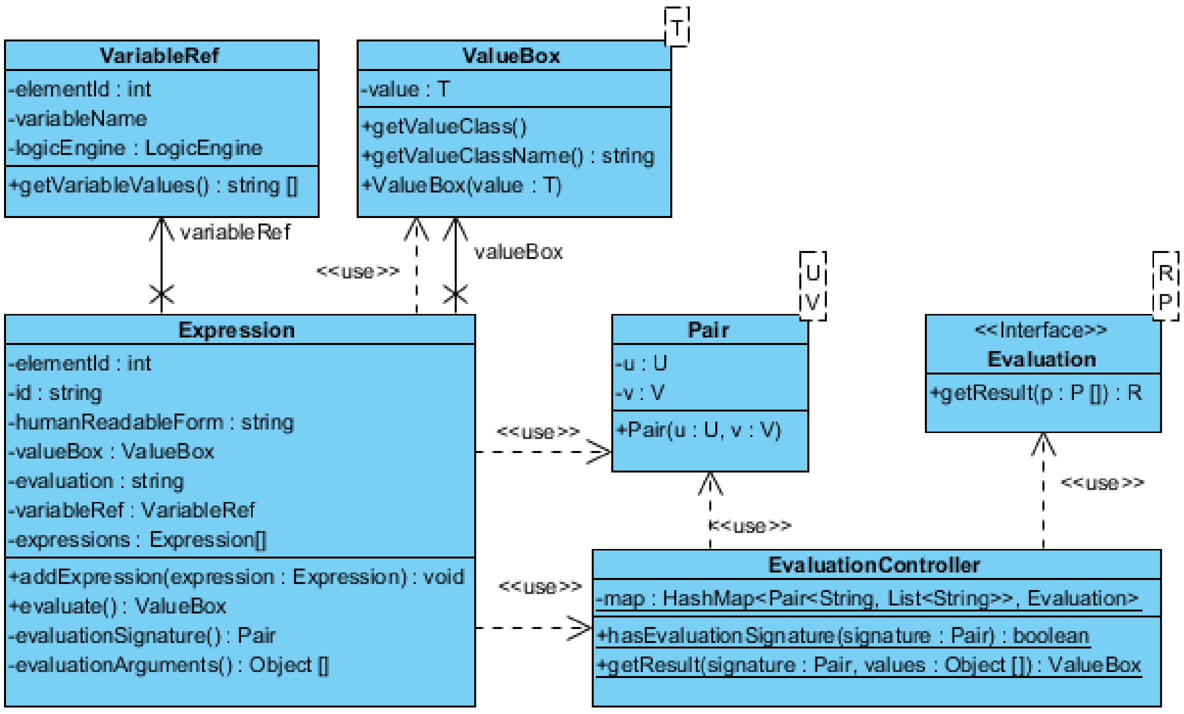

VariableRef
-elementId : int
-variableName
-logicEngine : LogicEngine
+getVariableValues() : string []
ValueBox
T
-value : T
+getValueClass()
+getValueClassName() : string
+ValueBox(value : T)
variableRef
<<use>>
valueBox
U
V
R
P
Expression
-elementId : int
-id : string
-humanReadableForm : string
-valueBox : ValueBox
-evaluation : string
-variableRef : VariableRef
-expressions : Expression[]
+addExpression(expression : Expression) : void
+evaluate() : ValueBox
-evaluationSignature() : Pair
-evaluationArguments() : Object []
Pair
-u : U
-v : V
+Pair(u : U, v : V)
<<Interface>>
Evaluation
+getResult(p : P []) : R
<<use>>
<<use>>
<<use>>
<<use>>
EvaluationController
-map : HashMap<Pair<String, List<String>>, Evaluation>
+hasEvaluationSignature(signature : Pair) : boolean
+getResult(signature : Pair, values : Object []) : ValueBox

FIGURE 11.1 CLASS DIAGRAM FOR EXPRESSION EVALUATION

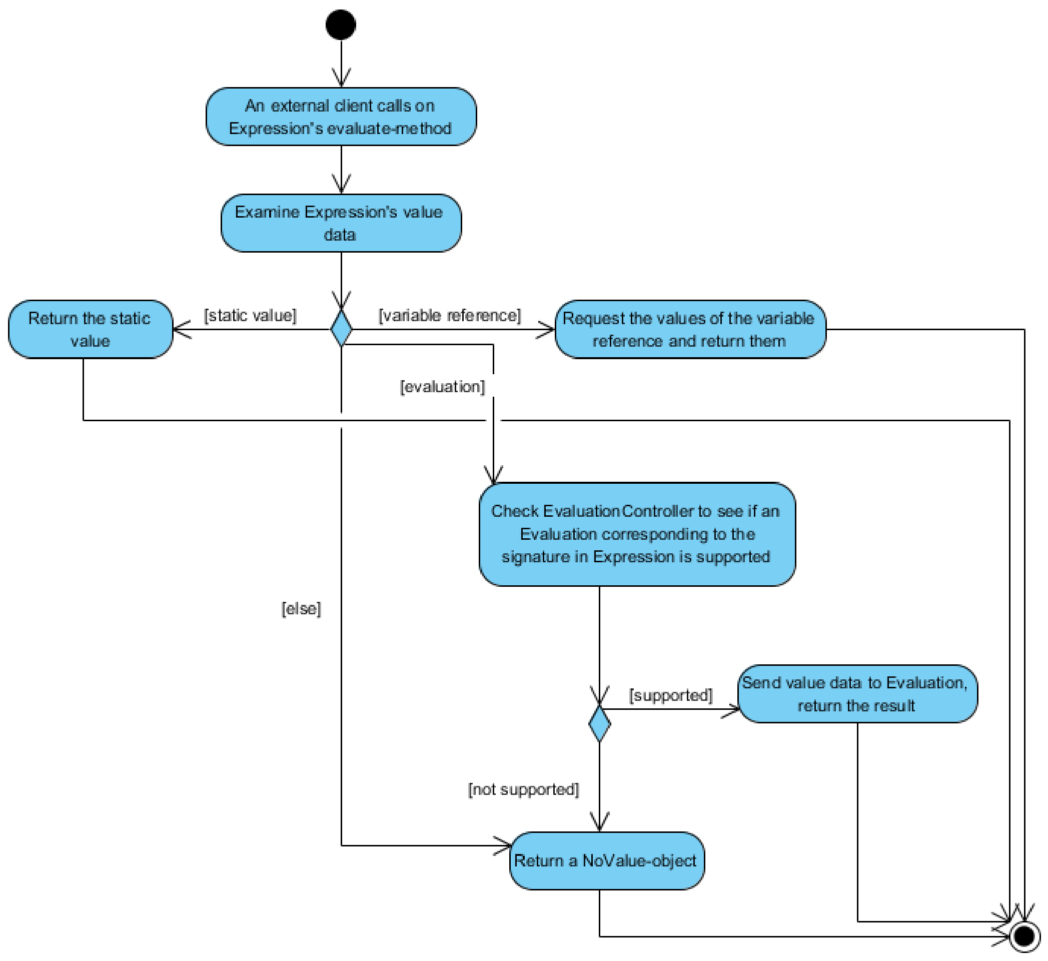

An external client calls on Expression's evaluate-method
Examine Expression's value data
[static value]
Return the static value
[variable reference]
Request the values of the variable reference and return them
[evaluation]
Check EvaluationController to see if an Evaluation corresponding to the signature in Expression is supported
[else]
[supported]
Send value data to Evaluation, return the result
[not supported]
Return a NoValue-object

FIGURE 11.2 ACTIVITY DIAGRAM FOR EVALUATING EXPRESSIONS

12 REFERENCE(S)

1. *International Specification for technical publications utilizing a common source database S1000D*, Technical Publications Specification Maintenance Group (TPSMG), 01-Aug-2008 (Issue 4.0)

2. *Product Backlog,* Ameo (last updated 07-Mar-2011), 2010 HiBu

3. *GWT with MVP: A Case Study*, David Chandler (published Mar-2010), DevNexus, http://www.slideshare.net/turbomanage/gwt-mvp-case-study (last visited 07-Mar-2011)

4. *Should I Use SAX or DOM*, Nazmul Idris, (published May 23rd 1999), http://developerlife.com/tutorials/?p=28 (last visited 08-Mar-2011)

5. *Large scale application development and MVP*, Chris Ramsdale, http://code.google.com/webtoolkit/articles/mvp-architecture.html (last visited 16-Mar-2011)

6. *Java DB,* Oracle and Sun, http://www.oracle.com/technetwork/java/javadb/overview/index.html (last visited 25-Mar-2011)

7. *Relational Persistence for Java and .NET,* JBoss Community HiberNate, http://www.hibernate.org/ (last visited 27-Apr-2011)

8. *Hibernate (Java),* Wikipedia (En), http://en.wikipedia.org/wiki/Hibernate_%28Java%29 (last visited 27-Apr-2011)

SCRUM PROCEDURES

HIBU STUDENT PROJECT 2011
CORENA S1000D 4.0 PROCESS DATA MODULE RENDERER

____
Arild Oldervoll

____
Marius Haraldseth

____
Eirik André Eidså

____
Olav Brandt

TABLE OF CONTENTS

REVISION HISTORY			
Version Number	**Date**	**Changes**	**Assigned**
1	04-Jan-2011	First version	AOL

1 INTRODUCTION

Our assignment to develop a logic engine for support execution of S1000D process Data Modules version 4.0, is given by CORENA Norge A/S. This logic engine can take input from users and/or external applications, such as sensors, to decide its next step in a flow of process Data Modules. It can branch and loop through the selected process Data Module. The logic engine will be an interpreter for process Data Modules. We will make it a stand-alone logic engine in a web based application running on Jetty/Tomcat. The user of the application should be able to move back and forth through the steps using previous and next buttons. The user should be able to save the session to continue at a later time. The logic engine should support the International S1000D Specification Issue 4.0.

1.1 INTENTION OF THIS DOCUMENT

This document will elaborate on what Scrum is and how we use it in our project. Scrum is an agile software development methodology that was introduced in the 90's and has become very popular. Scrum has a strong focus on the customers and developers, and the customers will be strongly involved in the project throughout its lifecycle. The team of developers are also the "project manager" and they are doing most of the planning of the project. The project is divided into iterations called Sprints.

1.2 AUTHOR(S)

Arild Oldervoll (AOL)

1.3 ASSIGNED

Arild Oldervoll (AOL)

2 WHY SCRUM

Scrum has several advantages. Following is a description of some of the advantages that are most important to us in Ameo.

2.1 THE TEAM IS THE PROJECT LEADER

The amount of work to be done in a Sprint and what work to do is discussed between the team and the Product Owner. The team then commits itself to the Sprint goal and is equally responsible for achieving this goal before the end of the Sprint. The team can use any means necessary to achieve the desired outcome. The ScrumMaster will remove any impediments on the team and make sure that they can work uninterrupted during the Sprint. This works well because it is the team that is working on the problem and thus has the best knowledge about the complexity. Therefore they are the best resource for Sprint planning. In addition, overhead time on planning is reduced.

2.2 THE MOST IMPORTANT TASKS ARE DONE FIRST

Many project development methodologies will either attack the most complex problems first, or the easiest problems first. In contrast, Scrum starts with the User Stories that are most important for the customers, represented by the Product Owner. Even *within* a Sprint the most important work is done first, as opposed to analysing and designing everything for an iteration at the beginning of the iteration.

2.3 SCRUM IS TRULY AGILE

Scrum is a very agile process. It is only the current and maybe the next Sprint that is planned to any detail. The priority and requested features from the customers are expected to and will change during the project, so no time spent on project planning is lost *when* the requirements are changed. The team can also add and remove tasks during the Sprint as they see it necessary, as long as the main goal of the Sprint is unchanged. The assigned tasks can change on a day-to-day basis without planning effort being lost.

2.4 ROLES ARE NOT DEFINED

With Scrum there are only three active roles in a project: Product Owner, ScrumMaster and the Team. This implies that everyone on the team are considered equal, and it is up to the team to assign the available resources to different tasks to reach the Sprint goals. This means that for example a tester can be used on analysis if required.

3 SCRUM PROJECT PLANNING IN GENERAL

Compared to many traditional project methodologies like waterfall and RUP, Scrum has some differences, but is also similar in some aspects. The biggest difference is in which phase of the project the detailed planning is done, although the plans should contain much of the same level of detail at the completion of the project.

3.1 SCRUM PLANNING DURING THE PRESTUDY

What resembles a traditional project plan and requirements specification is in Scrum called the *Product Backlog*. The Product Backlog is a list containing all the *User Stories* known at the current time (see *User Stories*, section 3.2). Each User Story has a priority that the Product Owner has determined and an effort estimate (see *Calculation of Time Estimates in Scrum*, chapter 5) that the team has determined. The Product Backlog is normally ordered with the highest priority item on the top. When development starts, the team will pick User Stories from the top of the Product Backlog. Therefore the User Stories on the top of the Product Backlog are more detailed and have a more accurate estimate.

3.2 SPRINT PLANNING

A Sprint starts with a Sprint Planning Meeting. This is actually divided into two meetings, each timeboxed to 4 hours. The first meeting is a meeting between the team and the Product Owner, where they discuss the next Sprint and the User Story or Stories that the team will be working on during this Sprint. The Product Owner explains the User Stories in detail. They then agree on how much and what work the team will commit to complete during the next Sprint and an overall goal for that Sprint is determined.

For the second part of the meeting (the last 4 hours), the team is left alone to plan the Sprint and how to achieve the Sprint goal. They do this by determining all the tasks required to complete the Sprint. They then enter these into a Sprint Backlog with a time estimate for each task. Even though the team is doing this on their own, they are allowed to invite consultants and experts on areas they will be working on if they need a better insight into certain areas of the Sprint, to determine tasks required and time estimation for these tasks. These consultants and experts will leave the meeting when their expertise is not required anymore.

3.3 DAY TO DAY PLANNING AND CONTROL

As already seen for the Sprint planning, the *team* is the "project leader". Therefore they are self-organising, which means that there is no-one who will tell them what to do or how to do it. The working day starts with a *daily stand-up meeting.* During this meeting, each team member answers 3 questions: What has the team member done since the last daily stand-up meeting, what is the team member planning on doing until the next stand-up meeting, and has the team member any impediments that restricts him/her from doing the planned work, or slows him/her down. This can be anything from an algorithm that the team member is struggling with to a management member who has asked him/her to do some additional work not related to the task. Solutions to these impediments are not discussed during the daily stand-up meeting, but they are noted by the ScrumMaster and can be discussed after the meeting. In general it is the ScrumMaster's responsibility to remove these impediments.

The purpose of this meeting is to let everyone get an insight into what everybody else is working on, the general progress of the project and if they can help or get help from some of the other team members. It's also during this meeting that tasks are assigned by the team members themselves.

At the end of a working day, the hours actually used on tasks worked on and a new estimate of the remaining hours are both entered into the Sprint Backlog. From this, a burn-down chart can be produced, which shows the progress of the project and the "burn-down" of tasks visually. This gives a good overview of the actual progression, and the team can use this to easily see if they have to increase their effort or by other means increase the Sprint burn-down velocity.

4 MORE ON SCRUM TOOLS

4.1 PRODUCT BACKLOG

It is important to notice that the Product Backlog is subject to change at any time. It is available to every stakeholder, and is maintained by the *Product Owner*. The Product Owner represents all the customers and creates and manages the Product Backlog based on the needs and wishes from all the customers. It is his job to prioritise the User Stories in a way that is satisfactory to the customers. The priority can be changed for a User Story at any time as long as it is not started on by the team if the customer requirements changes or other User Stories that are considered more important are added to the Product Backlog. The customers can use the Product Backlog to see which features they will get and in which release they will get them. The team will not use the Product Backlog to any extent, as they should not plan for anything not in the current Sprint since everything that is not in that Sprint is subject to change.

4.2 USER STORIES

A User Story is written from a customer's perspective. In fact, it is often the customer that writes the User Stories. If there are many customers that write a similar User Story, the Product Owner can write a common one. The stories are written from a non-technical perspective and use a common language to describe something that the customer can see directly in the program. E.g. it does not explain a fast search algorithm that a search function will implement, just that the search result is displayed quickly after the button is pushed. A common format for the User Story is:

As a *User* ***I want to*** *do something* ***so that*** *I learn something.*

Here, the user can be changed with any other role, like administrator, manager, editor etc. The "I want to" is a description of the actual feature described from a user perspective, and the "so that" part explains the reason for why the feature is wanted. The "so that" is not mandatory and can be left out if it is obvious why you want the feature, but it is important for the developers to understand the motivation for that User Story.

5 THE PROS AND CONS OF SCRUM PROJECT PLANNING

As was said in the introduction, the main difference in the planning is apparent in when the project plan is detailed. For those supporting Scrum this is often highlighted as one of the strong features, while those who are arguing against Scrum would often highlight it as a weakness. So it is safe to say that it is both a strength and a weakness.

It is a **strength** because in Scrum there is almost no overhead time lost on planning a feature or part of an iteration that later needs to be changed or removed because of the circumstances. The team will only spend time on planning the tasks that they are certain they will complete.

It is a **weakness** because the team does not take into consideration features and User Stories further down the Product Backlog, and therefore these can be harder to implement at a later time without redoing the work already done.

Another **strength** is that it is the team itself that does the planning and time estimation, and not a project leader. This is because it is the team that works on the project every day and therefore has the best knowledge about the complexity of a new task, and when and how it should be done.

A complex problem can often be almost impossible to plan on a detailed level because there are too many unknown factors, and the time required to plan such a complex problem increases almost exponentially with the increase in complexity. But with Scrum, there would normally not be spent much more time on planning than the time spent on the Scrum meetings. The team finds a small part of the complex problem that they can start on to deliver a potential shippable product at the end of the first Sprint, and has probably acquired more knowledge on how to proceed and what to do next at the end of that Sprint.

6 CALCULATION OF TIME ESTIMATES IN SCRUM

6.1 STORY POINTS

In the Product Backlog the time and resource estimate for the User Stories are given in *story points.* This is a dimensionless number, meaning it does not reflect a number of hours, days or weeks to complete a User Story. Rather, it illustrates the complexity of a User Story compared to the other User Stories in the Product Backlog. This means that, if a User Story is given an estimate of 2 story points, it's about half as complex as a User Story that is estimated to 4 story points.

6.2 POKER PLANNING

To determine the estimates for the User Stories, we use a meeting called *poker planning*, or just *playing poker.* At a poker meeting all the team members are present, and are dealt a hand of cards. The cards we use are special *poker planning* cards, each card with a valid story point value, and each member will have a full hand, i.e. all the different values.

The valid story point values for our poker meetings are *0, 0.5, 1, 2, 3, 5, 8, 13, 20, 40, 100 and ∞.* The reason why only these values are valid is that it's not possible to give a really accurate estimate of whether it will take 5, 6, 7 or 8 points, but it should be possible to say if it's about twice as big. The higher the value is, the harder it gets to give an accurate estimate of the complexity. The fact that it is considered a big and complex User Story implies that it might be several unknown factors and therefore hard to give an accurate estimate.

When the cards are dealt, the team starts to discuss one of the User Stories in the Product Backlog. They discuss what the User Story involves and the complexity of it, but they do not discuss actual story points. When they are done discussing, each player puts one of the cards down on the table with the value facing down. When everybody has decided on a card, the cards are turned. If the players disagree on the estimate, the cards are pulled back and the discussion continues. If some team members have picked a high value, this might indicate that they know or realise something about the User Story that the other team members don't. They continue the discussion and then play cards until they agree on one value. When everybody agrees, they have the estimate for that User Story, and can continue with the next story until all the stories are done.

6.3 VELOCITY

The velocity is the number of story points a team can complete during a Sprint scaled by a suitable factor. This number is based on the team member's availability and number of hours they can work on that Sprint. The planned velocity can be used to determine in which Sprint a User Story will be completed. Of course, this will change as the Product Backlog changes. At the end of a Sprint, the actual Sprint velocity for that Sprint can be used to calculate a more accurate velocity for the following Sprints. Normally, an average of the last 2-3 Sprints are used to calculate the estimated velocity if the team availability stays the same. The velocity may also be adjusted up or down as necessary.

7 REFERENCE(S)

1. *Agile Software Development with Scrum*, Ken Scwhaber and Mike Beedle, Pearson International Edition (ISBN-13: 978-0-13-207489-6).

2. *Agile Project Management with Scrum*, Ken Schwaber, Microsoft Press, (ISBN-13: 9780735619937).

SCRUM RULES

HIBU STUDENT PROJECT 2011
CORENA S1000D 4.0 PROCESS DATA MODULE RENDERER

____
Arild Oldervoll

____
Marius Haraldseth

____
Eirik André Eidså

____
Olav Brandt

TABLE OF CONTENTS

REVISION HISTORY			
Version Number	**Date**	**Changes**	**Assigned**
1	06-Jan-2011	First version	AOL, OBR

1 INTRODUCTION

Our assignment to develop a logic engine for support execution of S1000D process Data Modules version 4.0, is given by CORENA Norge A/S. This logic engine can take input from users and/or external applications, such as sensors, to decide its next step in a flow of process Data Modules. It can branch and loop through the selected process Data Module. The logic engine will be an interpreter for process Data Modules. We will make it a stand-alone logic engine in a web based application running on Jetty/Tomcat. The user of the application should be able to move back and forth through the steps using previous and next buttons. The user should be able to save the session to continue at a later time. The logic engine should support the International S1000D Specification Issue 4.0.

1.1 INTENTION OF THIS DOCUMENT

The purpose of this document is to define Scrum rules the project group agrees to follow during the project.

1.2 SYNONYMS, ACRONYMS AND DEFINITIONS

Term	Explanation
Time-box	A period of time that cannot be exceeded.
Sprint	A defined time-box of 14 to 30 calendar days, often called an iteration in other models
Team	The group members committed to a Sprint.
ScrumMaster	The person responsible for the Scrum process.
Stakeholder	Someone with an interest in the outcome of the project.
Product Backlog	Requirements listed in prioritised order with time estimates.
Sprint Backlog	List of tasks for the current Sprint, defined by the team
Product Owner	Represents all stakeholders in the project and is responsible for the Product Backlog.
Chicken	Someone interested in the project but does not have a formal Scrum responsibility.
Pig	Someone committed to the project, occupying one of the formal Scrum roles.

1.3 AUTHOR(S)

Arild Oldervoll (AOL)

1.4 ASSIGNED

Arild Oldervoll (AOL)

2 OVERVIEW

The rules defined here are mostly a copy of the rules that Ken Schwaber has listed in his book *Agile Project Management With Scrum* [1]. Schwaber is one of the two main creators of Scrum, and he claims that these rules have worked in thousands of projects. It is important to have defined rules for the Scrum process that everyone involved in the process has agreed to follow. This is both to hold the Scrum process together and to avoid confusion.

2.1 COMMITMENT

All those who have a pig-role in the project, i.e. those who have made a commitment to the project, also commit to follow these rules

2.2 CHANGES TO THE RULES

At the end of every Sprint there is the Sprint retrospective meeting. Rule changes should be suggested and discussed at this meeting. It is important that these changes originate from the team, and not the management. Before a rule change is approved, the ScrumMaster should be convinced that this rule change will enhance the process and that the team understands the full meaning of the rule change.

2.3 SPRINT PLANNING MEETING

The Sprint planning meeting is time-boxed to 8 hours and consists of two segments that are time-boxed to 4 hours each. The first segment is for selecting Product Backlog and the second segment is for preparing a Sprint Backlog.

2.3.1 GENERAL

- The attendees for the Sprint planning meeting are the ScrumMaster, the Product Owner and the team. Additional parties can be invited by any of these people to provide additional business domain or technology domain information and advice, but they are dismissed after this information is provided. There are no chickens as observers present.
- The Product Owner must prepare the Product Backlog prior to the meeting. In absence of either the Product Owner or the Product Backlog, the ScrumMaster is required to construct an adequate Product Backlog prior to the meeting and to stand in for the Product Owner.

2.3.2 FIRST SEGMENT

- The goal of the first segment, or first 4 hours, is for the team to select those Product Backlog items that it believes it can commit to turn into an increment of potentially shippable product functionality. The team will demonstrate this functionality to the Product Owner and stakeholders at the Sprint review meeting at the end of the Sprint.
- The team can make suggestions, but the decision of what Product Backlog can constitute the Sprint is the responsibility of the Product Owner.
- The team is responsible for determining how much of the Product Backlog that the Product Owner wants worked on that the team will attempt to do during the Sprint.
- Time-boxing the first segment to 4 hours means that this is all of the time that is available for analysing the Product Backlog. Further analysis must be performed during the Sprint. Large-grained, high-priority Product Backlog with imprecise estimates might not be thoroughly understood during this part of the Sprint planning meeting and might result in the team not being able to complete all of the Product Backlog it selects.

2.3.3 SECOND SEGMENT

- The second segment of the Sprint planning meeting occurs immediately after the first segment and is also time-boxed to 4 hours.
- The Product Owner must be available to the team during the second segment to answer questions that the team might have about the Product Backlog.
- It is up to the team, acting solely on its own and without any direction from outside the team, to figure out during the second segment how it will turn the selected Product Backlog into an increment of potentially shippable product functionality. No one else is allowed to do anything but observe or answer questions that the team may have in order to obtain further information.
- The output of the second segment of the Sprint planning meeting is a list, called the Sprint Backlog, of tasks, task estimates and assignments that will start the team on the work of developing the functionality. The task list might not be complete, but it must be completed enough to reflect mutual commitment on the part of all team members and to carry them through the first part of the Sprint, while the team devises more tasks in the Sprint Backlog.

2.4 DAILY SCRUM MEETING

- The Daily Scrum meeting is time-boxed to 15 minutes.
- The Daily Scrum will be held at the same place and the same time every work day.

- All team members are required to attend. If for some reason a team member cannot attend in person, the absent member must attend either by telephone/Skype or by having another team member report on the absent member's status.
- Team members must be prompt. The ScrumMaster starts the meeting at the appointed time, regardless of who is present.
- The ScrumMaster begins the meeting by starting with the person immediately to his or her left and proceeding clockwise around the room until everyone has reported
- Each team member should respond to three questions only:
  - What have you done since the last Daily Scrum regarding this project?
  - What will you do between now and the next Daily Scrum meeting regarding this project?
  - What impedes you from performing your work as effectively as possible?
- Team members should limit themselves to answering these three questions, not digressing into issues, designs, discussion of problems, or gossip. The ScrumMaster is responsible for moving the reporting along briskly, from person to person.
- During the Daily Scrum, only one person talks at a time. That person is the one who is reporting his or her status. Everyone else listens. There are no side conversations.
- When a team member reports something that is of interest to other team members or needs assistance of other team members, any team member can immediately arrange for all interested parties to get together after the Daily Scrum to set up a meeting.
- Chickens are not allowed to talk, make observations, make faces, or otherwise make their presence in the Daily Scrum meeting obtrusive.
- Chickens stand on the periphery of the team so as not to interfere with the meeting.
- If too many chickens attend the meeting, the ScrumMaster can limit attendance so that the meeting can remain orderly and focused.
- Chickens are not allowed to talk with team members after the meeting for clarification or to provide advice or instructions.
- Pigs or chickens who cannot or will not conform to the above rules can be excluded from the meeting (chickens) or removed from the team (pigs).

3 SPRINT

- We have decided to have the Sprints time-boxed to 21-30 consecutive calendar days.
- The team can seek outside advice, help, information and support during the Sprint.
- No one can provide advice, instructions, commentary, or direction to the team during the Sprint unless asked by the team. The team is utterly self-managing.
- The team commits to Product Backlog during the Sprint planning meeting. No one is allowed to change this Product Backlog during the Sprint. The Product Backlog is frozen until the end of the Sprint.
- If the Sprint proves to be not viable, the ScrumMaster can abnormally terminate the Sprint and call a new Sprint planning meeting to initiate the next Sprint. The ScrumMaster can make this change of his or her own accord or as requested by the team or the Product Owner. The Sprint can prove not to be viable if the technology proves unworkable, if the business conditions change so that the Sprint will not be of value to the business, or if the team is interfered with during the Sprint by anyone outside the team.
- If the team feels itself unable to complete all of the committed Product Backlog during the Sprint, it can consult with the Product Owner on which items to remove from the current Sprint. If so many items require removal that the Sprint has lost its value and meaning, the ScrumMaster can abnormally terminate the Sprint, as previously stated.
- If the team determines that it can address more Product Backlog during the Sprint than it selects during the Sprint planning meeting, it can consult with the Product Owner on which additional Product Backlog items can be added to the Sprint.
- The team members have two administrative responsibilities during the Sprint:
    - They are to attend the Daily Scrum meeting,
    - They are to keep the Sprint Backlog up-to-date and available in a public folder on a public server, visible to all. New tasks must be added to the Sprint Backlog as they are conceived, and the running, day-to-day estimated hours remaining for each task must be kept up-to-date.

4 SPRINT REVIEW MEETING

- The Sprint review meeting is time-boxed to 4 hours.
- The team should not spend more than 1 hour preparing for the Sprint review.
- The purpose of the Sprint review is for the team to present to the Product Owner and stakeholders functionality that is done. *Done* means here that the functionality is completely engineered with documentation and could be potentially shipped or implemented.
- Functionality that has not achieved *done* status cannot be presented.
- Artefacts that aren't functionality cannot be presented except when used in support of understanding the demonstrated functionality. Artefacts cannot be shown as work products, and their use must be minimised to avoid confusing stakeholders or requiring them to understand how system development works.
- Functionality should be presented on the team members' workstations and executed from the server closest to production.
- The team Sprint review starts with a team member presenting the Sprint goal, the Product Backlog committed to, and the Product Backlog completed. Different team members can then discuss what went well and what did not go well in the Sprint.
- The majority of the Sprint review is spent with team members presenting functionality, answering stakeholder's questions regarding the presentation, and noting changes that are desired.
- At the end of the presentations, the stakeholders are polled, one by one, to get their impressions, any desired changes, and the priority of these changes.
- The Product Owner discusses with the stakeholders and the team potential rearrangement of the Product Backlog based on the feedback.
- Stakeholders are free to voice any comments, observations, or criticisms regarding the increment of potentially shippable product functionality between presentations.
- Stakeholders can identify functionality that occurs to them as they view the presentation and request that the functionality is added to the Product Backlog for prioritisation.
- The ScrumMaster should attempt to determine the number of people who expect to attend the Sprint review meeting and set up the meeting to accommodate them.
- At the end of the Sprint review, the ScrumMaster announces the place and date of the next Sprint review to the Product Owner and all stakeholders.

5 SPRINT RETROSPECTIVE MEETING

- The Sprint retrospective meeting is time-boxed to 3 hours.
- It is attended only by the team, the ScrumMaster and the Product Owner. The Product Owner is optional.
- The meeting starts with all team members answering two questions:
    - What went well during the last Sprint?
    - What could be improved in the next Sprint?
- The ScrumMaster writes down the team's answers in summary form.
- The team prioritises in which order it wants to talk about the potential improvements.
- The ScrumMaster is not at this meeting to provide answers, but to facilitate the team's search for better ways for the Scrum process to work for it.
- Actionable items that can be added to the next Sprint should be devised as high-priority non-functional Product Backlog. Retrospectives that don't result in change are sterile and frustrating.

6 REFERENCE(S)

1. *Agile Project Management With Scrum*, Ken Schwaber, Microsoft Press, 2003. (ISBN-13: 978-0-7356-1993-7)

Product Backlog - CORENA S1000D 4.0 Process Data Module Renderer

Priority	Id	Estimate	Sprint	User Type	Story	Story Type
1	38	5	11	User	**As a** User **I want** all booleanAction attribute values listed on the S1000D pDM specification to be supported by the logic engine.	Story
2	40	5	11	User	**As a** User **I want** all the setAction attribute values listed on the S1000D pDM specification to be supported by the logic engine.	Story
3	41	13	11	User	**As a** User **I want** all StringAction and SubStringAction attribute values listed on the S1000D pDM specification to be supported by the logic engine.	Story
4	22	8	11	User	**As a** User **I want** the logic engine to evaluate an <expression> to determine if the <dmThenSeq> or the <dmElseSeq> is valid **so that** the correct next <dmNode> is displayed when the next button is pressed.	Story
5	24	20	11	User	**As a** User **I want** all the <proceduralStep> and <proceduralStepAlt> elements in a <dmNode> to be displayed correctly on the screen **so that** the logic engine processes all <proceduralStep> and <proceduralStepAlt> elements before displaying them to calculate the correct display according to the S1000D specification.	Story
6	17	13	11	User	**As a** User **I want** a 'previous' button **so that** I can go back to previous steps.	Story
7	25	5	12	User	**As a** User **I want** the next and previous buttons to be disabled if a <proceduralStep> contains a <dialog> or <dialogAlt> **so that** the buttons are yeilding for the OK and Cancel buttons defined in the dialog elements.	Story
8	26	13	12	User	**As a** User **I want** the correct element from an element group such as <dmNodeAlt>, <proceduralStepAlt>, <dialogAlt> or <messageAlt> to be displayed based on the evaluation of the <expression> element.	Story
9	39	13	12	User	**As a** User **I want** all numberAction attribute values listed on the S1000D pDM specification to be supported by the logic engine.	Story
10	6	20	12	User	**As a** User **I want** to communicate with external applications **so that** the next appropriate step is automatically displayed.	Story
11	27	8	12	User	**As a** User **I want** the <dmRef> element to be processed correctly by the logic engine **so that** if the reference is another process data module the logic engine will execute it or **so that** if the reference is not to another process data module the referenced data module is sent to the IETP.	Story
12	30	20	12	User	**As a** User **I want** the logic engine to handle failed evaluations and error conditions occuring while evaluating expressions as defined in the specification.	Story

Product Backlog - CORENA S1000D 4.0 Process Data Module Renderer

Priority	Id	Estimate	Sprint	User Type	Story	Story Type
13	45	13	12	User	**As a** User **I want** <table> elements to be displayed when specified in a dmNode	Story
14	46	8	12	User	**As a** User **I want** <message> elements to be displayed when specified in a dmNode	Story
15	32	20	12	User	**As a** User **I want** <figure> elements to be displayed in the main screen with the actual content of the element displayed.	Story
16	33	20	12	User	**As a** User **I want** <multimedia> elements to be displayed in the main screen.	Story
17	23	13	12	User	**As a** User **I want** the logic engine to enter a loop where it is specified by a <dmLoop> element until the <expression> evaluates to false.	Story
18	8	13	12	User	**As a** User **I want a** warning message to be displayed if the process Data Module contains unsupported elements **so that** I will be aware of the problem.	Story
19	43	8	13	User	**As a** User **I want** information about runtime errors to be recorded in detail **so that** it is possible to reconstruct the errors in order to pinpoint the problem.	Story
20	42	40	13	User	**As a** User **I want** all runtime errors to be detected and handled by the logic engine	Story
21	42	40	13	User	**As a** User **I want** all runtime errors to be detected and handled by the logic engine	Story
99	9	0	13	User	**As a** User **I want** a web based user interface.	Story
99	9	0	13	User	**As a** User **I want** a web based user interface.	Story

QUALITY ASSURANCE

HIBU STUDENT PROJECT 2011

CORENA S1000D 4.0 PROCESS DATA MODULE RENDERER

____

Arild Oldervoll

____

Marius Haraldseth

____

Eirik André Eidså

____

Olav Brandt

TABLE OF CONTENTS

REVISION HISTORY			
Version Number	**Date**	**Changes**	**Assigned**
1	04-Jan-2011	First version	OBR

1 INTRODUCTION

Our assignment to develop a logic engine for support execution of S1000D process Data Modules version 4.0, is given by CORENA Norge A/S. This logic engine can take input from users and/or external applications, such as sensors, to decide its next step in a flow of process Data Modules. It can branch and loop through the selected process Data Module. The logic engine will be an interpreter for process Data Modules. We will make it a stand-alone logic engine in a web based application running on Jetty/Tomcat. The user of the application should be able to move back and forth through the steps using previous and next buttons. The user should be able to save the session to continue at a later time. The logic engine should support the International S1000D Specification Issue 4.0.

1.1 INTENTION OF THIS DOCUMENT

This document describes the strategies, methods and procedures we will follow in regards to quality assurance for our product.

1.2 SYNONYMS, ACRONYMS AND DEFINITIONS

Synonym	Description
Process	Set of instructions and conditions being executed
Data Module	Container of data/information.
IETP	Interactive Electronic Technical Publication - Technical manual prepared in digital form on a suitable medium.
Process DM	Process Data Module
HiBu	Høgskolen i Buskerud

1.3 AUTHOR(S)

Olav Brandt (OBR)

1.4 ASSIGNED

Olav Brandt (OBR)

2 DOCUMENTATION REVIEW

All of the documentation we produce during this project will be thoroughly reviewed. After a document has initially been written, two non-authoring group members review it, before sending it back to the original author. Prior to delivery, the documents undergo another round. Each group member reviews the document, and the assigned person must confirm the changes before marking it *Official*.

3 SOFTWARE TESTING

3.1 WHAT DO WE TEST?

Before defining *how* we are going to perform our tests, it could be interesting to take a look at *what* we are going to test.

3.1.1 VERIFICATION AND VALIDATION

When we are testing, we are basically attempting to do two things in relation to our product; verifying and validating its correctness.

Verification is the process of determining whether or not you have *built the product correctly*. In question form; “Does the software fulfil the requirements?” and/or “Does the software work correctly according to the requirements?”

Validation is about checking if you have *built the correct product*. “Are the requirements correct in view of the needs and demands of the customer?” and/or “Does the software do what the customer wants it to do?”

Naturally, *both* of these activities are fundamental to any kind of product development. If the software you have built fails verification, it means it’s in a broken state and not usable by the customer. If it on the other hand passes verification, but fails validation, it means you might have built a working, high quality piece of software – but not the correct *kind* of software.

3.1.2 LEVELS OF TESTING

A software development process is usually divided into levels; at the “top” we find requirement gathering and specification, then analysis, system design, and at the “bottom”, code/class design. Each of these levels of development should have a corresponding level of testing.

Acceptance testing is at the highest level. Here, both requirements and completed (pieces of) software are tested in order to validate them in view of the customer’s wishes.
System/scenario testing accompanies the analysis level, and tests the correctness of the entire system.
Integration/subsystem testing targets the integration of individual units into subsystem collections.
Unit testing is the testing of the smallest part of the software, the unit. In object-oriented software testing, the units are classes.

3.1.3 FUNCTIONAL AND NON-FUNCTIONAL TESTING

Additionally, testing is divided into two primary categories; functional and non-functional tests. A **functional** test is necessarily the result of one or more functional requirements. “Can the software perform the action described in this requirement?” A **non-functional** test is likewise connected to non-functional requirements, including aspects like stability, usability, and performance.

3.2 HOW DO WE TEST?

In a testing process, all of these areas must be covered. However, as opposed to a normal industry project where developers and testers are separate people with separate roles, we need to fill both pairs of shoes. Of course, that doesn't mean we don't have to perform all of the necessary tests, but our testing strategy is influenced by this reality.

3.2.1 TECHNIQUES

There are two main types of testing techniques, correlating to the tester's point of view when looking at the software.

White box testing refers to tests done "from the inside". In this technique, the tester has a full view of the inside of the software, i.e. the source code, and is testing the inner workings of it.

Black box is the opposite, where the testing occurs "from the outside". Here, the tester cannot see the insides of the software, but is testing functionality described in the requirements.

White box testing requires technical knowledge of programming and of the inner workings of the software, and is commonly performed by a developer. Black box testing requires no such knowledge, and often follows pre-written test cases with some chosen input, and expected output. In a typical industry environment, this is performed by a dedicated tester.

Since we will be doing both, we need to be extra careful and make sure everything is tested properly according to the specifications – not just simply satisfying our preconceived expectations. As far as possible, we will try having different people developing and black box testing a piece of software.

3.2.2 APPROACHES

Testing can be approached either with dynamic or static methods.

Dynamic testing is where you test by running the parts of the software to be tested. Meaning, the tests examine the program behaviour, watching how it responds to given specific input.

Static testing on the other hand, does not involve actually using the software. Instead, work produced during development is read, reviewed and checked. For instance, reading through the requirements and making sure they fit the user's wishes, or testing that an algorithm is valid by manually inspecting and analysing it.

Although our testing will be largely dynamic, we'll also perform some static testing by doing code reviews. After the initial completion of a programming task, the developer will send his code to another team member for review. This other team member will pass this reviewed (and revised) code along to a third person, who, once finished, will send it back to the original developer for his final review.

3.3 FUNCTIONAL TESTS

For our (dynamic) testing of functionality and functionality derived areas, we will follow a bottom-up strategy. Meaning, we will begin at the "bottom", testing small software units, and then work ourselves up, testing subsystems and larger pieces of software made up from the smaller units. This of course corresponds to the standard way of doing software development in object-oriented programming. Some modifications and/or additional steps will be necessary as we're not following a linear development model.

3.3.1 UNIT TESTING

The basis of our testing will be the unit testing of classes (including their methods). In short, the point of a unit test is the testing of a unit (class) on its own; in isolation. By specifying different input and the output that should be the result, you are able to test if your unit performs as expected. It also makes refactoring code much less scary, as you can constantly test to see if your changes broke the class. As a side note; because the unit should be completely isolated (you only want to find errors in the unit you're testing), any dependencies will need to be faked/mocked by the use of interfaces and dummy objects – a demand that places certain requirements on the way your classes are designed.

By verifying that all our units are working correctly, the testing of the next level should be limited in complexity, because the problem area is reduced to the interaction between (tested) units. This again should help in the same way with the testing of the next level after that, which is the interaction between (tested) subsystems. Additionally, having a full suite of unit tests will certainly be helpful when doing regression testing (checking to see if anything was broken because of a code change/fix)

Unit tests are white box tests, and will be done via an automated unit testing framework, and various other tools (mocking, coverage, etc.) where appropriate.

3.3.2 INTEGRATION/SUBSYSTEM TESTING

As already mentioned; because we have unit tested all necessary classes, the next level of testing should be limited in both complexity and scope. For our integration/subsystem tests, we will be testing the communication/cooperation of a collection/subsystem made up by the units we tested at the previous level.

Additionally, another bonus of having done unit tests is the natural extension these have into the integration/subsystem testing; while you in unit testing wanted to avoid dependencies and used dummy objects, the opposite is a central part of what you *want* to test at this level.

This means that you can reuse large parts of the unit tests, only changing the dummy objects with the real ones. When this is not enough, additional automated tests will be written where possible, otherwise manual testing must be done.

These mentioned tests are considered white box tests. Black box testing will be done as well, and because we are using Scrum, we are essentially testing the integration of a new feature into our pre-existing, and already tested, program. This testing will vary depending on *what* is being integrated, like the user interface, file handling, or other logic, and is assumed to be mostly done manually, according to test cases. Automated testing might be implemented for some features where this is applicable and beneficial.

3.3.3 SYSTEM/SCENARIO TESTING

If the proper unit and integration testing has been performed, system testing should be a matter of checking that the subsystems play nice together, and that everything is "hooked up" correctly. At this level, our automated white box tests will probably be too difficult to use due to the increased complexity. Black box testing done manually will thus be the preferred approach. Since system testing is mainly about test case-driven functionality testing, this kind of hands on testing is something that needs to be done extensively either way.

3.3.4 ACCEPTANCE TESTING

The final level of testing is customer acceptance testing. This is where a selected group of the actual customers *validates* that the product performs as they expect. In conventional testing, acceptance tests are usually run once the requirements are finished, and again once the entire program is completed. In Scrum, however, additional *User Story acceptance tests* are done after a Sprint, on each completed User Story. That way, every feature will go through validation while the project is still in development, and potential changes are identified at a much earlier stage than in conventional testing.

4 REFERENCES

1. *Prosjekthåndbok 2010*, Torbjørn Strøm and Olaf Hallan Graven, Høgskolen i Buskerud, 2010

2. *Unit testing*, Wikipedia, http://en.wikipedia.org/wiki/Unit_testing (last visited 14-Dec-2010)

3. *Integration testing*, Wikipedia, http://en.wikipedia.org/wiki/Integration_testing (last visited 14-Dec-2010)

4. *System testing*, Wikipedia, http://en.wikipedia.org/wiki/System_testing (last visited 14-Dec-2010)

5. *Acceptance testing*, Wikipedia, http://en.wikipedia.org/wiki/Acceptance_testing (last visited 14-Dec-2010)

6. *Test automation*, Wikipedia, http://en.wikipedia.org/wiki/Test_automation (last visited 14-Dec-2010)

7. *Manual testing*, Wikipedia, http://en.wikipedia.org/wiki/Manual_testing (last visited 14-Dec-2010)

8. *Software testing*, Wikipedia, http://en.wikipedia.org/wiki/Software_testing (last visited 14-Dec-2010)

HIGH LEVEL ACTIVITY OVERVIEW

HIBU STUDENT PROJECT 2011

CORENA S1000D 4.0 PROCESS DATA MODULE RENDERER

Arild Oldervoll

Marius Haraldseth

Eirik André Eidså

Olav Brandt

Activity Number	Activity Name
100	**PROJECT MANAGEMENT**
101	PLANNING
102	ECONOMY
104	PROGRESS REPORTS
105	TIME TRACKING
199	OTHER PROJECT MANAGEMENT
200	**DOCUMENTATION**
201	TEMPLATES
202	VISION DOCUMENT
203	PRESTUDY REPORT
204	REQUIREMENTS SPECIFICATION
205	TEST SPECIFICATION
206	TECHNOLOGY DOCUMENTATION
207	USER MANUAL
208	PROJECT PLAN
209	SCRUM DOCUMENTATION
210	CODE STANDARD DOCUMENT
211	S1000D PDM DOCUMENTATION
212	ANALYSIS DOCUMENT
213	DESIGN DOCUMENT
214	IMPLEMENTATION DOCUMENT
215	PROJECT REVIEW
216	HAND-OFF DOCUMENT
299	OTHER DOCUMENTATION
300	**MEETINGS**
301	MEETING PREPARATION
302	SCRUM MEETINGS
303	STATUS MEETINGS
304	MEETING MINUTES
305	PLANNING MEETINGS
399	OTHER MEETINGS
400	**RESEARCH**
401	TECHNOLOGY
402	TOOLS
499	OTHER RESEARCH
500	**ANALYSIS AND DESIGN**
501	ANALYSIS
502	DESIGN
600	**IMPLEMENTATION**

601	PROTOTYPING
602	PROGRAMMING
700	**QUALITY ASSURANCE**
701	TESTING
702	DOCUMENT REVIEW
800	**GENERAL**
801	WEBSITE
802	PRESENTATIONS
803	STANDARDS
804	LECTURES
805	SETUP ENVIRONMENT
899	OTHER

VISION DOCUMENT

HIBU STUDENT PROJECT 2011
CORENA S1000D 4.0 PROCESS DATA MODULE RENDERER

____
Arild Oldervoll

____
Marius Haraldseth

____
Eirik André Eidså

____
Olav Brandt

TABLE OF CONTENTS

REVISION HISTORY			
Version Number	**Date**	**Changes**	**Assigned**
2	06-Jan-2011	Second Official Document - Added front page, and standard introduction. - Added description to group name, edited title. - Several smaller changes. - Changed logo pictures. - Added references. - Document Review by all team members.	MHA

1 INTRODUCTION

Our assignment to develop a logic engine for support execution of S1000D process Data Modules version 4.0, is given by CORENA Norge A/S. This logic engine can take input from users and/or external applications, such as sensors, to decide its next step in a flow of process Data Modules. It can branch and loop through the selected process Data Module. The logic engine will be an interpreter for process Data Modules. We will make it a stand-alone logic engine in a web based application running on Jetty/Tomcat. The user of the application should be able to move back and forth through the steps using previous and next buttons. The user should be able to save the session to continue at a later time. The logic engine should support the International S1000D Specification Issue 4.0.

1.1 INTENTION OF THIS DOCUMENT

It is two main objectives for the vision document.

1.1.1 OBJECTIVE 1:

To clarify the assignment, scope and terms with the employer, i.e. CORENA NORGE AS.

1.1.2 OBJECTIVE 2

To get the assignment and project group approved by HiBu Kongsberg.

1.2 SYNONYMS, ACRONYMS AND DEFINITIONS

Synonym	Description
Process	Set of instructions and conditions being executed
Data Module	Container of data/information.
IETP	Interactive Electronic Technical Publication - Technical manual prepared in digital form on a suitable medium.
Process DM	Process Data Module
HiBu	Høgskolen I Buskerud

1.3 AUTHOR(S)

Marius Haraldseth (MHA)

1.4 ASSIGNED

Marius Haraldseth (MHA)

2 PARTICIPANTS

2.1 STUDENTS

The group name *Ameo* comes from the first letter in our names.

- Arild Oldervoll (AOL) – Project Manager
  - arild@oldervoll.com – phn: 41 45 29 60
- Marius Haraldseth (MHA)
  - marius@haraldseth.net – phn: 41 52 06 10
- Eirik André Eidså (EAE)
  - eirik@eidsa.no – phn: 45 85 92 44
- Olav Brandt (OBR)
  - olav@brafa.net – phn: 98 44 87 17

2.2 CORENA

- Øyvind Ottersen – External Examiner
- Roger Werner Laug – CORENA S1000D Product Owner
- Tommy Sivertsen – CORENA IETP Product Owner and our External Mentor
- Jarle Hjortland – Responsible Architect
- Per Jøran Lund – System building and install
- Øystein Hansen – Web Client / GWT responsible in the project

3 THE ASSIGNMENT

3.1 EMPLOYER

CORENA S1000D is a product for configuration control, maintenance, viewing and production of maintenance documentation for larger machines and vessels. Customers using it are e.g. Pratt & Whitney for their aircraft engines in the F-35 combat aircraft, Boeing, Bombardier's C-series and new regional jets, Goodrich, Eurocopter (Norwegian defense's new NH-90 Helicopters) etc. Previously this was software only available for the military market, but lately it has also become available for the civil market.

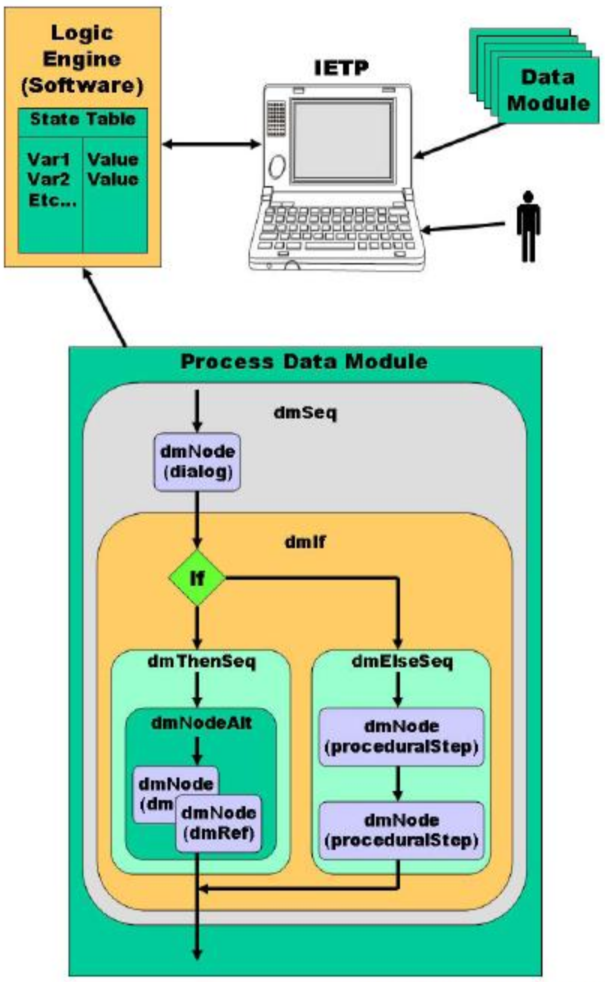
CORENA is one of the three leading software companies in the world in their area, and their goal and ambition is to become number one.

3.2 THE TASK

A process Data Module (DM) is an XML schema which describes different steps in an information process. Based on interactive user dialogs, sensors, applications and external services through an interface the logic engine decides the next Data Module and/or step in the sequence. Decision points (branching), looping and selective filtering are supported in the S1000D process DM. The process DM can be seen as a process flow script, where the return values define the next action. The S1000D process DM represents a procedural flow consisting of several DM's and/or steps that are sequenced.

The process DM is input data which is processed in a logic engine. The logic engine is an interpreter for the process DM's, and throughout the process it defines states. Our assignment is to develop this logic engine for the execution of a process DM's components. Based on the components in the DM, an interaction between the user and the IETP should occur (see figure 3.1).

The process DM is especially well suited to represent procedural data, fault data and descriptive data.



ICN-S1000D-A-070201-A-D0216-00011-A-002-01

Fig 1 Process data module conceptual diagram

FIGURE 3.1 PROCESS DATA MODULE CONCEPTUAL DIAGRAM

3.3 WHAT ARE THE REQUIREMENTS FOR THE LOGIC ENGINE

- Previous function.
- Next function.
- Log the states in a state table.
- Save current position with history before exit.
- The support execution of the logical document should be according to the international S1000D issue 4.0 specification which can be integrated with CORENA's products.
- The implementation should be based on technology which is base for CORENA S1000D Web Client and CORENA IETP products.
- The interpreter is supposed to execute in an independent web based IETP application based on Jetty/Tomcat.

S1000D specification resources: http://www.s1000d.org

Relevant chapters:

- 4.11 – Information management – Process Data Module
- 7.7.1 – Guidance and examples – Logic Engine
- 7.7.2 – Guidance and examples – Process Data Module Nodes

Common technology platform for the CORENA S1000D Web Client and CORENA IETP products is:

- Java
- XML / XSL
- Spring Framework
- POJO
- Web services
- Ajax, GWT / GXT (Google Web Toolkit)

3.4 OTHER REQUIREMENTS

- All technical documentation which will be used later in CORENA should be in English.
- CORENA wants us to use the Scrum development methodology.

4 EXPECTED RESULTS

CORENA is expecting a logic engine which should be integrated in an independent web-based solution running on Jetty/Tomcat. The user can choose a process DM, and step through this module, with the logic engine. Steps in this process will be decided by user dialogs and input from sensors. The user should be presented with the choices ‘previous’ and ‘next’, so they can navigate back and forth through their choices, and/or change some of their answers. The user should have the option of continuing from a previously saved session.

5 WHY CORENA NEEDS THIS

This assignment is made because CORENA has a customer who has requested this functionality for their previously delivered CORENA program. We are supposed to make a solution, which CORENA after hand will implement in their existing S1000D solution.

6 WHY WE WANT TO DO THIS ASSIGNMENT

We think this is a challenging assignment in which we can learn a lot of new programming and scripting techniques and technologies. We also think this is an exciting assignment, with a lot of complexity which we need to understand in order to get this working.

CORENA would also like to offer an office for our disposal at CORENA. This gives us a good opportunity to see how they are working, and it will give us good support during this assignment.

7 OUR VISION FOR THIS PROJECT

We hope, and think that we can go through and finish this assignment and get a good result that CORENA can make use of in their products, as their intention of this assignment also is. We hope to learn a lot of new ways to program/script, and get a good understanding on how it is to work with an agile development method like Scrum.

8 REFERENCE(S)

1. *Assignment Description,* CORENA Norge AS, 2010

2. *International Specification for technical publications utilizing a common source database S1000D*, Technical Publications Specification Maintenance Group (TPSMG), 01-Aug-2008 (Issue 4.0)

GOOGLE WEB TOOLKIT
TECHNOLOGY DOCUMENT

HIBU STUDENT PROJECT 2011
CORENA S1000D 4.0 PROCESS DATA MODULE RENDERER

Arild Oldervoll

Marius Haraldseth

Eirik André Eidså

Olav Brandt

TABLE OF CONTENTS

REVISION HISTORY

Version Number	Date	Changes	Assigned
1	07-Jan-2011	First Official Version	EAE, AOL

1 INTRODUCTION

Our assignment is coming from CORENA Norge A/S. We are going to make a logic engine for support execution of S1000D process Data Modules. This logic engine can take input from users, and/or external applications, such as sensors to decide its next step in a flow of process Data Modules. This logic engine can branch and loop through the selected modules. The logic engine will be an interpreter for process Data Modules. We will make it a stand-alone logic engine in an IETP based web application. The user of the application should be presented with the ability to move back and forth through the steps using two buttons. The user should be able to save the session for future continuation. The logic engine should be according to the International S1000D Specification v 4.0.

1.1 INTENTION OF THIS DOCUMENT

This document is written to give an introduction to the Google Web Toolkit. It will just scratch the surface and give an overview, but it also provides sources of information for a more in depth study.

1.2 SYNONYMS, ACRONYMS AND DEFINITIONS

Synonym	Description
GWT	Google Web Toolkit, framework from Google to write AJAX applications.
GXT	Ext GWT, Java library for building RIA with GWT
RIA	Rich Internet Application, a web application with many similar characteristics as a desktop application.
AJAX	Asynchronous JavaScript and XML, se chapter 2
JavaScript	Very much used scripting language for the web. Not to be confused with Java (programming language)
XML	eXtensible Markup Language – a set of rules for encoding data in a machine-readable format.
CSS	Cascading Style Sheets, used for design of web pages
HTML	HyperText Markup Language, used to sew the webpage together with content. Then use CSS to design it.
IDE	Integrated Development Environment
JSP	Java Server Pages

1.3 AUTHOR(S)

Eirik André Eidså (EAE)

1.4 ASSIGNED

Eirik André Eidså (EAE)

2 AJAX

Before we say too much about GWT, we should take a look at AJAX. The idea behind AJAX is to retrieve data from the server asynchronously in the background without interfering with the display or behaviour of the existing page [5]. In reality, it is a group of interrelated web development techniques used on the client side of interactive web applications. Normally we would use *HttpRequest*, which would refresh the whole page. AJAX on the other hand uses *XMLHttpRequest*, which is an object that is used for communication between the server and the browser. It should also be mentioned that IE6 and earlier versions don´t support this object.

With only one little AJAX application on a webpage, it would work very well with just a few simple JavaScript lines of code. The problem comes with large projects like Google Docs, Wave or Gmail, which could be very difficult to write in JavaScript. This is where GWT comes into play.

3 WHAT IS GWT?

GWT is an open-source, Java-based framework for creating AJAX applications. When developing applications using this framework, you write Java similar to Swing applications, and when compiled, the client-side code is converted to HTML, CSS and JavaScript. GWT also includes many ways to communicate with the server, all asynchronous (which is the definition of AJAX applications). We are going to use the Spring Framework at the server-side, which is one of the main features in GWT 2.1. See reference [1] and chapter 4.5.

3.1 WORKFLOW

The workflow using GWT is very practical and would look something like this:

- Write Java code with use of GWT modules and CSS.
- Test with the help of JUnit and developer mode in Eclipse (No JavaScript yet).
- Compile the project to JavaScript.
- Finalise webpage working in any browser based on HTML, JavaScript and CSS.

3.2 WHAT IS INCLUDED IN GWT? [2]

Hosted Web Browser: Allows you to preview your Java application the same way an end user would see it.

Web Interface Library: Lets you create and use Web browser widgets, such as labels, text boxes, and radio buttons. You program in Java using these widgets, and the compilation process transforms them into HTML equivalents.

Java Emulation Library: Provides JavaScript-equivalent implementations of the most common Java standard classes.

Java-to-JavaScript compiler: Produces the final Web code.

4 WHY USE GWT?

4.1 JAVA AND ECLIPSE

One of the biggest advantages of using GWT is that you develop in Java and in an object oriented world. Google has made a very good Eclipse-plugin that makes it easy to start developing. GWT supports most of the core Java language syntax and semantics, but there are a few differences that you should be aware of, more about that in reference [7]. In addition, GWT brings many new libraries into the project, called modules or widgets. For example, buttons and tab panels that modify their colour and shape to fit an operating system and/or browser.

One thing to remember is that JavaScript is just single threaded. GWT has something, which is not real multithreading, but in most cases does what you need [8]. If true multithreading is required, design the application in a way so that the server-side can handle it.

4.2 TESTING

One of the biggest issues in developing for the web is browser testing and adaption. When creating web applications, support for each type and version of browsers must be developed individually. To adopt all the different browsers often takes a lot more of the developer's time than to create the core application. With GWT this problem is gone. The GWT compiler compiles one special piece of code to each browser automatically so that it has the same appearance and behaviour in all browsers. Now we can talk about "WYSIWYG" (What you see, is what you get)! The developer can spend the time on real development instead of troubleshooting and hacking to get it to work everywhere.

With the use of a good IDE like Eclipse, there are very good possibilities for testing of code with breakpoints and unit testing using JUnit like we are used to in Java.

4.2.1 DEVELOPER MODE

One very nice feature of GWT is that you can test your code almost on the fly in the browser with a GWT developer plugin in most modern browsers. This is called *developer mode*. At this stage we are only using Java and HTML/CSS. The JavaScript is first made at compile time at the end.

4.3 USER EXPERIENCE

The user experience is in general very good in GWT applications. First of all you get rid of problems with back and forward buttons in browsers. Have you noticed that some places it says something like “Do not press backwards – or your credit card will be charged twice!”? In GWT you can control where it is possible to use these buttons, and where it is not. Put in another way; you say when to put a bookmark in the browsing history that the browser uses for this function.

4.4 PERFORMANCE

Performance is a very big key in user experience. GWT has done something that is quite smart. Normally you have to do a lot of server calls to retrieve a webpage. It is all these calls that take most of the time. The browser first calls for the HTML index page, then the browser finds some scripts it has to call, one at the time, then images, style sheets and so on. GWT makes this more efficient so the browser can use fewer calls. Normally you also have to download all the specially made code for all of the browsers. In GWT you just get a package that is made for your specific browser. These two key features help very much when it comes to the performance.

You may also think that auto generated JavaScript code is slower than hand written, but that is actually not true. The performance is usually better or on the same level, dependent on the developer’s skills.

4.5 SERVER SIDE

GWT is very robust when regarding what servers it can work with; it can use many different server solutions. It also works with the Spring Framework [1], JSP, and Jetty/Tomcat, which we are going to use. GWT uses something called *Servlets* that is a part of JSP. A Servlet is used to pass data dynamically between the server and clients. For more information see reference [9] and [10].

5 EXT GWT (GXT)

Ext GWT is an additional Java library that can be used to develop rich internet applications (RIA) with the Google Web Toolkit. CORENA wants us to use this as it is used in their existing technology, but Google does no longer develop the library and it has been superseded by *Smart GWT*.

6 DISADVANTAGES

There are disadvantages with using GWT, but that is the case with most technologies. When it comes to GWT, there are more advantages than disadvantages for our use. Some say that the use of Java is its disadvantage just like it also is an advantage. That is because of the development circle. When you have to compile, it may take a long time, but that time is nothing compared to the time you save and the frustration you are relieved of regarding getting it to work in all browsers. Besides, it is not healthy to sit for too long in front of a computer, so the time can be used to get up and stretch. But still, you don't have to compile all the time to test the code when you develop in "hosted mode".

7 FURTHER READING

7.1 INSTALLATION

You first have to have Eclipse installed for this guide. More detailed guide in reference [6].

7.2 GETTING STARTED

There are some in-depth tutorials from Google. This is good to get the feeling of how this works. See reference [3].

7.3 DOCUMENTATION

This is one of the few open source projects that are well documented from the creator. The documentation is really good. See reference [4].

8 REFERENCE(S)

1. Technical document - Spring Framework, Ameo, 05-Jan-2010
2. http://www.linux.com/archive/feature/124163 (last visited 01-Dec-2010)
3. http://code.google.com/intl/no-NO/webtoolkit/doc/latest/tutorial/index.html (last visited 01-Dec-2010)
4. http://code.google.com/intl/no-NO/webtoolkit/doc/latest/DevGuide.htm (last visited 01-Dec-2010)
5. http://www.wmps.com/blog/website-design/website-build/why-use-gwt-for-ajax-applications/ (last visited 1-Dec-2010)
6. http://code.google.com/intl/no-NO/eclipse/docs/install-eclipse-3.6.html (last visited 01-Dec-2010)
7. http://code.google.com/intl/no-NO/webtoolkit/doc/latest/DevGuideCodingBasicsCompatibility.html (last visited 01-Dec-2010)
8. http://stackoverflow.com/questions/2590850/threading-in-gwt-client (last visited 01-Dec-2010)
9. http://en.wikipedia.org/wiki/Java_Servlet (last visited 01-Dec-2010)
10. http://en.wikipedia.org/wiki/JavaServer_Pages (last visited 01-Dec-2010)
11. http://code.google.com/p/gwt-ext/ (last visited 07-Jan-2010)