Rafael Valiev
Kasatkin Timur
Gregory Yermolaev

Innopolis University

myCrashVariant Design Document - v 0.0.5 -

Sunday 26th March, 2017 - 15:21

Springer

Contents

List of Figures

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.

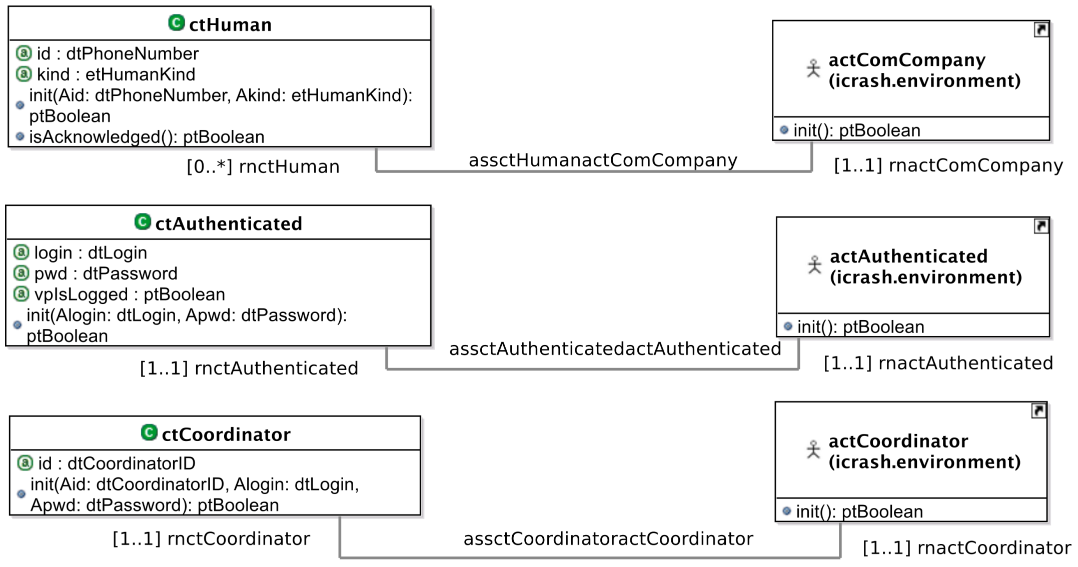
Chapter 6 lists all known problems of myCrashVariant and describes them in detail.

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

Fig. 2.1 Concept Model - Primary types class types global view

2.1.1 Primary types - Class types descriptions

2.1.1.1 ctAuthenticated

Used to model system's representation about actors that need to authenticate to access some specific functionalities.

2.1.1.2 ctAdministrator

Used to caracterize internally the entity that is responsible of administrating the myCrashVariant system.

2.1.1.3 ctCoordinator

Used to model system's representation about the actors that have the responsibility to handle alerts and crisis.

2.1.1.4 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 myCrashVariant system.

2.1.1.5 ctHuman

Used to model system's representation about the indirect actors that has alerted of potential crisis.

2.1.1.6 ctAlert

Used to model crisis alerts sent by any human having communication capability using communication companies belonging to the system's environment.

2.1.1.7 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

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 myCrashVariant user

dtLongitude

Used to define a longitude value of a geograpical positions on earth.

dtPassword

A password string used to authentify an myCrashVariant user

dtPhoneNumber

A string used to store the phone number from the human declaring the crisis or the alert.

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.

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.

2.2 Environment Model

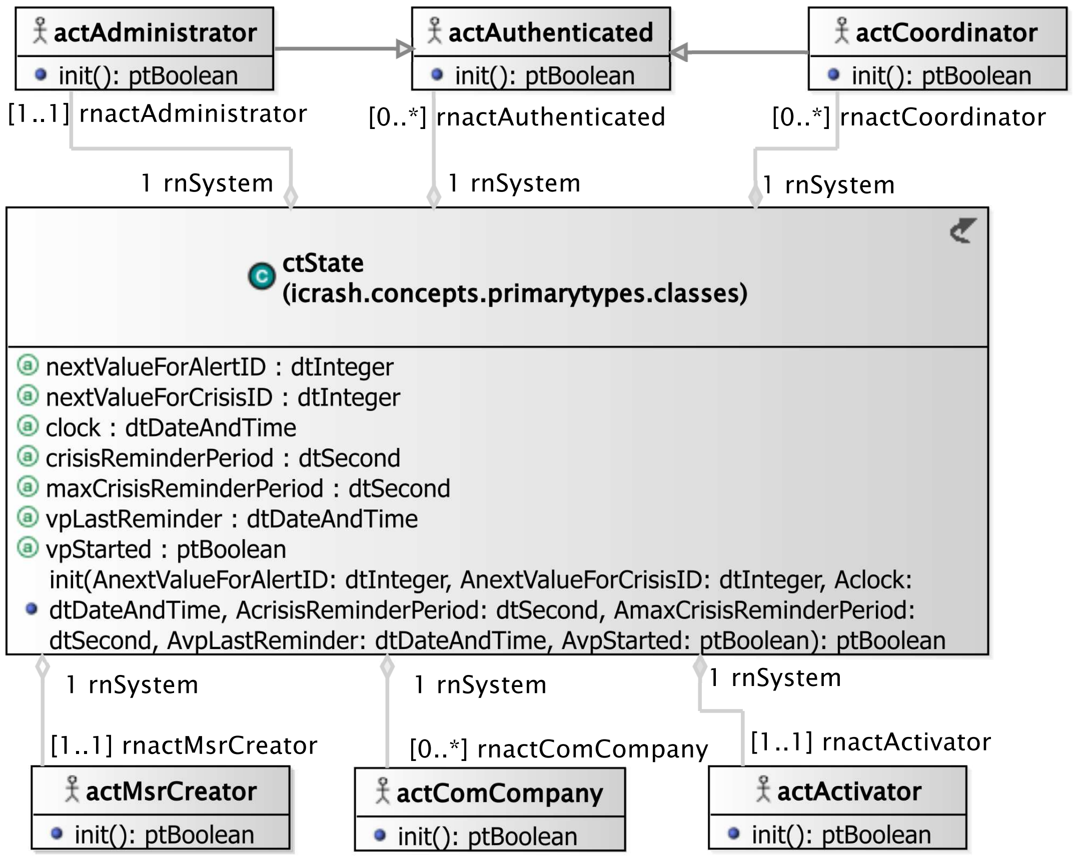

actAdministrator
init(): ptBoolean
actAuthenticated
init(): ptBoolean
actCoordinator
init(): ptBoolean
[1..1] rnactAdministrator
[0..*] rnactAuthenticated
[0..*] rnactCoordinator
1 rnSystem
1 rnSystem
1 rnSystem
ctState
(icrash.concepts.primarytypes.classes)
nextValueForAlertID : dtInteger
nextValueForCrisisID : 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. 2.2 Environment Model - Global View

2.2.1 actMsrCreator Actor

Represents the creator stakeholder in charge of state and environment initialization.

2.2.2 actActivator Actor

Represents a logical actor for time automatic message sending based on system's or environment status.

2.2.3 actAuthenticated Actor

Abstract actor providing reusable input and output interfaces for actors that need to authenticate themselves.

2.2.4 actAdministrator Actor

Represents an actor responsible of administration tasks for the myCrashVariant system.

2.2.5 actCoordinator Actor

Represents actor responsible of handling one or several crisis for the myCrashVariant system.

2.2.6 actComCompany Actor

Represents the communication company stakeholder ensuring the input/ouput of textual messages with humans having communication devices.

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

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.

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 Component xx.yy.zz.c1

TODO

4.2.2 Component xx.yy.zz.c2

TODO

4.2.3 Component xx.yy.zz.c3

TODO

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

Glossary

Abstract Actor an actor that does not exist in real life. 7

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. 7, 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. 7, 17

Model-View-Controller the pattern followed to design the Graphical User Interfaces of the software system. 7

System Operation a functionality of the system that can be triggered by a message sent by an actor belonging to the environment. 7, 19

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. . 7, 17, 18

References