The Journal of Systems and Software 122 (2016) 239–273

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Supporting pattern-based dependability engineering via model-driven development: Approach, tool-support and empirical validation

Brahim Hamid [a,*], Jon Perez [b]

[a] IRIT, University of Toulouse, 118 Route de Narbonne, 31062 Toulouse Cedex 9, France
[b] IKERLAN-IK4 Research Centre, Mondragon, Spain

ARTICLE INFO

Article history:
Received 28 December 2015
Revised 15 July 2016
Accepted 18 September 2016
Available online 21 September 2016

Keywords:
Dependability
Safety
System engineering
Patterns
Meta-modeling
Model driven engineering

ABSTRACT

Safety-critical systems require a high level of safety and integrity. Therefore, generating such systems involves specific software building processes. Many domains are not traditionally involved in these types of software problems and must adapt their current processes accordingly. Typically, such requirements are developed ad hoc for each system, preventing further reuse beyond the domain-specific boundaries. This paper proposes a solution for software system development based on the reuse of dedicated subsystems, i.e., so-called *dependability patterns* that have been pre-engineered to adapt to a specific domain. We use Model-Driven Engineering (MDE) to describe dependability patterns and a methodology for developing dependable software systems using these patterns. Moreover, we describe an operational architecture for development tools to support the approach. An empirical evaluation of the proposed approach is presented through its practical application to a case study in the railway domain, which has strong dependability requirements, to support a pattern-based development approach. This case study is followed by a survey to better understand the perceptions of practitioners regarding our approach.

1. Introduction

Safety-critical systems require a high level of safety and integrity. Therefore, the generation of such systems involves specific software building processes. These processes are often error-prone because they are not fully auto- mated, even if some level of automatic code generation or model-driven engineering support is applied. Furthermore, many critical systems also have assurance requirements, ranging from very strong levels involving certification (e.g., EN-50,129 (CENELEC, 1999) for railway systems and DO-178B (RTCA, 1992) for airborne systems) to reduced levels based on industry practices. These systems can be found in many application sectors such as automotive, aerospace, and home control, and come with many common characteristics, including real-time and temperature constraints, computational processing, power constraints and/or limited energy and common extra-functional properties such as dependability, security and efficiency (Ravi et al., 2004; Kopetz, 2011).

The integration of various concerns, such as dependability, requires the availability of both application development and expertise. Many domains not traditionally involved in this type of software development and must adapt their current processes accordingly. Typically, such requirements are developed ad hoc for each system, preventing further reuse beyond the domain-specific boundaries. This is especially true for railway systems, as they exist in many use cases. Many of these systems belong to critical infrastructures, where other economic and social aspects are based on. Hence capturing and providing this expertise via *dependability patterns* (Daniels and Vouks, 1997; Powel, 2003; Tichy et al., 2004; Radermacher et al., 2013) has become recently an area of research. Dependability patterns enable the development of dependable applications and liberate the developer from having to address technical details. We believe that the specification and packaging of dependability patterns can provide an efficient means of addressing these problems, improving industrial efficiency and fostering technology *reuse* across domains (the reuse of models at different levels), thus reducing the time and effort required to design a complex system (McClure, 1997; Agresti, 2011). Model-driven engineering (MDE) (Selic, 2003; Atkinson and Ku¨hne, 2003) also provides a very useful contribution to the design of safety-critical systems (Ziani et al., 2012; Panesar-Walawege et al., 2013) because it reduces the time/cost required for understanding and analyzing system artifact descriptions due to the abstraction mechanisms. Moreover, it reduces the cost of the development process thanks to the generation mechanisms. Hence, dependability pattern integration must be considered during the MDE process.

* Corresponding author. Fax: +33 5 6150 4173.
E-mail addresses: hamid@irit.fr (B. Hamid), JPerez@ikerlan.es (J. Perez).

http://dx.doi.org/10.1016/j.jss.2016.09.027
0164-1212/

In system and software engineering, design patterns (Gamma et al., 1995; Henninger et al., 2007) are considered effective tools for the reuse of specific information. They are widely used today to provide architects and designers with reusable design knowledge. They are triples that describe *solutions* for commonly occurring problems in specific contexts. Indeed, pattern-based development has recently gained more attention in software engineering by addressing new challenges that had not been targeted in the past (Henninger et al., 2007). In fact, they are applied in modern software architecture for distributed systems, including middleware and real-time embedded systems (Schmidt and Buschmann, 2003). There are patterns for generic architecture problems (Buschmann et al., 1996, 2007), security (Schumacher, 2003; Fernandez, 2013), safety (Alexander et al., 2007; Preschern et al., 2013) and other non-functional requirements (Powel, 2003). The related approaches promote the use of patterns via reusable design artifacts. However, a gap between the development of systems using patterns and the information in the pattern representations remains (Zdun and Avgeriou, 2008). This becomes even more observable when addressing specific concerns, such as dependability.

In this paper, we present a model-based approach for dependability system and software engineering that uses *patterns* to represent dependability solutions and knowledge, which fosters reuse. In such a vision, the dependability patterns derived from (resp. associated with) domain-specific models are designed to assist the application developer integrate application models with dependability building-block solutions. Dependability patterns are defined from a platform-independent perspective (i.e., they are independent of the implementation) and are expressed in a consistent manner with domain-specific dependability models. Consequently, they will be much easier to understand and validate by application designers in a specific area. This work is conducted within the context of a model-based security and dependability research project, and our collaboration with safety-critical system suppliers suggested a need for this work. The dependability solutions used by safety-critical system developers are based on the application domain and occasionally on the software development environment, including the design and coding stages. There is a need to link these concepts to dependability, which will ease the certification process. The lack of appropriate links between application domain concepts and dependability concepts poses three main challenges. First, the dependability engineer must re-engineer its existing solutions. Second, the application designer may not understand domain-specific dependability solutions. Finally, it is very difficult for the system developer to guarantee the availability of dependability solutions to cover all the dependability requirements of the targeted application using the application domain concepts.

To provide a concrete example, we introduce a railway case study from the TERESA project[1] called Safe4Rail, which is a simplified version of a real ETCS (European Train Control System) (UNISIG, 2009; Stanley, 2011). The main functionality of this demonstrator is to ensure that the traveled speed and distance do not exceed the authorized maximum values provided by the railway infrastructure. To implement this functionality, the system is composed of multiple subsystems, including the European Vital Computer (EVC), which executes the safety application, and a set of odometry sensors and actuators. The odometry sensors provide the speed and acceleration of the train. With these values, the system must be able to calculate accurate speed and position values (odometry). There is a family of products in the railway sector, including regional trains, tramways, and high-speed trains. These units share common parts, although they differ in distinct ways. For example, consider the calculation of the actual speed and position by Safe4Rail. The implementation varies according to each type of train product, which depends on the safety level to be met and the type and number of sensors and actuators that are involved. These considerations greatly influence how each product is implemented because several issues should be considered: the number of channel redundancies, the diversity of the channels, the monitoring of the channels, and the interaction with assorted data (type, weight, etc.).

A proprietary embedded system has been designed to meet stricter safety regulations. In our case, the SIL4 level is pursued. To achieve this level, several design techniques from related standards, such as IEC-61,508 (IEC, 2010b) and EN-50,126 (CENELEC, 1999), are used, including redundancies, votations, diagnostics, and secure and safe communications. Hence, capturing and providing this expertise by means of a repository of dependability patterns and models can enhance the development of embedded systems. We seek mechanisms that allow a safer, easier and faster safety-critical development process. To illustrate this statement, two different railway industry scenarios are described. The first takes place within a railway manufacturing group, whereas second occurs within an SME.

Scenario 1. *Railway manufacturing group.* The first scenario takes place within a group of railway manufacturers. The group is divided into subsidiary companies that specialize in the development of train and railway infrastructure systems (e.g., traction, central control, infotainment, railway signaling, interlocking, Communication Based Train Control (CBTC) (IEEE, 2004), etc.). The group intends to develop its own safety-related subsystems instead of buying them from providers and competitors. The problem is that the engineering cost associated with these safety-related systems is relatively high; moreover, the number of qualified engineers is limited. Additionally, safety concepts and design patterns are not present in commonly used modeling languages (e.g., UML, SysML, etc.), which leads to design ambiguity. The expected benefits for this scenario are the following: (1) reduce product development cost and time by reusing design patterns for other projects and companies, and (2) reduce the probability of systematic faults by reducing ambiguity.

Scenario 2. *SME company.* In the second scenario, an SME that develops safety-related embedded systems is presented. This company has proven experience in the development of IEC-61,508-based safety-related embedded systems, although the company has little experience in the railway domain. If this company wants to enter the railway market, it must overcome a great barrier and analyze relevant railway standards, adapt the solutions it has been using for years in other sectors, and be ready for success. Some of the problems are the same as those that arise in the first scenario: the development cost for safety-related systems is high, and the number of qualified engineers is limited. Additionally, in this context, systems are developed with different standards derived from IEC-61,508 (the differences are well known). The expected benefits for this scenario are the following: (1) reduce product development cost and time, with the cross-domain reuse of design patterns between projects; (2) reduce the probability of systematic fault by reducing ambiguity; and (3) provide a cross-domain arsenal of design patterns that can be used in new domains.

To address issues related to scenario 1, an infrastructure that allows the reuse of the most common techniques used to achieve the dependability requirements in most subsystems would allow these subsidiaries to reduce efforts, and costs while also reducing the number of required and scarce safety experts. With regard to the issues related to scenario 2, we offer an infrastructure that allows the cross-domain reutilization of the techniques used to achieve the target safety level.

[1] http://www.teresa-project.org/.