Farrukh Kholmukhamedov
Rustam Tukhvatov

Innopolis University

myCrashVarian.G01: iCrash system improvements Design Document - v 0.1 -

Monday 24th April, 2017 - 22:01

Springer

Contents

List of Figures

Listings

Chapter 1
Introduction

1.1 Overview

This software system is aimed for car crash crisis management.

Excerption from the analysis document: the ***iCrash*** is a simple system dedicated to any person who wants to inform of a car crash crisis situation in order to allow for crisis handling. At anytime and anywhere, anyone can be the witness or victim of a car crash and might be in a situation allowing for alerting this crisis. The ***iCrash*** system has for objectives to support crisis declaration and secure administration and crisis handling by the ***iCrash*** professional users.

As any software system, ***iCrash*** requires some improvements. This document covers two new features: biometric authentication and information diffusion.

1.2 Purpose and recipients of the document

This document is a design document. The aim of this document is to provide an example of how the design of a particular software system should be documented.

The recipient of this document is the development company (ADC) in charge of delivering the software system. The company's developers are expected to use this document as the basis for carrying out the actual development and deployment of the product (i.e. implementation, testing and maintenance).

1.3 Definitions, acronyms and abbreviations

Biometrics or **biometry** — the measurement and analysis of unique physical or behavioral characteristics (as fingerprint or voice patterns) especially as a means of verifying personal identity

1.4 Document structure

This document is organised as follows: Section 2 provides a general overview of the main concepts gathered during the analysis phase, in particular those concerning the software system abstract types, as well as the actors that interact with the software system through their interfaces.

The technologies used not only during the design and development phase, but also those required to make the software system runnable are presented in Section 3.

The architecture of the software system to be implemented and deployed is described in Section 4. This section presents the components of the software system architecture along with their interactions, both from the static and dynamic viewpoints.

The detailed design of each System Operation is given in Section 5, whereas Section 6 enumerates the current limitations of the software system at the writing time of this document.

Next, Section **??** presents the different test cases used to verify the correct behaviour of the software system's functional and not functional requirements.

Finally, Section 7 draws the conclusion achieved during the design and implementation of the software system.

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

2.1.1 Primary types - Class types descriptions

The diagram 2.1 shows a global view of the ***iCrash*** system primary types.

Fig. 2.1 Concept model - PrimaryTypes-Classes

- **ctAdministrator** Used to caracterize internally the entity that is responsible of administrating the *iCrash* system.
- **ctAlert** Used to model crisis alerts sent by any human having communication capability using communication companies belonging to the system's environment.
- **ctAuthenticated** Used to model system's representation about actors that need to authenticate to access some specific functionalities.
- **ctCoordinator** Used to model system's representation about the actors that have the responsibility to handle alerts and crisis.
- **ctCrisis** Used to model crisis that are infered from the reception of at least one alert message. Crisis aer entities that are handled by the *iCrash* system.
- **ctHuman** Used to model system's representation about the indirect actors that has alerted of potential crisis.

- **clReport** Used to model a report of the occurred crisis. It shows current state of the accident to send the information to appropriate organizations.
- **ctState** Used to model the system. There is only one instance at any state of the abstract machine after creation.

2.1.2 Primary types - Datatypes types descriptions

The diagram 2.2 shows the basic data types of the *iCrash* system primary datatypes.

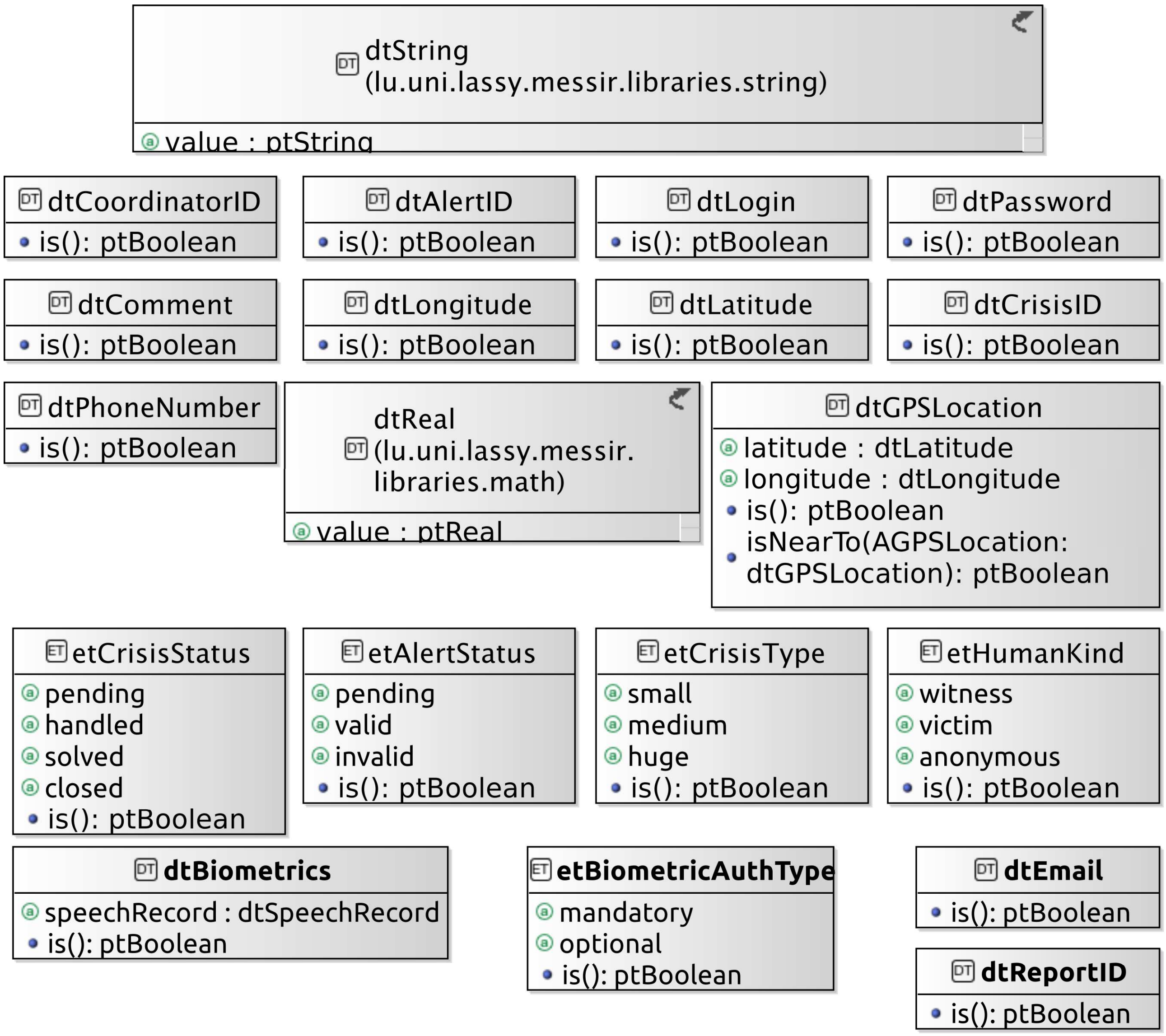

Fig. 2.2 Concept model - PrimaryTypes-Classes

2.1.2.1 Datatypes

- **dtAlertID** A string used to identify alerts.

- **dtComment** A datatype made of a string value used to receive,store and send textual information about crisis and alerts.
- **dtCoordinatorID** A string used to identify coordinators.
- **dtCrisisID** A string used to identify crisis.
- **dtGPSLocation** Used to define coordinates of geograpical positions on earth. It is defined a couple made of a latitude and a longitude.
- **dtLatitude** Used to define a latitude value of a geograpical positions on earth.
- **dtLogin** A login string used to authentify an ***iCrash*** user
- **dtLongitude** Used to define a longitude value of a geograpical positions on earth.
- **dtPassword** A password string used to authentify an ***iCrash*** user
- **dtPhoneNumber** A string used to store the phone number from the human declaring the crisis or the alert.
- **dtEmail** A string used to store the email address of the recipient of the crisis report.
- **ReportID** A string used to identify reports.

2.1.2.2 Enumerations

- **etAlertStatus** This type is used to indicate the different validation status of an alert.
- **etCrisisStatus** This type is used to indicate the different handling status of a crisis.
- **etCrisisType** This type is used to indicate the different types of a crisis.
- **etHumanKind** This type is used to indicate the kind of human that informs about a car crash crisis.
- **etBiometricAuthType** This type is used to indicate the different possibilities of biometric authentication.

2.1.3 Secondary types - Datatypes types descriptions

2.1.3.1 Datatypes

- **dtSMS** A datatype made of a string value used to send textual information to human mobile devices.
- **dtSpeechRecord** A datatype made of a record value used to determine human's voice

2.2 Environment Model

The diagram 2.3 shows a global view of the ***iCrash*** system environment model.

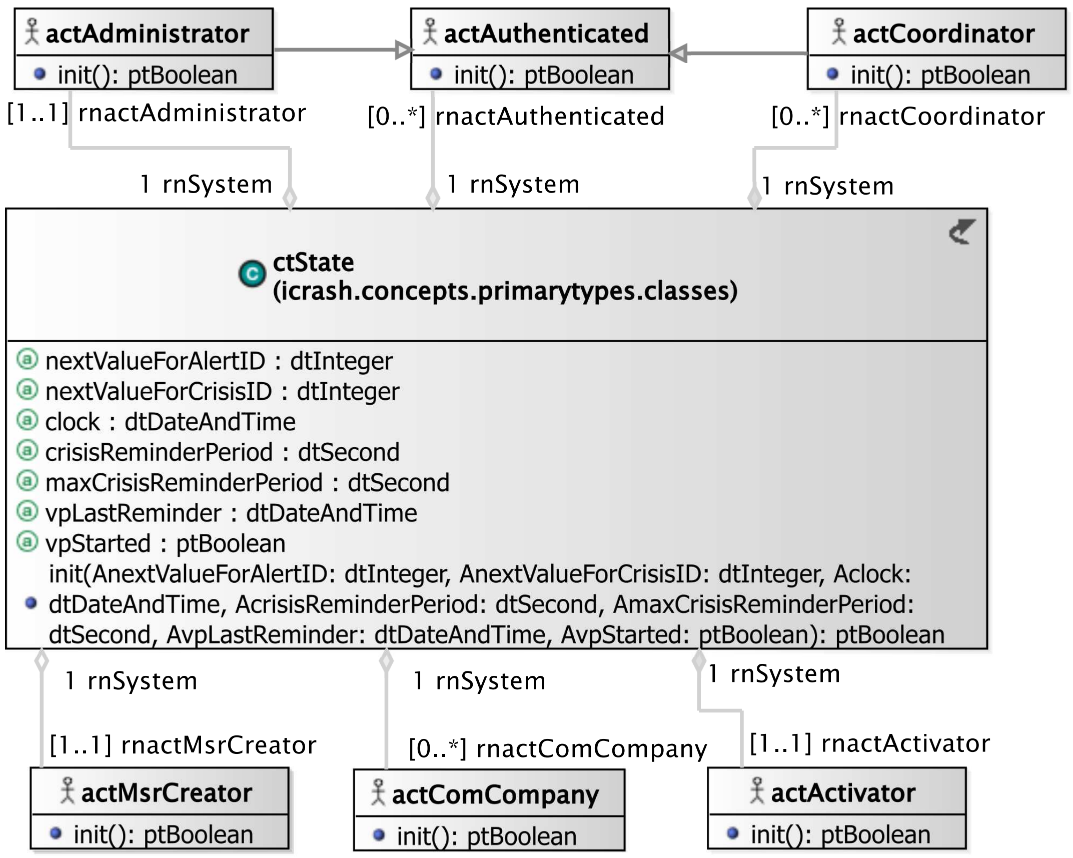
actAdministrator
init(): ptBoolean
actAuthenticated
init(): ptBoolean
actCoordinator
init(): ptBoolean
[1..1] rnactAdministrator
[0..*] rnactAuthenticated
[0..*] rnactCoordinator
1 rnSystem
1 rnSystem
1 rnSystem
ctState
(icrash.concepts.primarytypes.classes)
nextValueForAlertID : dtInteger
nextValueForCrisisID : dtInteger
clock : dtDateAndTime
crisisReminderPeriod : dtSecond
maxCrisisReminderPeriod : dtSecond
vpLastReminder : dtDateAndTime
vpStarted : ptBoolean
init(AnextValueForAlertID: dtInteger, AnextValueForCrisisID: dtInteger, Aclock: dtDateAndTime, AcrisisReminderPeriod: dtSecond, AmaxCrisisReminderPeriod: dtSecond, AvpLastReminder: dtDateAndTime, AvpStarted: ptBoolean): ptBoolean
1 rnSystem
1 rnSystem
1 rnSystem
[1..1] rnactMsrCreator
[0..*] rnactComCompany
[1..1] rnactActivator
actMsrCreator
init(): ptBoolean
actComCompany
init(): ptBoolean
actActivator
init(): ptBoolean

Fig. 2.3 Environment Model - Global View

- **actActivator Actor** Represents a logical actor for time automatic message sending based on system's or environment status.
- **actAdministrator Actor** Represents an actor responsible of administration tasks for the *iCrash* system.
- **actAuthenticated Actor** Abstract actor providing reusable input and output interfaces for actors that need to authenticate themselves.
- **actComCompany Actor** Represents the communication company stakeholder ensuring the input/ouput of textual messages with humans having communication devices.
- **actCoordinator Actor** Represents actor responsible of handling one or several crisis for the *iCrash* system.
- **actMsrCreator Actor** Represents the creator stakeholder in charge of state and environment initialization.

Chapter 3
Technological frameworks

This chapter provides a description of the technological frameworks used during the development phase of **iCrash** system.

3.1 Java Programming Language

Java is a programming language. Java is general-purpose multi-paradigm, class-based language with high portability of the software. Java is a high-level programming language originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.

3.2 MySQL

MySQ is an open-source relational database management system (RDBMS). MySQL is the world's most popular open source database. With its proven performance, reliability, and ease-of-use, MySQL has become the leading database choice for web-based applications, used by high profile web properties including Facebook, Twitter, YouTube, and all five of the top five websites*. Additionally, it is an extremely popular choice as embedded database, distributed by thousands of ISVs and OEMs.

3.3 Eclipse

Eclipse is an integrated development environment (IDE) used in computer programming, and is the most widely used Java IDE. It contains a base workspace and an extensible plug-in system for customizing the environment. Eclipse is written mostly in Java and its primary use is for developing Java applications, but it may also be used to develop applications in other programming languages.

3.4 Git

labelsec:Git Git is a version control system (VCS) for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for software development,but it can be used to keep track of changes in any files. As a distributed revision control system it is aimed at speed, data integrity, and support for distributed, non-linear workflows.

3.5 LaTeX

LaTeX is a document preparation system for high-quality typesetting. It is most often used for medium-to-large technical or scientific documents but it can be used for almost any form of publishing.

3.6 Excalibur

Excalibur is a tool supporting the Messir methodology, a Scientific Method for the Software Engineering Master, used in Software Engineering Lectures at bachelor and master levels.

Excalibur tool covers the phase of Requirements Analysis and its main features are requirements analysis specification (its own DSL), requirements report generation (latex/pdf) and requirements simulation (prolog). It relies on Eclipse technologies as XText for textual specification and Sirius for graphical views of the textual specifications

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

The aim of the Deployment View is to describe the different processing nodes that compose the deployment infrastructure and how they are interconnected. A processing node corresponds to a piece of hardware aimed at executing either the whole software system or a sub-part of it.

The **iCrash** has Client-Server architecture. The diagram 5.2 represents the components that the system consists of. The Client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. To store the data is used MySQL DBMS server.

To connection Server and Database server used JDBC (Java DataBase Connectivity) protocol. The protocol is built over TCP/IP network protocol. Client and Server node use Java RMI protocol to connection. The protocol is also built over TCP/IP network protocol.

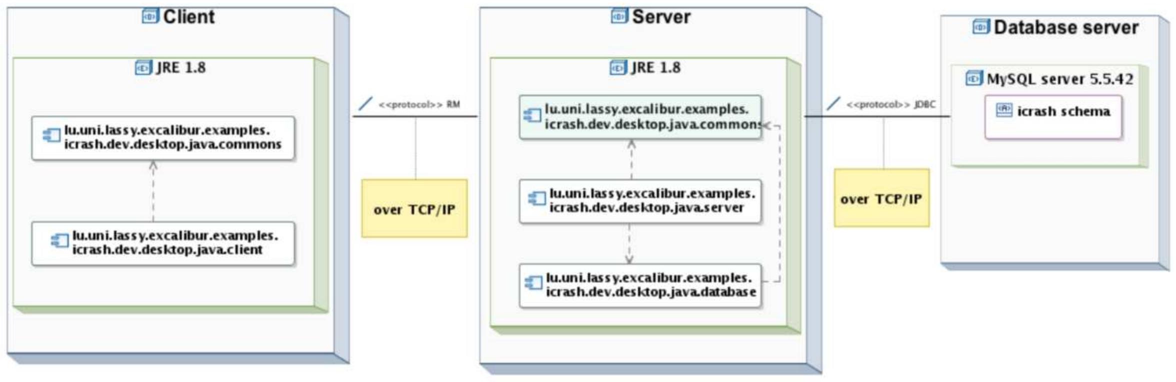
Client
JRE 1.8
lu.uni.lassy.excalibur.examples.icrash.dev.desktop.java.commons
lu.uni.lassy.excalibur.examples.icrash.dev.desktop.java.client
<<protocol>> RMI
over TCP/IP
Server
JRE 1.8
lu.uni.lassy.excalibur.examples.icrash.dev.desktop.java.commons
lu.uni.lassy.excalibur.examples.icrash.dev.desktop.java.server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

4.2 Implementation view

The Implementation View describes each software system component and how they are organised and combined to make the targeted software system.

4.2.1 Client

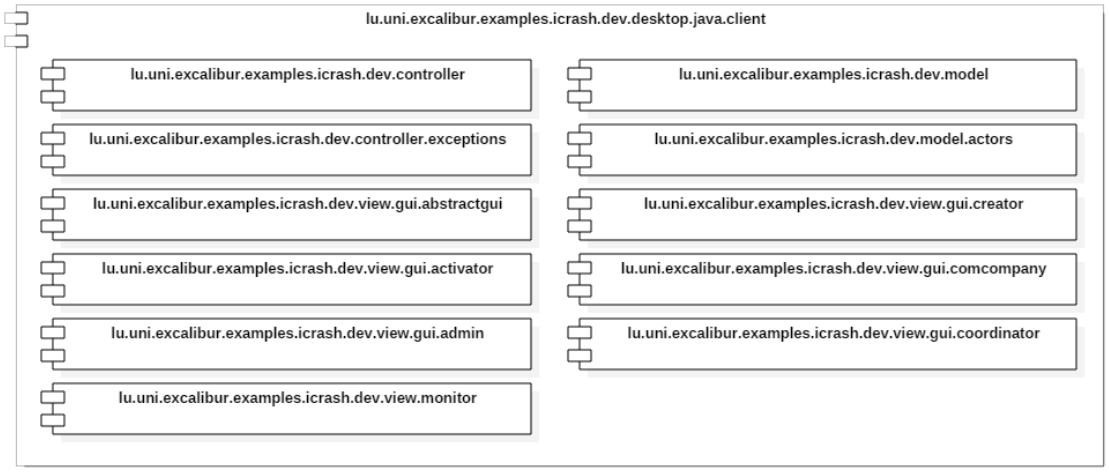
lu.uni.excalibur.examples.icrash.dev.desktop.java.client
lu.uni.excalibur.examples.icrash.dev.controller
lu.uni.excalibur.examples.icrash.dev.model
lu.uni.excalibur.examples.icrash.dev.controller.exceptions
lu.uni.excalibur.examples.icrash.dev.model.actors
lu.uni.excalibur.examples.icrash.dev.view.gui.abstractgui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

Fig. 4.2 Client component

4.2.2 Server

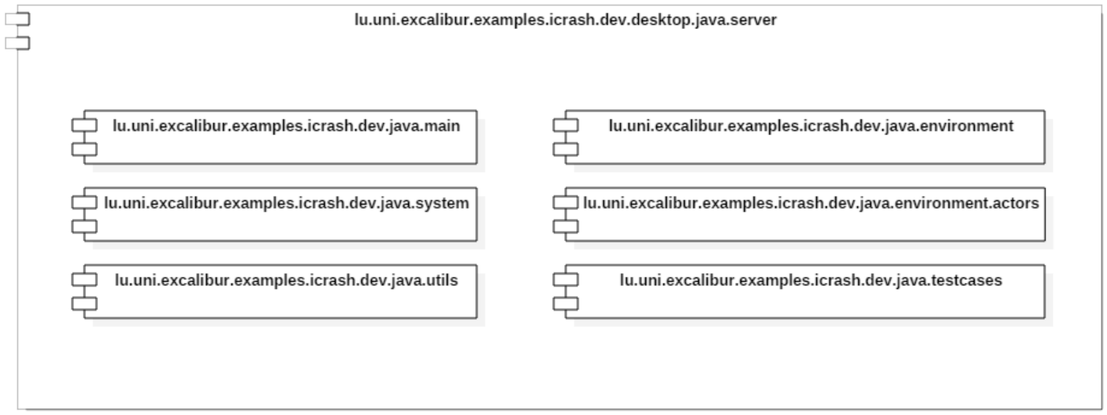
lu.uni.excalibur.examples.icrash.dev.desktop.java.server
lu.uni.excalibur.examples.icrash.dev.java.main
lu.uni.excalibur.examples.icrash.dev.java.environment
lu.uni.excalibur.examples.icrash.dev.java.system
lu.uni.excalibur.examples.icrash.dev.java.environment.actors
lu.uni.excalibur.examples.icrash.dev.java.utils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

Fig. 4.3 Server component

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.

4.4 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

Java programming language allows achieve high performance through the optimization of the code in the required environment.

4.4.2 Concurrency and Parallelism

The application may handle requests from different active actors in parallel, it provided by Java and MySQL database provide parallelism of the system to handle requests in parallel.

4.4.3 Scalability

The system supports vertical scalability that means, that additional performance boost may be achieved by using more powerful hardware. Horizontal scalability is possible only for database.

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 oeLogin

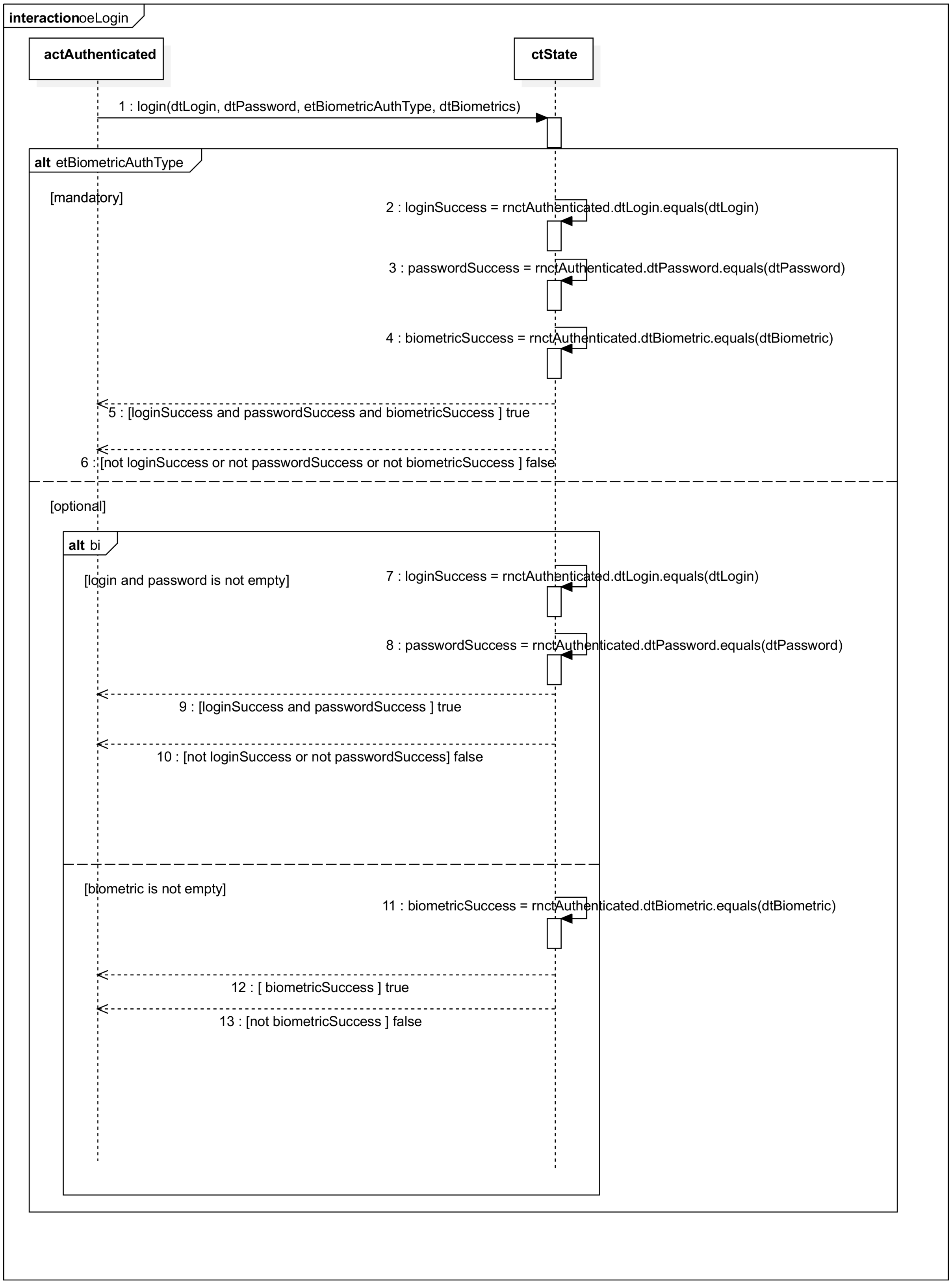
interaction oeLogin
actAuthenticated
ctState
1 : login(dtLogin, dtPassword, etBiometricAuthType, dtBiometrics)
alt etBiometricAuthType
[mandatory]
2 : loginSuccess = rnctAuthenticated.dtLogin.equals(dtLogin)
3 : passwordSuccess = rnctAuthenticated.dtPassword.equals(dtPassword)
4 : biometricSuccess = rnctAuthenticated.dtBiometric.equals(dtBiometric)
5 : [loginSuccess and passwordSuccess and biometricSuccess] true
6 : [not loginSuccess or not passwordSuccess or not biometricSuccess] false
[optional]
alt bi
[login and password is not empty]
7 : loginSuccess = rnctAuthenticated.dtLogin.equals(dtLogin)
8 : passwordSuccess = rnctAuthenticated.dtPassword.equals(dtPassword)
9 : [loginSuccess and passwordSuccess] true
10 : [not loginSuccess or not passwordSuccess] false
[biometric is not empty]
11 : biometricSuccess = rnctAuthenticated.dtBiometric.equals(dtBiometric)
12 : [biometricSuccess] true
13 : [not biometricSuccess] false

Fig. 5.1 Deployment Diagram

5.1.2 oeMakeFullReport

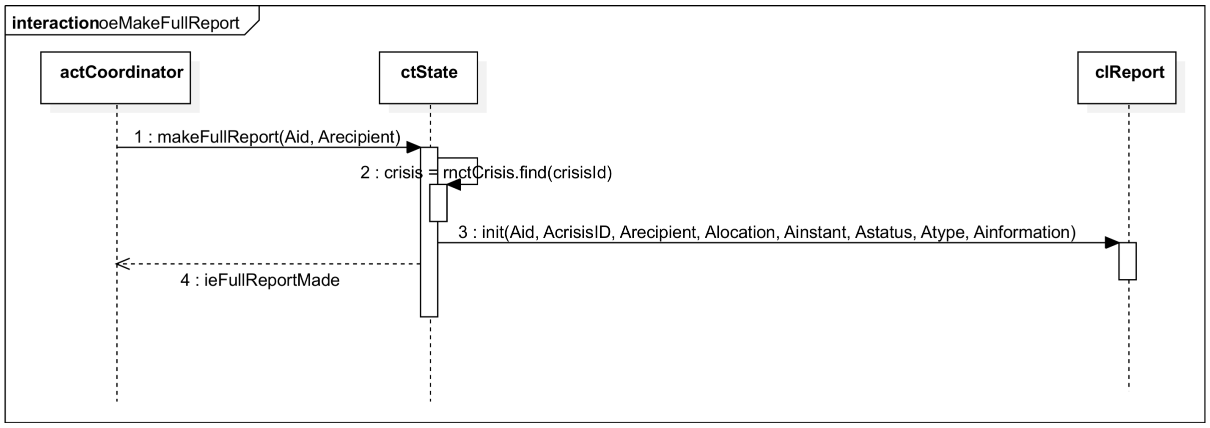
interaction oeMakeFullReport
actCoordinator
ctState
clReport
1 : makeFullReport(Aid, Arecipient)
2 : crisis = rnctCrisis.find(crisisId)
3 : init(Aid, AcrisisID, Arecipient, Alocation, Ainstant, Astatus, Atype, Ainformation)
4 : ieFullReportMade

Fig. 5.2 Deployment Diagram

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.

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 Human factor

The system has only one Administrator and authorithation for the administrator mandatory requires the administrator's voice (customer's desire). If the administrator is sick or somehow lost his voice the login to the system will not be possible. If a coordinator lost his voice the problem may be solved by contacting the administrator.

6.2 Responsibility of the coordinator

Responsibility for checking the reliability and consistency of information about the current situation about the accident belongs to the coordinator of the crisis. Before sending the report to a recipient coordinator must fully check the information.

6.3 Location of the accident

One of the main limitation is situation when two or several assidents hapen in the same time. Some information may be confusing or contradictory, when alerts appear in the system.

Chapter 7
Conclusion

This document has discussed the design of the ***iCrash*** System. It provides guidance and template material which is intended to assist the relevant management or technical staff, whether client or supplier, in producing a project-specific Design Document document. It is also useful background reading for anyone involved in developing or monitoring the ***iCrash*** System.

The main emphasis in this document is made on features that the **team 1** are responsible:

- biometric authentication;
- information diffusion.

Future versions of the document will fill in the missing information.

Glossary

Concept Model the Model that describes the different types required to specify the software system. . . . 9

Deployment View The physical view depicts the system from a system engineer's point-of-view. It is concerned with the topology of software components on the physical layer, as well as the physical connections between these components. For example, how many nodes are used and what is deployed on what node. A Deployment View is modelled as a UML Deployment Diagram. 17

Environment Model the Model that describes the different actors supposed to interact with the software system. 9

Implementation View This view describes the software system components. It focuses on software modules and subsystems. It describes the hierarchies or layers for components. This view is modelled as a UML Component Diagram. 17, 18

System Operation a functionality of the system that can be triggered by a message sent by an actor belonging to the environment. 8, 21

UI Processing View A Processing view is aimed at explaining the required object interactions that allow a system operation to be called. A UI Processing View is modelled as a UML Sequence Diagram. . . 17, 18

References