UDC 004.05

R. LEDYAYEV[1], B. RIES[2], A. GORBENKO[1]

[1] ***Department of Computer Systems and Networks, National Aerospace University, Kharkiv, Ukraine***
[2] ***Laboratory for Advanced Software Systems, University of Luxembourg, Luxembourg***

REACT: AN ARCHITECTURAL FRAMEWORK FOR THE DEVELOPMENT OF A SOFTWARE PRODUCT LINE FOR DEPENDABLE CRISIS MANAGEMENT SYSTEMS

The paper reports our practical research efforts and experience in specifying REACT framework in the context of software product line engineering, striving to incorporate dependability into the system requirements. We present an requirements specification derived from the REACT and discuss how REACT-based applications can be implemented in the context of service-oriented architecture. Dependability policies configuration capabilities were incorporated to enable developers to control the behaviour of the systems facing certain threats and failures. We discuss a Car Crash Management System as a case study of REACT-based project.

Key words: *software product line engineering; dependability policies; crisis management systems; architectural framework; service-oriented architecture.*

Introduction

The need for crisis management systems has grown significantly over time. A crisis can range from major to catastrophic, affecting many segments of society. Crisis management involves identifying, assessing, and handling crisis situations. A crisis management system (CMS) facilitates this process by orchestrating the communication between all parties involved in handling the crisis. The CMS allocates and manages resources, and provides access to relevant crisis-related information to authorized users of the CMS.

A Software Product Line (SPL), as defined by Clemens and Northrop [3], is "a set of software-intensive systems sharing a common, managed set of features that satisfy specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way".

The variety of CMS and their commonalties have motivated us to consider applying SPL engineering approach to the development of dependable crisis management systems. Due to their nature, crisis management systems must be dependable, i.e. they must be able to deliver services that can justifiably be trusted [1]. Few techniques have been defined that take into consideration dependability constraints and SPL engineering approach. In this paper, we will experiment with some of the techniques developed at the University of Luxembourg [4, 8] in a particular development environment and introduce an extension to the platform to deal with dependability attributes at the implementation-level. The development environment used for this experimentation is the REACT framework, used by instructors and students of the University of Luxembourg in teaching courses on software engineering and software product line engineering. It is also targeted at researchers as a case study to perform experimental validations.

The rest of the paper is organized as follows. Section 1 highlights the general approach followed in this paper that combines the concepts of: SPL, dependability and architectural framework. Section 2 describes the process of combining various methodologies for the requirements elicitation of the REACT platform. Section 3 provides some theoretical and practical details on creating REACT applications and enhancing them with dependability policies configuration capabilities. Section 4 discusses the Car Crash Management System application case study. Finally, the analysis of the results of our research and development efforts, with some suggestions for future research, is presented in the Conclusion.

1. Dependable Crisis Management Systems Development with REACT

In this work, we have followed a software product-line approach to develop dependable crisis management systems. The activities that we performed to develop such systems are based on the abstract process for SPL engineering defined by Pohl et al. [9]. This process defines a number of activities divided into two categories: d*omain activities,* which are related to the *software platform,* and *application activities* – related to the concrete systems under development.

We have focused our work on the following three activities: Domain Requirements Engineering, Application Requirements Engineering, Application Design.

© R. Ledyayev, B. Ries, A. Gorbenko

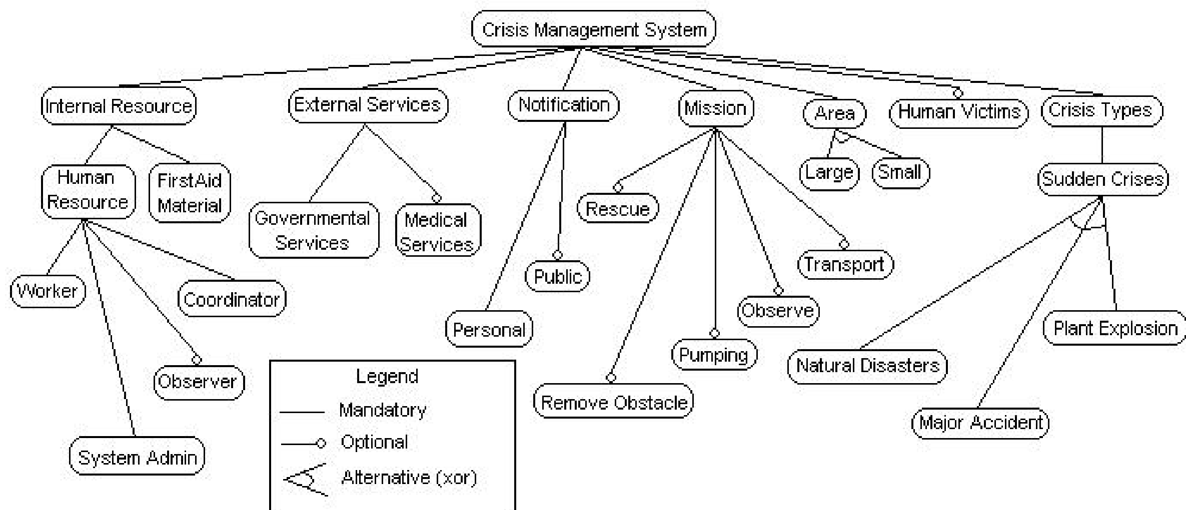

Fig. 1. Some Features of the REACT Platform

This abstract process must be instantiated with concrete techniques in order to be usable.

We have selected specification notations from the previous work held at the University of Luxembourg. In the Domain and Application Requirements Engineering activities, we use the artifacts defined in the FIDJI methodology [8] for the analysis phase in a software product line approach, and complement it with DRET [4] for the specification of dependability attributes.

In our approach, the software platform used to develop dependable crisis management systems is implemented as an *architectural framework*. An architectural framework [8] is "a layered set of reusable models characterizing core assets devoted to the specification and realization of a specific SPL". The Application Design activity is performed by means of derivation from the platform, following a set of guidelines which allow developing systems that reuse some of the platform components and have architecture compliant with that of the framework.

2. Dependability-enhanced Requirements for REACT Framework

We built our work on the previous efforts of others to draw requirements for general crisis management product line as part of the REACT project. The work on REACT requirements began with selecting the core set of features that could be represented in the systems which we set out to describe. From a large set of proposed features we have selected a subset, large enough to be representative and small enough to be manageable. This subset is depicted in Fig. 1. The next step consisted in the actual Domain requirements engineering with DRET. DRET is a requirements elicitation template, suitable for the elicitation of dependable SPLs. This template is composed of two parts: a DOMain Elicitation Template (DOMET), which represents a data dictionary and is depicted using a tabular notation; and a Use Case Elicitation Template (UCET), which represents SPL members' behaviour and is depicted using a use-case-scenario-based template. In the context of our car crash management system case study, concrete examples of DOMET and UCET are given in Section 4. Complete specification of these two templates is defined by Gallina et al. [4].

3. REACT Applications with Dependability Policies Configuration Capabilities

One of the major goals of creating the REACT framework was to enable developers to derive new systems belonging to the same family (line) from this architectural framework without unnecessary time and effort overheads needed for creating the same systems from scratch. We have accomplished this by including most general business objects and functionality into our framework and providing the means for new product derivation along with the framework distribution.

The process of deriving new systems from REACT is fairly straightforward. Two kinds of such systems could be created using the framework: server and client. Each client can simultaneously use one, two or any other number of servers located anywhere in the world. A typical physical architecture of a REACT application is shown in Fig. 2.

Another goal we set out to accomplish was to make these newly derived systems more dependable and resilient. This was accomplished by introducing dependability configuration capabilities for each of these systems.

What is of particular interest for us here is the existence of two configuration files, one on the server and the other on the client side, which have direct impact on systems behaviour and dependability.

The first of these files is an XML file, called configurations.xml, stored on the server. In the current implementation of REACT server, the purpose of this file is twofold:

1. To specify which services to load at startup.
2. To provide dependability metadata about each of the available services.

At the moment, the metadata is collected and registered in this file manually. This metadata consists of the following: Name of the service, Minimum response time, Maximum response time, Average response time, Transactions per second, Bytes per second and Errors.

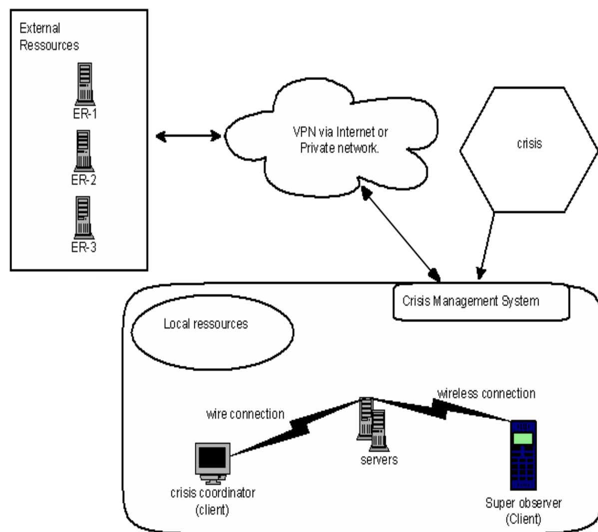

Fig. 2. Physical Architecture of a REACT Application

The second file is an XML file, called ClientConfigurations.xml, stored on the client. Currently, this file serves two purposes:

1. Choosing which dependability policy to use.
2. Specifying maximum allowed response time for the service called.

Four patterns of dependability-oriented composition [6], adapted from our previous work, have been used to develop policies for the Client application. These patterns were not 100% implemented but simply used as a basis for our dependability policies for the purpose of illustration. In a nutshell, each of these patterns could be used to prescribe different system behavior, which is the gist of what we have done with them. For more detailed information on these patterns we refer the interested reader to the work by Gorbenko et al. [5].

4. Car Crash Management Application derived from the REACT Framework

Let us illustrate all of the theoretical information above with a brief Car Crash Management System case study. To create this system we have selected features corresponding to car crash management activities and eliminated the features we did not need.

Next, we have described our system using the combination of FIDJI and DRET and registered all of the necessary features in the DOMET table. Part of this table is depicted in Fig. 3.

The behavior of the system is described using UCET use cases. The Car Crash Management system summary level use case is described as follows:

ID: UC01

Collaborative Use Case name: Deal with the crisis.

Selection category: Mand.

Description: Describes the main sequence of events that take place when car crash happens.

Synchronous Primary Actors: System Administrator, Worker, Coordinator, External Services.

Synchronization: Among other Single and Collaborative UCETs of the system.

Resources: DB-data (competitive sharing, transactional resources), Notification data (cooperative resource).

Dependency: Includes Single UC02, Single UC03, Single UC04, Single UC05, Single UC07, Collaborative UC08, Single UC11.

Preconditions: The crisis has occurred. Worker is at the scene of the crisis. Administrator is not signed in. Missions are not assigned and not executed.

Postconditions: The crisis is dealt with and its consequences are eliminated (main valid scenario postcondition). The crisis is not dealt with and not all of the consequences are eliminated (mis-scenario postcondition).

Concept Name	Var Type	Description	Dependencies	Misuse & class(es)	Misconception consequence & class(es)	Priority Level
System Admin	Mand	Crisis management system administrator. Manages users and devices.	Specialization of Human Resource	Wrong information entering M/NM	Wrong content. CF	High
…	…	…	…	…	…	…

Fig. 3. Car Crash Management System. Fragment of DOMET Specification

Main scenario (abbreviated):

1. Worker calls the Car Crash Management centre and Administrator answers the call.
2. Worker reports the crisis to Administrator.
3. Administrator signs into the system (Single UC02).
4. Administrator registers the crisis with the system (Single UC03).
5. Administrator selects a Coordinator and sends him a Personal Notification (Single UC04) …

Alternatives to the main scenario:

3a. If the System Administrator credentials are invalid, he is invited to retry. If they are valid the main sequence continues starting from step 4.

Variation points description: V1: Type=Alt, Format={worker, witness, device}, Concerns=Behavior.

Non-Functional: security: it must always be true that logging into the system is done via secure connection; efficiency: it must always be true that there are no delays in communication between the actors of the system; reliability, scalability.

Duration: Dx hours (where Dx is the time between Administrator`s logging in and Coordinator`s informing the system that the crisis was successfully managed).

Mis-scenarios: Administrator misunderstands column meaning and enters wrong information about the crisis.

Recovery scenarios: The erroneous condition concerning the wrong information is detected and the Administrator is requested to enter the information again.

Note that fields in italics have been introduced in DRET to help in recognizing commonalities and variabilities; and that underlined fields specify dependability-related properties. Following this requirements elicitation stage, we have derived a Car Crash Management server application from the REACT framework and developed a client application following the requirements we elicited in the previous stage.

The Apache Tuscany framework [2, 7] for developing service-oriented applications is the heart of all of the service binding happening behind the scenes. Tuscany implements a number of standard communication protocols (e.g. SCA, SOAP, RMI). REACT service components may communicate with each other or with external services using any of these protocols.

In this paper, we focus on implementation parts related to dependability. What is of special interest for us here and, therefore, is worth mentioning in this article is what happens every time there is a need to call a particular service. The method getServiceAccordingToPolicy (policy, service1, service2)**,** belonging to one of the classes on the client side, is the heart of the logic performing all of the necessary voting on which service to choose to carry out the requested functionality. The choice is based on the *policy* used and the *metadata* available for each service. Each time we want to call a specific service:

1) the name of the policy to be used and/or any additional information is read from the ClientConfigurations.properties file located on the Client;

2) based on the policy, the appropriate metadata about the given service is read from the configurations.xml files located on the Servers;

3) based on this metadata and/or any other conditions specified in the policy the voting takes place on which service fits the policy criteria best;

4) the appropriate service is returned to be used by the Client.

Conclusion

The contribution of this paper is two-fold. In the first place, we have reported on experimentation with existing techniques in a consecutive case study: the crisis management systems product line. In the second place, we have extended the REACT platform with dependability capabilities, so that REACT applications (derived from the platform) increase their dependability by using some predefined dependability policies and metadata. Experimentation has been performed that derives three different CMS applications. This paper presents one of these three derived applications with the selected methodological techniques. This experimentation allows us to formulate the following feedback.

At the requirements elicitation stage of our work we have experienced that it was not easy to determine which features should be included in the framework and which should not, to determine which business objects are general enough to belong to any derived system and to eliminate all of the unnecessary dependencies.

We suggest that more attention should be given to carefully examining/re-engineering the: Features belonging to the framework; Existing business objects; Dependencies between various modules; Current UCETs and DOMETs. We propose several solutions for ensuring dependability of our SOA applications:

1. Using specific metadata to describe services
2. Using configuration files (.xml, .properties)

Configuration files are used to determine application behaviour (policies) and to store various dependability metadata (description).

In addition, some questions still remain concerning the following:

– which dependability metadata should be stored on the server and which on the client?

– who should collect this metadata?

– from where this metadata should be collected?

We believe that REACT framework itself could be enhanced with dependability collecting capabilities. Development of such dependable, scalable and high-performance generic architectural framework as REACT is in a great demand.

References

1. Basic Concepts and Taxonomy of Dependable and Secure Computing [Text] / A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr // IEEE Trans. on Dependable and Secure Computing. – 2004. – Vol. 1, No. 1. – P. 11–33.

2. Apache Tuscany [Electronic recourse]. – Access mode: http://tuscany.apache.org. – 15.03.2012 y.

3. Clements, P. Software Product Lines: Practices and Patterns [Text] / P. Clements, L. Northrop. – New York: Addison-Wesley Profrssional, 2001. – 608 p.

4. Gallina, B. A Template for Requirement Elicitation of Dependable Product Lines / B. Gallina, N. Guelfi. – Conf. on Requirements Engineering: Foundation for Software Quality – REFSQ'07: conf. proc. – Trondheim (Norway), 2007. – P. 63–77.

5. Gorbenko, A. Using Inherent Service Redundancy and Diversity to Ensure Web Services Dependability [Text] / A. Gorbenko, V. Kharchenko, A. Romanovsky // Methods, Models and Tools for Fault Tolerance, LNCS 5454 / Eds. M. Butler, C. Jones, A. Romanovsky, E. Troubitsyna. – Berlin, Heidelberg (Germany): Springer-Verlag, 2009. – P. 324–341.

6. Ledyayev, R. Developing Techniques for Increasing Dependability of Service-Oriented Systems. BS Thesis / R. Ledyayev. – Kharkiv: National Aerospace University, 2008. – 108 p.

7. Service Component Architecture. Assembly Model / Eds. M. Beisiegel, A. Karmarkar, S. Patil, M. Rowley. – OASIS, 2011. – 138 p.

8. Perrouin, G. Architecting Software Systems Using Model Transformations and Architectural Frameworks. PhD thesis / G. Perrouin. – Univ. of Luxembourg, 2007. – 142 p.

9. Pohl, C. Software Product Line Engineering / C. Pohl, G. Böckle, F. J. van der Linden. – Berlin: Springer-Verlag Heidelberg, 2005. – 334 p.

Поступила в редакцию 15.03.2012

Рецензент: д-р техн. наук, проф., зав. каф. компьютерных систем и сетей В.С. Харченко, Национальный аэрокосмический университет им. Н.Е. Жуковского «ХАИ», Харьков.

REACT: АРХІТЕКТУРНИЙ КАРКАС РОЗРОБКИ СІМЕЙСТВА ПРОГРАМНИХ ПРОДУКТІВ ДЛЯ ГАРАНТОЗДАТНИХ СИСТЕМ УПРАВЛІННЯ КРИЗАМИ

Р. Ледяев, Б. Рієс, А. Горбенко

В статті представлені результати дослідження й розробки специфікації для архітектурного каркасу REACT в контексті інжинірингу сімейства програмних продуктів з урахуванням вимог до гарантоздатності. Представлено елементи специфікації вимог до REACT та розглянуто особливості організації в контексті використання сервіс-орієнтованої архітектури. В проекті REACT було реалізовано можливість конфігурування політик гарантоздатності, що дозволяє розробникам контролювати поведінку систем управління кризами за умов відмов та зовнішніх загроз. В якості прикладу реалізації проекту REACT розглянута система ліквідації наслідків автомобільних аварій.

Ключові слова: інжиніринг сімейства програмних продуктів, політики гарантоздатності, системи управління кризами, архітектурний каркас, сервіс-орієнтована архітектура.

REACT: АРХИТЕКТУРНЫЙ КАРКАС РАЗРАБОТКИ СЕМЕЙСТВА ПРОГРАММНЫХ ПРОДУКТОВ ДЛЯ ГАРАНТОСПОСОБНЫХ СИСТЕМ КРИЗИС-МЕНЕДЖМЕНТА

Р. Ледяев, Б. Риес, А. Горбенко

В статье представлены результаты исследования и разработки спецификации для архитектурного каркаса REACT в контексте инжиниринга семейства программных продуктов с учетом требований к гарантоспособности. Представлены элементы спецификации требований к REACT и рассмотрені особенности реализации в контексте использования сервис-ориентированной архитектуры. В проекте REACT была реализована возможность конфигурирования политик гарантоспособности, что позволяет разработчикам контролировать поведение систем кризис-менеджмента в условиях отказов и внешних угроз. В качестве примера реализации проекта REACT рассмотрена система ликвидации последствий автомобильных аварий.

Ключевые слова: инжиниринг семейства программных продуктов, политики гарантоспособности, системы кризис-менеджмента, архитектурный каркас, сервис-ориентированная архитектура.

Ledyayev Roman – MSc, Analyst Programmer, Application Owner at Accenture Inc., Riga, Latvia.

Ries Benoît – PhD, Scientific Support Staff Member, Laboratory for Advanced Software Systems, University of Luxembourg, Luxembourg.

Gorbenko Anatoliy – PhD, Docent, Associate Professor, Department of Computer Systems and Networks, National Aerospace University, Kharkiv, Ukraine.