Automatic Reliability Management in SOA-based critical systems

Senol Arikan
Integrated Communication Systems Group
University of Kaiserslautern, Germany
E-Mail: arikan@informatik.uni-kl.de
Supervised by Prof. Dr. Paul Mueller
http://www.icsy.de

Abstract. A well-known concept for the design and development of distributed software systems is service-orientation. In SOA, an interacting group of autonomous services realize a dynamic adaptive heterogeneous distributed system. Because of its flexibility, SOA allows an easy adaptation of new business requirements. This also makes the service-orientation idea a suitable concept for development of critical software systems. Reliability is a central parameter for developing critical software systems. SOA brings some additional requirements to the usual reliability models currently being used for standard software solutions. In order to fulfill all requirements and guarantee a certain degree of reliability, a generic reliability management model is needed for SOA based software systems. This article defines research challenges in this area and gives an approach to solve this problem.

Keywords: Distributed systems, SOA, critical systems, reliability, fault tolerance

1 Introduction

Service Oriented Architecture (SOA) provides a dynamic and adaptive solution for building distributed systems. Large software systems are built using loosely coupled, autonomous services that dynamically bind and discover each other through standard protocols. It allows for easy integration of old systems and rapid adaptation of new requirements. These characteristics make service orientation a suitable concept for development of critical IT landscapes especially for business critical systems.

A critical system is one where failures may have grave consequences, e.g. physical harm, environmental harm, high financial loses. Well-known examples of critical systems are medical devices, aircraft control, nuclear systems, and financial software systems like online banking [1]. It is essential to design and make those systems as dependable as possible before their complexity becomes too overwhelming and handling failures becomes too difficult [2]. One of the most important conditions for developing a dependable software system is reliability.
IEEE 610.12-1990 [5] defines reliability as "The ability of a system or component to perform its required functions under stated conditions for a specified period of time.” The main purpose behind focusing on reliability is to guarantee that the resources managed and used by the system are under control. It must be guaranteed that a user can complete its task with a certain probability when it is invoked. Software reliability management is defined in IEEE 982.1-1988 [6] as "The process of optimizing the reliability of software through a program that emphasizes software error prevention, fault detection and removal, and use of measurements to maximize reliability in light of project constraints such as resources, schedule, and performance". From these definitions the following classification of methods to develop reliable software systems can be derived: fault prevention, fault tolerance, fault removal, and fault forecasting. Reliability measurement plays a particularly important role in these methods, which are described in the next sections.

A software reliability model is comprised of certain assumptions about the system, relevant factors about the software, and mathematical functions relating the reliability to the factors. The most suitable models for reliability of SOAs are the ones based on architecture. The main reason is that they focused on the components and on the control and data flow between components. There are three distinct approaches in architecture-based reliability modelling: state-based models, additive models, and path-based models [4] [7]. State-based models analytically estimate reliability by using the control flow graph of a software architecture. Path-based models compute all possible execution paths, then calculate the reliability of each path by multiplying the reliabilities of the components along that path. The system reliability is the average path reliability over all paths. Finally, additive models describe each component’s reliability and the system reliability as Nonhomogeneous Poisson Processes (NHPP)[4].

SOA brings new challenges to the usual reliability models currently in use:

− Software systems are commonly developed and maintained by one entity with full control of its development. In SOA, the entire software system consists of an interacting group of autonomous services. These services are developed and provided by different (geographically distributed) stakeholders and they can be executed separately or together in a workflow. Each service provider offers varying degrees of reliability, and there is no guarantee that a service will be available and operating correctly.

− All major elements of the architecture are loosely coupled, which means that they have to be evaluated separately (with potential lack of information). Reliability should be guaranteed for the basic service, for the data flow, for composition of services and for the complete workflow.

− Service-oriented architectures are by nature dynamic, with service publication, registration, discovery, binding, and composition occurring while the system runs. During runtime new services can be added, existing services can be removed, and running services can fail. This dynamic character of SOA requires a reliability model which reacts at runtime to system changes to ensure accessibility of services.
Another point is the statelessness of services. In SOA, the services do not store any state information about a task. It means all data will be lost if a task gets interrupted. This problem must be also considered by a reliability model for SOA.

The above mentioned characteristics challenge new reliability models and solutions. The rest of this paper will discuss existing reliability solutions, define research challenges and describe a proposed solution for the defined problems.

2 Related Works

SORM, Service-Oriented Software Reliability Model [9], is one of the earliest architecture-based reliability model for SOA. It consists of two stages: group testing to evaluate the reliability of atomic services, followed by the evaluation of composite services through the analysis of components and their relationships. In this work, Tsai et al. use a group testing technique [10], from the medical field to detect faults. Assume a service $S$ has a group of functionally equivalent services $G_f$. When $S$ receives an input, $S$ sends it to all services in $G_f$. Each output (including that from $S$) is then sent to the voting service, and voted on. The current reliabilities of the services are used as weights for the voting. A service initially has its reliability set to zero. A fault is detected when the output of a service disagrees with the weighted majority output. The second evaluation stage deals with composite services. The method used tries to determine the reliability of each component and their relationships. Before further discussion, three assumptions about reliability in this model should be discuss. First, assignment operations never generate new failures. Second, a condition fails when any data associated with it fails. Finally, there is no cyclic dependency among ACDATE [9] entities.

In [11] a new service, ReServE, is proposed that will transparently recover the state of a business process, ensuring that it is consistently perceived by client and services. The main component of the ReServE-architecture, the Recovery Management Unit - RMU, records all communication (requests and responses) between client and services. For each service there is a Service Proxy Unit (SPU) that monitors the service and detects the failure. When a service fails, its SPU can initiate the rollback-recovery process.

A unified reliability modelling framework is defined in [3]. It provides a hierarchical reliability model. The reliability of simple services is addressed by considering data reliability and service pools as backup alternatives. Discrete Time Markov Chains (DTMCs) are used for analysing service composition reliability.

Using information published with each service, [12] presents an architecture-based reliability prediction model for SOAs. In this work, reliability is viewed as a measure of a services ability to successfully carry out its task when invoked. The final goal is to have an automatic and compositional reliability prediction method.
Every solution discussed above has its own focus and meets a part of SOA requirements. For example, according to the definition of Erl, statelessness is a principle of service design [14]. If a service request is interrupted, all execution data will be lost. A restart of the service request from the beginning will waste resources and produce unnecessary cost. The idea of Danilecki et al. [11] provides a solution to recover state of a service request. The solution of Tsai et al. [9] focus on the proper detection of failures. They assume the existence of a set of equivalent services that could be used in a voting system, to ensure identifying of unavailable services. The proposed model in [12] can be used for fault forecasting. Each related work presents a partial solution for reliability. In order to get a complete solution, first it is required to analyse characteristics of SOA and define corresponding requirements. After that, it should be discussed which existing solutions can be used to fulfill the defined requirements and which solutions should be newly developed. The next section discusses research challenges and describes a solution approach.

3 Discussion and Research Approach

In order to optimize the reliability of a system and to satisfy runtime requirements such as use of resources, performance, etc., a reliability management model is needed. The required model must be able to define failure-critical system components and decide the most suitable reliability technique. The redundancies for failure-critical (or failed) services must be deployed and configured automatically. This step alone implies some important questions which must be considered. The configuration and generation of redundancies and then the deployment and integration of them into running system (including service discovery, service binding, service execution) are most important issues. The focus of this PhD will be dealing with these issues to develop an automatic reliability management in SOA. The rest of this section presents the first architectural draft (see Fig. 1) of the proposed reliability management concept. It provides a general overview and describes some future contributions.

In Service-Oriented Architectures, the entire system can be separated into three layers: business layer, services layer and hardware layer[14]. The reliability of the target system must be considered on all these layers. Reliability can be guaranteed in three steps: fault detection, fault forecasting and fault elimination[8]. First the whole system must be monitored to detect failed system components (fault detection). Second, the desired reliability model must validate the reliability of all system components to identify possible faults before system failure (fault forecasting). Then the redundancies must be defined and automatically switched on when required (fault elimination).

In Fig. 1, the proposed solution architecture consists of different modules which map the steps defined above. The Module Monitoring component monitors all system components and inputs. It identifies failed components and informs the Management Module. This module also analyses inputs to identify malicious attacks (such as DoS attacks) on the system. The Module Monitoring component
Fig. 1. Reliability Management Model

collects some data about system components to support runtime learning process of reliability model (for example how many request per second can be executed - capacity of system components). The Module Reliability Validator calculates and analyses the reliability of system components. The Module Redundancy Installer generates and deploys redundancies. It configures the redundancies and integrates them into the system. Module Request Splitter distributes requests on the redundancies (through using some load balancing solutions [13]), if a system component achieves its maximum performance or redirects requests, if a system component fails. The Module Reliability Management collects data from other modules and makes runtime decisions for an appropriate reliability strategy, redundancy strategies, consensus strategies etc. It also manages the configuration and integration of the redundancies into the entire system. If the system is under outside attacks, the Management Module can abort inputs to avoid undesired system failures. Additionally, it notifies clients about modifications in the system.

4 Conclusion

In order to build dependable critical systems, we must consider reliability. Reliability ensures system users a performance continuity despite failures of weak system components. Dynamic and adaptive SOA systems entail some additional requirements to the common reliability solutions. To meet these requirements and to ensure reliability of SOA based software systems, a generic reliability management model is needed. The contribution of this PhD will be developing an efficient solution approaches to optimize the reliability of SOA. This paper analyses several existing proposals from other researchers in this area. It defines
some important requirements for SOA reliability, starts to design first architectural draft. This paper provides an overview of what has been achieved so far and shares with you some thoughts for the future work.

References