Journal of Systems Architecture 000 (2017) 1–11

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

Architecting resilient computing systems: A component-based approach for adaptive fault tolerance

Miruna Stoicescu[1], Jean-Charles Fabre, Matthieu Roy*

LAAS-CNRS, Université de Toulouse, CNRS, INP, Toulouse, France

ARTICLE INFO

Article history:
Received 20 April 2016
Revised 29 November 2016
Accepted 21 December 2016
Available online xxx

Keywords:
Dependability
Adaptivity
Fault tolerance
Component-based middleware
Runtime reconfiguration

ABSTRACT

Evolution of systems during their operational life is mandatory and both updates and upgrades should not impair their dependability properties. Dependable systems must evolve to accommodate changes, such as new threats and undesirable events, application updates or variations in available resources. A system that remains dependable when facing changes is called resilient. In this paper, we present an innovative approach taking advantage of component-based software engineering technologies for tackling the on-line adaptation of fault tolerance mechanisms. We propose a development process that relies on two key factors: designing fault tolerance mechanisms for adaptation and leveraging a reflective component-based middleware enabling fine-grained control and modification of the software architecture at runtime. We thoroughly describe the methodology, the development of adaptive fault tolerance mechanisms and evaluate the approach in terms of performance and agility.

1. Introduction

Dependable systems are becoming increasingly complex and their capacity to evolve in order to efficiently accommodate changes is a requirement of utmost importance. Changes have different origins, such as fluctuations in available resources, additional features requested by users, environmental perturbations... All changes that may occur during service life are rarely foreseeable when designing the system. Dependable systems must cope with changes while maintaining their ability to deliver trustworthy services and their required attributes, e.g., availability, reliability, integrity [1].

A lot of effort has been put into building applications that can adapt themselves to changing conditions. A rich body of research exists in the field of software engineering consisting of concepts, tools, methodologies and best practices for designing and developing adaptive software [2]. For instance, *agile software development* approaches [3] emphasize the importance of accommodating change during the lifecycle of an application at a reasonable cost, rather than striving to anticipate an exhaustive set of requirements. Component-based approaches [4] separate service interfaces from their actual implementation in order to increase flexibility, evolvability and reuse. Although dependable systems could benefit from these advancements in order to become more flexible and adaptive, very little has been done in this direction for now. In this paper, we describe an innovative approach for tackling on-line adaptation of dependability mechanisms that leverages component-based software engineering technologies. Component-based fault tolerance mechanisms can be easily updated through transition packages.

The paper is organized as follows. Section 2 presents the context and motivation of our work. Section 3.1 briefly describes the resilient system architecture we consider; next, we describe a set of Fault Tolerance Mechanisms (FTMs) and analyze transitions between them in Section 3.2. Section 4 presents the process of designing FTMs for subsequent adaptation. In Section 4.4, we detail the practical implementation of FTMs on top of a reflective component-based support and in Section 5, we describe the implementation of on-line transitions between FTMs. Next, we evaluate our approach in Section 6. In Section 7, we discuss related work and in Section 8 we present the lessons learned and conclude.

2. Problem statement

Resilient computing refers to dependability in the presence of changes due to system evolution [5]. Our work focuses on *Fault Tolerance Mechanisms* (FTMs) that can be influenced by changes occurring in the system or in its environment. Ideally, fault-tolerant applications consist of two interconnected abstraction layers. The first one (the *base level*) contains the business logic that implements the functional requirements . The second one (the

* Corresponding author.
E-mail address: roy@laas.fr (M. Roy).
[1] EUMETSAT (European Organisation for the Exploitation of Meteorological Satellites)

http://dx.doi.org/10.1016/j.sysarc.2016.12.005
1383-7621/

meta level) contains fault tolerant mechanisms and is attached to the first one through clearly identified hooks. As the development of a fault-tolerant application implies different roles/stakeholders (application developer(s), safety expert, integrator), separation of concerns is a key concept. Separation of concerns has some limit since fault tolerance strategies often depend on the semantics of the business logic. However, hooks can be defined to externalize some "application defined assertions" used to parameterize fault tolerance mechanisms. In any case, the concept of separation of concerns is essential to implement adaptive fault tolerance without impacting deeply the business logic.

Among the well-established and documented FTMs, e.g., [6,7], the choice of an appropriate FTM for a given application depends on the values of several parameters. We identified three classes of parameters: fault tolerance requirements, application characteristics and available resources.

• **Fault tolerance requirements** (*FT*). This class of parameters contains the considered fault model. A fault model determines the category of FTM to be used (e.g., simple replication or diversification). Our fault model classification is well-known [1], dealing with crash faults, value faults and development faults. We focus on hardware faults (permanent and transient physical faults) but the approach can be extended to other FTMs.

• **Application characteristics** (*A*). The characteristics that have an impact on the choice of an FTM are application statefulness, state accessibility and determinism. State accessibility is essential for checkpointing-based fault tolerance strategies. Determinism refers here to behavioural determinism, i.e., the same inputs produce the same outputs in the absence of faults, mandatory for active replication.

• **Resources** (*R*). FTMs require resources in terms of CPU, battery life/energy, bandwidth, etc. A cost function can be associated to each FTM based on the values of such parameters. For a given set of resources, several mechanisms can be used with different trade-offs (e.g., more CPU, less bandwidth).

The first two parameters *FT* and *A* correspond to assumptions to be considered for the selection of an appropriate mechanism and to determine its validity. The resource dimension *R* states the amount of resources needed to accept a given solution according to system resources availability.

In practice, based on the values of (*FT,A,R*) set at design time, an FTM is attached to an application when the system is installed for the first time. As far as resilient computing [5] is concerned, the challenge lies in maintaining consistency between the FTM(s) and the non-functional requirements despite variations of the parameters at runtime, e.g.:

• new threats/faults and physical perturbations such as electromagnetic interferences trigger variations of *FT*;

• the introduction of new versions of applications or modules may trigger variations of *A*;

• resource loss or addition of hardware components imply variations of *R*.

If the FTM is inconsistent with the current (*FT,A,R*) values, it will most likely fail to tolerate the faults the system is confronted with. Therefore, a transition towards a new FTM is required before the current FTM becomes invalid, which is possible in Adaptive Fault Tolerance (AFT).

On-line adaptation of FTMs has attracted research efforts for some time now because dependable systems cannot be fully stopped for performing off-line modifications. However, most of the proposed solutions [8–10] tackle adaptation in a preprogrammed manner: all FTMs necessary during the service life of the system must be known and deployed from the beginning and adaptation consists in choosing the appropriate execution branch or tuning some parameters.

Nevertheless, predicting all events and threats that a system may encounter throughout its service life and making provisions for them is obviously impossible.

This approach is of interest for long-lived space systems (satellites and deep-space probes) and for automotive applications regarding *over-the-air software updates*, a very important trend in the automotive industry today.

In this context, we propose an alternative to preprogrammed adaptation that we denote *agile adaptation of FTMs* (the term "agile" is inspired from agile software development). Agile adaptation of FTMs enables their systematic evolution: new FTMs can be designed off-line at any point during service life and integrated on-line in a flexible manner, with limited impact on the existing software architecture. Our approach for the agile adaptation of FTMs leverages advancements in the field of software engineering such as Component-Based Software Engineering (CBSE) technologies [4], Service Component Architecture [11] and Aspect-Oriented Programming [12]. Using such concepts and technologies, we design FTMs as brick-based assemblies (similar to "Lego" constructions) that can be methodically modified at runtime through *fine-grained* modifications affecting a limited number of bricks. This approach maximizes reuse and flexibility, contrary to monolithic replacements of FTMs found in related work, e.g., [8–10]. It is worth noting that, whatever the approach is (pre-programmed or agile), when the FTM evolution goes outside the foreseen boundaries of the FTM loaded into the system, the system may fail. The benefits of the proposed agile approach is to provide means to react more quickly and to simplify updates of loaded FTMs.

Summary of contributions. In this paper, after a short description of the resilient system architecture, we describe our methodology for agile adaptation of FTMs and its results, focussing on the following three key contributions:

• A "design for adaptation" approach of a set of FTMs, based on a generic execution scheme.

• A component-based architecture of the considered FTMs and fine-grained on-line transitions between them.

• Illustration of on-line FTM adaptation through several transition scenarios.

3. Resilient system design

3.1. Architecture

The core objective of a resilient system is to guarantee the consistency of FTMs attached to applications according to major assumptions regarding the fault model and the application characteristics. First, this means identifying the link between applications and FTMs, and, more importantly, dynamically adapting FTMs according to operational conditions at runtime. As soon as FTMs are developed as assemblies of Lego bricks, it becomes easier to adjust their configuration by removing, adding, modifying individual bricks. This means that *bindings* between application and FTMs, but also inside FTMs can be managed dynamically. At runtime, any FTM implemented as a graph of Lego bricks is modified on-line when the FTM configuration is updated. Some Lego bricks can be uploaded when they are missing, after an off-line development when the FTM solution does not exist yet. The adaptation is carried out on-line by a specific service, called *Adaptation Engine* (see Fig. 1).

A second important feature of a resilient system, also shown on Fig. 1, is the runtime monitoring of the state of the system according to several view points. This core service of the architecture is called *Monitoring Engine*. The first conventional role of the monitoring is to measure resource usage *R*. The second important role of the monitoring is to analyze the non-functional behavior of the system. This task is more complex and requires specific

Please cite this article as: M. Stoicescu et al., Architecting resilient computing systems: A component-based approach for adaptive fault tolerance, Journal of Systems Architecture (2017), http://dx.doi.org/10.1016/j.sysarc.2016.12.005