Rafael Valiev
Kasatkin Timur
Gregory Yermolaev

Innopolis University

myCrashVariant Design Document - v 0.0.5 -

Sunday 26th March, 2017 - 23:48

Springer

Contents

1 Introduction . . . **7**
1.1 Overview . . . 7
1.2 Purpose and recipients of the document . . . 7
1.3 Definitions, acronyms and abbreviations . . . 7
1.4 Document structure . . . 7

2 Analysis Models . . . **9**
2.1 Concept Model . . . 9
2.2 Environment Model . . . 9

3 Technological frameworks . . . **11**
3.1 Java . . . 11
3.2 Eclipse . . . 11
3.3 MySQL . . . 11
3.4 MySQL Workbench . . . 11
3.5 JavaFX . . . 11
3.6 Scene Builder . . . 11
3.7 Remote Method Invocation (RMI) . . . 12

4 System Architecture . . . **13**
4.1 Deployment view . . . 13
4.2 Implementation view . . . 14
4.2.1 Component lu.uni.lassy.excalibur.examples.icrash.desktop.java.commons . . . 14
4.2.2 Component lu.uni.lassy.excalibur.examples.icrash.desktop.java.client . . . 15
4.2.3 Component lu.uni.lassy.excalibur.examples.icrash.desktop.java.database . . . 16
4.2.4 Component lu.uni.lassy.excalibur.examples.icrash.desktop.java.server . . . 16
4.3 UI Processing view . . . 17
4.3.1 UI Processing view for system operation oeSystemOperation1 . . . 17
4.3.2 UI Processing view for system operation oeSystemOperation2 . . . 17
4.3.3 UI Processing view for system operation oeSystemOperation3 . . . 17
4.4 Non-functional runtime concerns . . . 17
4.4.1 Performance . . . 17
4.4.2 Concurrency and Parallelism . . . 17
4.4.3 Scalability . . . 17

5 Detailed design . . . **19**
5.1 Interaction Model . . . 19
5.1.1 oeSystemOperation1 . . . 19
5.1.2 oeSystemOperation2 . . . 19
5.1.3 oeSystemOperation3 . . . 19
5.2 Design Class Model . . . 19
5.2.1 Design Class Model view1 . . . 20
5.2.2 Design Class Model view2 . . . 20
5.2.3 Design Class Model view3 . . . 20

6 Known limitations . . . **21**

6.1 Issue 1 21
6.2 Issue 2 21

7 Conclusion 23
References 25

List of Figures

4.1 Deployment Diagram . . . 13
4.2 High level Component diagram . . . 14
4.3 Commons Project Component diagram . . . 14
4.4 Client Project Component diagram . . . 15
4.5 Database Project Component diagram . . . 16
4.6 Server Project Component diagram . . . 16

Listings

Chapter 1
Introduction

1.1 Overview

The myCrashVariant software system is intended to be used for secure crisis handling and reporting. The myCrashVariant software system belongs to the Crisis Managment Systems Domain. It is intended to be used for secure and convenient crisis reporting by victim and witnesses (with a possibility to do so anonymously) and handling by coordinators.

1.2 Purpose and recipients of the document

This document is a software design document, which is intended to describe and summarize main decisions made about architecture and system design of myCrashVariant software system.

The audience of this document are employees of the development company (ADC), who are responsible for myCrashVariant software system production and delivering. Document should be consulted during actual system implementation and deployment and delivered software should meet all the decisions described in the document.

1.3 Definitions, acronyms and abbreviations

All the terms required for proper document understanding:

Model-View-Controller a common design pattern followed to design the Graphical User Interfaces in a consistent way.

An actor is a person, organization, or external system, which interacts with a system.

System Operation system functionality, which could be inviked by an actor passing a message.

Deployment View describes components topology on a physical layer.

Implementation View describes software system components layer.

UI Processing View describes the interaction to a system, which allows invocation of system operations.

Abstract Actor an actor which could not have an inctance.

1.4 Document structure

This document contains several chapters (sections), each of those describes a myCrashVariant design decisions from different points of view.

Chapter 1 contains document overview, description of the document purpose and recipients and definitions, acronyms and abbreviations used in a document.

Chapter 2 provides decription of decision made during analysis phase, in particular, software system types and system actors.

Chapter 3 describes in detail technological frameworks used for myCrashVariant production.

Chapter 4 describes both hardware, required for myCrashVariant deployment and running, as well as behaviour of the software during the runtime.

Chapter 5 in detail describes all system operations.

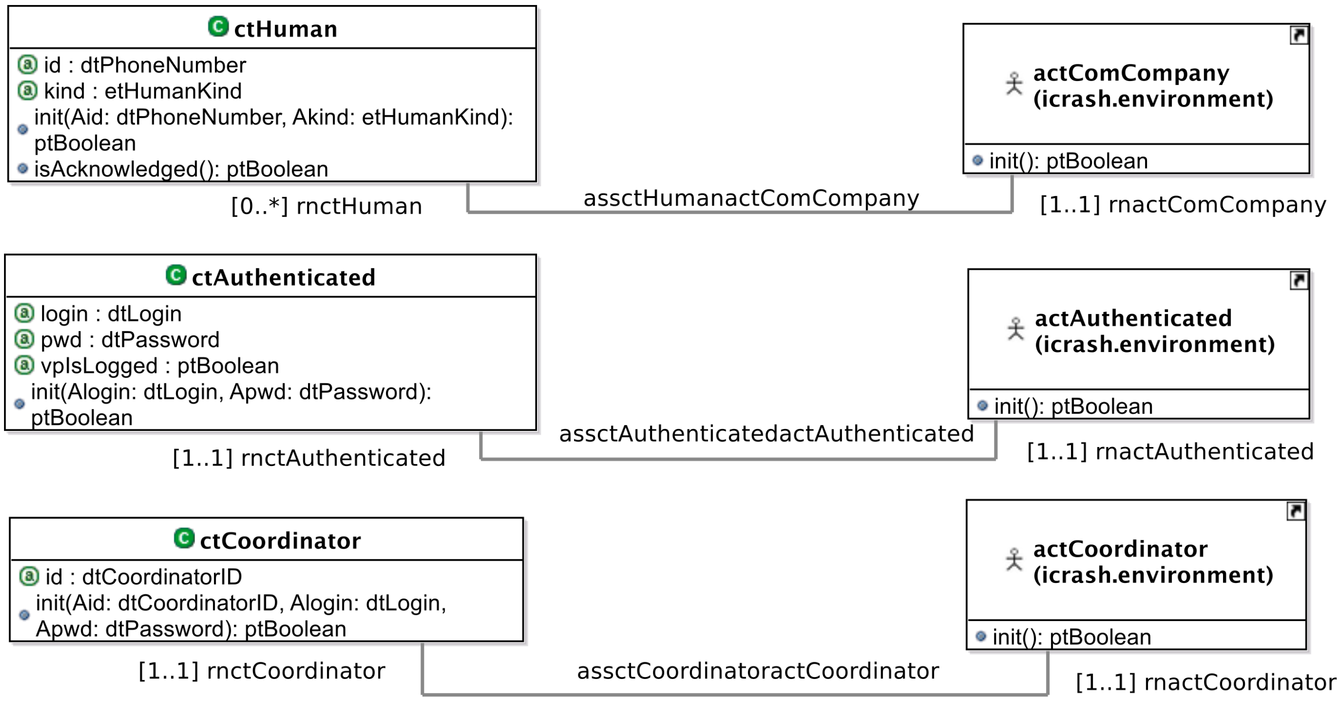
Chapter 6 lists all known problems of myCrashVariant and describes them in detail.

Chapter 7 summarizes all the information retrieved during design and implementation phase.

Chapter 2
Analysis Models

This chapter provides a general overview of the main concepts gathered during the analysis phase, in particular those concerning the software system types (i.e. classes, datatypes, and enumerations), as well as the actors that interact with the software system through their interfaces. Figures included in the Messir Requirement Document that correspond to the the Concept Models and the Environment Models could be also included in this chapter, as a means of synthesizing what are the requirements to which the design is supposed to sketch a solution.

2.1 Concept Model


ctHuman
id : dtPhoneNumber
kind : etHumanKind
init(Aid: dtPhoneNumber, Akind: etHumanKind): ptBoolean
isAcknowledged(): ptBoolean
[0..*] rnctHuman
assctHumanactComCompany
actComCompany (icrash.environment)
init(): ptBoolean
[1..1] rnactComCompany
ctAuthenticated
login : dtLogin
pwd : dtPassword
vpIsLogged : ptBoolean
init(Alogin: dtLogin, Apwd: dtPassword): ptBoolean
[1..1] rnctAuthenticated
assctAuthenticatedactAuthenticated
actAuthenticated (icrash.environment)
init(): ptBoolean
[1..1] rnactAuthenticated
ctCoordinator
id : dtCoordinatorID
init(Aid: dtCoordinatorID, Alogin: dtLogin, Apwd: dtPassword): ptBoolean
[1..1] rnctCoordinator
assctCoordinatoractCoordinator
actCoordinator (icrash.environment)
init(): ptBoolean
[1..1] rnactCoordinator

2.2 Environment Model

Chapter 3
Technological frameworks

3.1 Java

Due to the need for developing software system some programming language should be selected. Java has been chosen.

3.2 Eclipse

It was required to develop the system inside environment which supports Java programmers in most part of their implementation issues (highlighting syntax errors, project build, etc.). This environment should also be provided a on free basis. Thus, the open-source Integrated Development Environment (IDE) for Java programmers called Eclipse was chosen.

3.3 MySQL

It was required to persistently store system's actors' information (information about alerts, crises, etc.). Thus, some database engine should be chosen. MySQL has been selected.

3.4 MySQL Workbench

It was required to find some database toolkit to conveniently and effectively manage MySQL database engine. Thus, MySQL Workbench was selected.

3.5 JavaFX

It was required to choose technological framework for GUI development. JavaFX was chosen due to good separation of GUI and logic, and rich opportunities for creating GUI.

3.6 Scene Builder

It was required to find some tool to quickly define GUI layout with JavaFX. Scene Builder was chosen.

3.7 Remote Method Invocation (RMI)

It was required to determine protocol to be used for interaction between GUI client and server. Remote Method Invocation (RMI) protocol was chosen, because it is created exclusively for Java, provides good abstraction level (no technological aspects of network communication such as in sockets, http) and has well documented by Oracle company.

Chapter 4
System Architecture

This chapter presents information concerning the architecture of the software system both from the static and dynamic viewpoints. The static viewpoint focuses on the physical architecture (hardware) required to deploy and run the software system along with the manner in which the components that make such software system are grouped. On the other hand, the dynamic viewpoint focuses on the behaviour of the software system at runtime.

The static information is presented through the Deployment View and the Implementation View. The dynamic aspects of the system are presented by means of the UI Processing View.

4.1 Deployment view

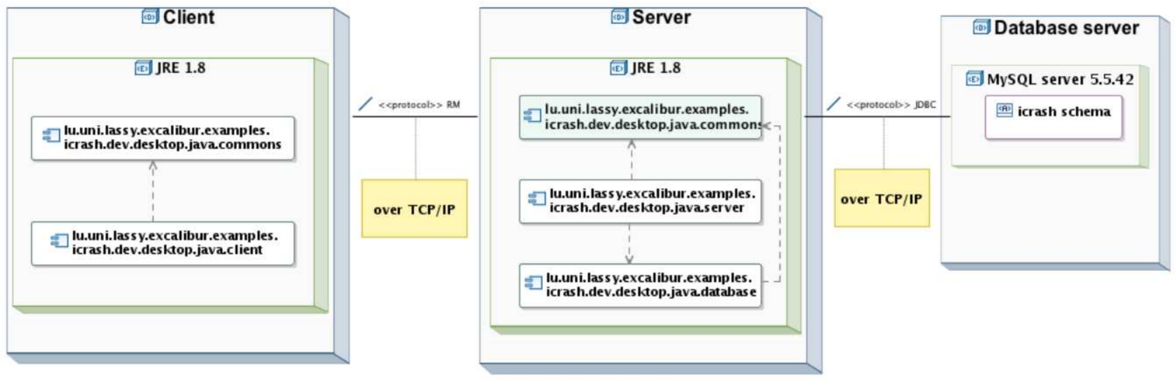

Client
JRE 1.8
lu.uni.lassy.excalibur.examples.icrash.dev.desktop.java.commons
lu.uni.lassy.excalibur.examples.icrash.dev.desktop.java.client
<<protocol>> RMI
over TCP/IP
Server
JRE 1.8
lu.uni.lassy.excalibur.examples.icrash.dev.desktop.java.commons
lu.uni.lassy.excalibur.examples.icrash.dev.desktop.java.server
lu.uni.lassy.excalibur.examples.icrash.dev.desktop.java.database
<<protocol>> JDBC
over TCP/IP
Database server
MySQL server 5.5.42
icrash schema

Fig. 4.1 Deployment Diagram

The system consists of 3 processing nodes:

- Client node represents client's machine with launched desktop GUI client, whose implementation consist of two software components.
- Server node represents main system's machine with implementation of core part of the system launched. Software components include one for interaction with database.
- On database server node there is a MySQL DBMS server launched used to permanently store information about crises.

Client and Server node interconnected using Java RMI protocol, which is built over TCP/IP network protocol.

Server and Database server nodes interconnected using JDBC (Java DataBase Connectivity) protocol, which is again works over TCP/IP.

4.2 Implementation view

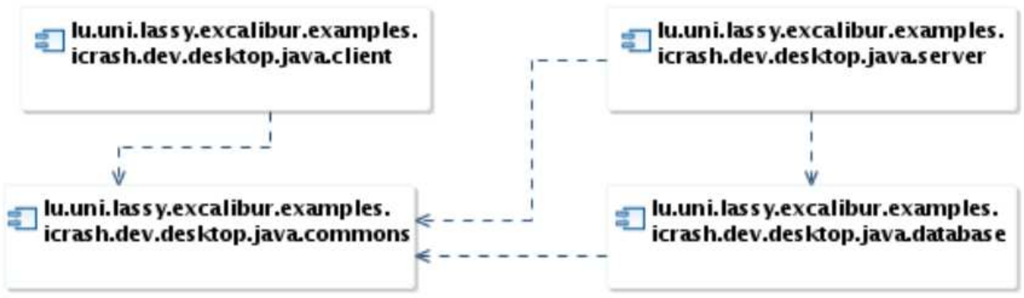

lu.uni.lassy.excalibur.examples.
icrash.dev.desktop.java.client
lu.uni.lassy.excalibur.examples.
icrash.dev.desktop.java.server
lu.uni.lassy.excalibur.examples.
icrash.dev.desktop.java.commons
lu.uni.lassy.excalibur.examples.
icrash.dev.desktop.java.database

Fig. 4.2 High level Component diagram

The system composed of 4 high level components:

- ***.commons** consist of functionality used by all other components. It includes system types and some util functions.
- ***.client** is an implementation of desktop GUI application, used by system actors to interact with the system.
- ***.database** is used to interact with system's database server.
- ***.server** is an implementation of the system's core part - server which accepts commands from GUI client and provides system's main functionality.

  ***.database** component is used by ***.server** one to performs operation on data storage.

4.2.1 Component lu.uni.lassy.excalibur.examples.icrash.desktop.java.commons

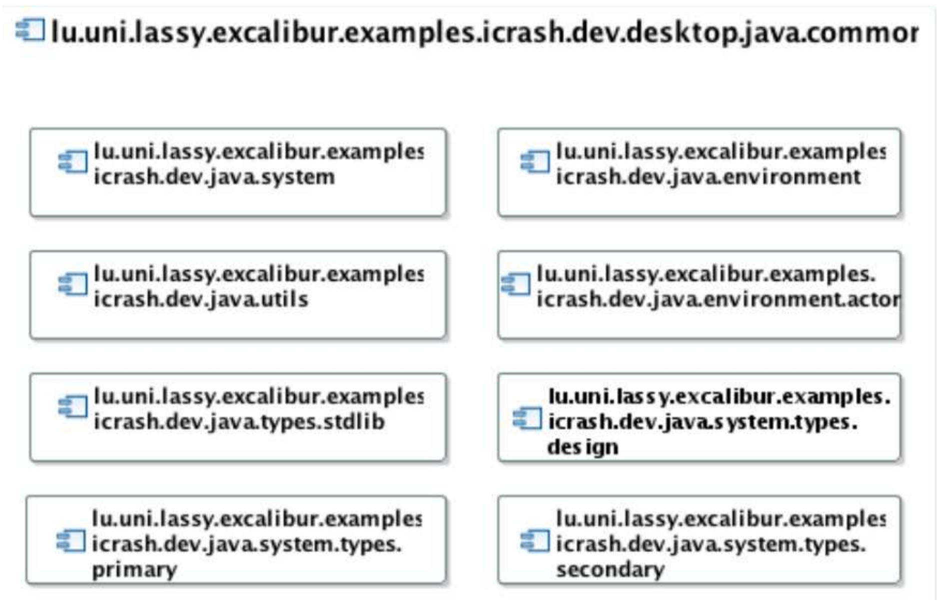

lu.uni.lassy.excalibur.examples.icrash.dev.desktop.java.commor
lu.uni.lassy.excalibur.examples
icrash.dev.java.system
lu.uni.lassy.excalibur.examples
icrash.dev.java.environment
lu.uni.lassy.excalibur.examples
icrash.dev.java.utils
lu.uni.lassy.excalibur.examples.
icrash.dev.java.environment.actor
lu.uni.lassy.excalibur.examples
icrash.dev.java.types.stdlib
lu.uni.lassy.excalibur.examples.
icrash.dev.java.system.types.
design
lu.uni.lassy.excalibur.examples
icrash.dev.java.system.types.
primary
lu.uni.lassy.excalibur.examples
icrash.dev.java.system.types.
secondary

Fig. 4.3 Commons Project Component diagram

This components contains common functionality used by all other high level components.
.system.types. components contain system's types.
***.types.stdlib** component includes standart library of datatypes.
***.utils** provides some util functions.
***.environment.actors** contains types representing system actors.

4.2.2 Component lu.uni.lassy.excalibur.examples.icrash.desktop.java.client

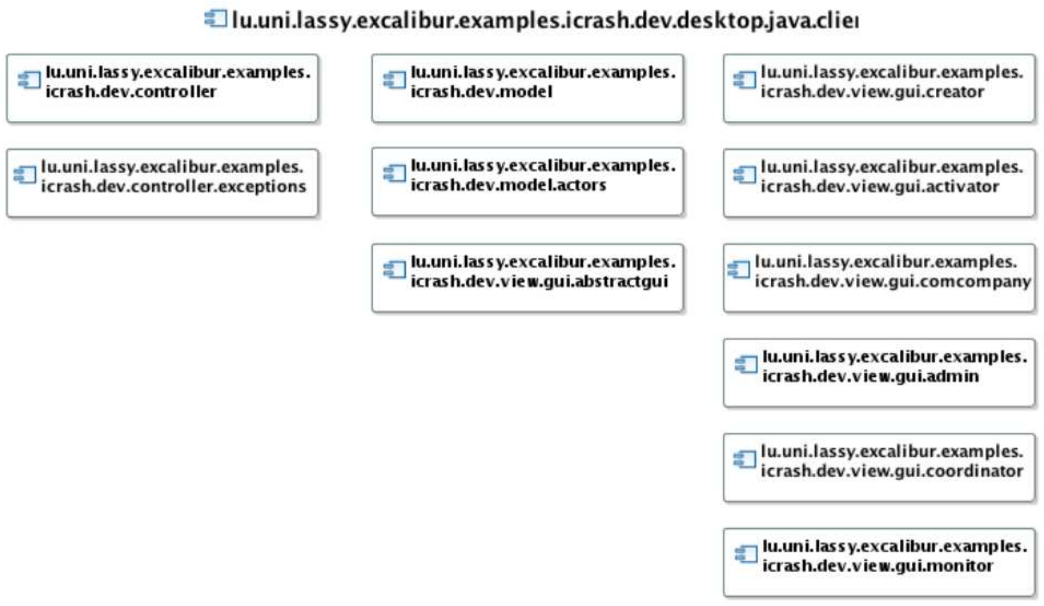

lu.uni.lassy.excalibur.examples.icrash.dev.desktop.java.clie
lu.uni.lassy.excalibur.examples.icrash.dev.controller
lu.uni.lassy.excalibur.examples.icrash.dev.model
lu.uni.lassy.excalibur.examples.icrash.dev.view.gui.creator
lu.uni.lassy.excalibur.examples.icrash.dev.controller.exceptions
lu.uni.lassy.excalibur.examples.icrash.dev.model.actors
lu.uni.lassy.excalibur.examples.icrash.dev.view.gui.activator
lu.uni.lassy.excalibur.examples.icrash.dev.view.gui.abstractgui
lu.uni.lassy.excalibur.examples.icrash.dev.view.gui.comcompany
lu.uni.lassy.excalibur.examples.icrash.dev.view.gui.admin
lu.uni.lassy.excalibur.examples.icrash.dev.view.gui.coordinator
lu.uni.lassy.excalibur.examples.icrash.dev.view.gui.monitor

Fig. 4.4 Client Project Component diagram

This component contains desktop GUI application used to interact with the system. The component is built using MVC architecture.
.controller. components contain Controller of MVC architecture, which contain logic of business work.
.model. components represents Model part of MVC.
.view.gui. components represents View part of MVC, i.e. application's GUI.

4.2.3 Component lu.uni.lassy.excalibur.examples.icrash.desktop.java.database

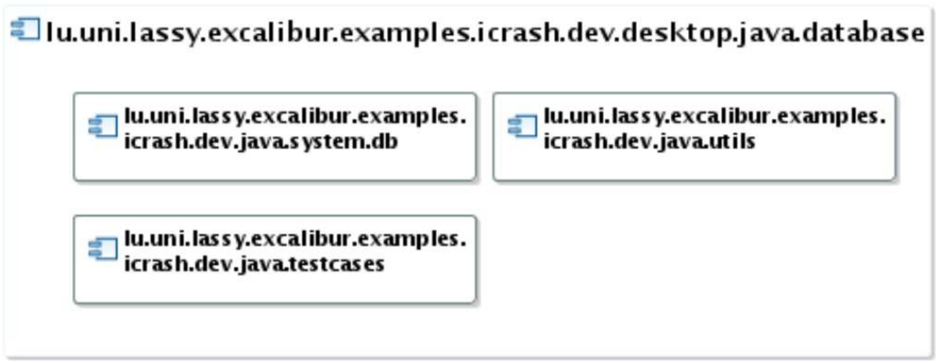

lu.uni.lassy.excalibur.examples.icrash.dev.desktop.java.database
lu.uni.lassy.excalibur.examples. icrash.dev.java.system.db
lu.uni.lassy.excalibur.examples. icrash.dev.java.utils
lu.uni.lassy.excalibur.examples. icrash.dev.java.testcases

Fig. 4.5 Database Project Component diagram

The component allows to interact with system's data storage (database server). E.g. add new coordinator, get existing crises, etc.

4.2.4 Component lu.uni.lassy.excalibur.examples.icrash.desktop.java.server

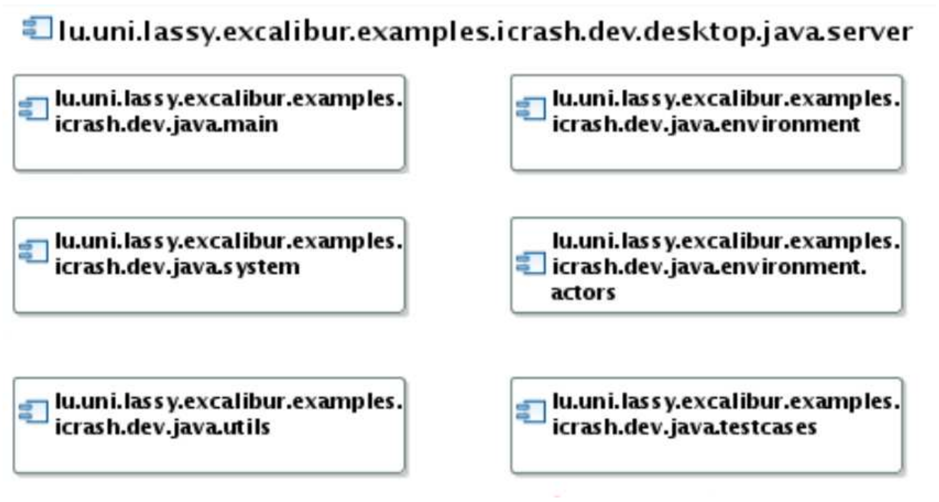

lu.uni.lassy.excalibur.examples.icrash.dev.desktop.java.server
lu.uni.lassy.excalibur.examples. icrash.dev.java.main
lu.uni.lassy.excalibur.examples. icrash.dev.java.environment
lu.uni.lassy.excalibur.examples. icrash.dev.java.system
lu.uni.lassy.excalibur.examples. icrash.dev.java.environment. actors
lu.uni.lassy.excalibur.examples. icrash.dev.java.utils
lu.uni.lassy.excalibur.examples. icrash.dev.java.testcases

Fig. 4.6 Server Project Component diagram

The component represents the system's core part - server which provides system's main functionality (handling crises) and answers to requests from GUI client.
***.main** used to launch the server.
***.system** contains system's main functions, built using other software components.
***.testcases** tests used to check server's availability.
.environment. - implementation of main system's actors representations.

4.3 UI Processing view

A UI Processing View is aimed at explaining the required message exchanges to achieve the launching of a system operation (specified in the Messir Analysis Document). These required message exchanges (which are not specified in the Messir Analysis Document) make part of the user interface (UI). Thus, the main interest of a UI Processing View is to describe the design choices made at the UI level, such that a system operation is launched. The description of a UI Processing View is given by means of a UML Sequence Diagram.

A complete Design Document should contain a UI Processing View for each non-proactive system operation specified in the Messir Analysis Document, as such kind of system operations are launched by actors through UIs that allows them to make so.

4.3.1 UI Processing view for system operation oeSystemOperation1

TODO

4.3.2 UI Processing view for system operation oeSystemOperation2

TODO

4.3.3 UI Processing view for system operation oeSystemOperation3

TODO

4.4 Non-functional runtime concerns

The description of the runtime processes should be complemented with free textual information regarding concurrency, distribution, performance and scalability aspects.

4.4.1 Performance

TODO

4.4.2 Concurrency and Parallelism

TODO

4.4.3 Scalability

TODO

Chapter 5

Detailed design

Provide an introduction to the proposed design. This introduction should help the reader to understand the design choices made.

5.1 Interaction Model

An Interaction Model describes how each System Operation (that appears in the Operation Model of the Messir Requirements Analysis Model) is designed to meet its specification. The design description of each system operation must be focused on the messages exchanged between the different first-class objects (i.e. instances of classes either included in Concept Model or introduced as result of a design choice). An Interaction Model is modeled as a UML Sequence Diagram.

5.1.1 oeSystemOperation1

TODO

5.1.2 oeSystemOperation2

TODO

5.1.3 oeSystemOperation3

TODO

5.2 Design Class Model

The Design Class Model is composed of the contents of all design classes (i.e. every class appearing in at least one Interaction Model), all the navigable associations between design classes, and the inheritance structure. The description of each class must contain its attributes and operations. The Design Class Model is modeled as a UML Class Diagram.

It is advised to split the Design Class Model into multiple views as such model may become pretty large.

5.2.1 Design Class Model view1

TODO

5.2.2 Design Class Model view2

TODO

5.2.3 Design Class Model view3

TODO

Chapter 6
Known limitations

All known and non solved issues (like bugs, missing functionalities, abnormal behavior, etc.) should be precisely stated and described.

6.1 Issue 1

TODO

6.2 Issue 2

TODO

Chapter 7
Conclusion

TODO

References