Increase Software Dependability via AOP-Based Runtime Adaptation

Shan Tang, Liping Li, and Bin Zhu

Abstract—Increasingly complex application systems operating in dynamic environments are forcing researchers to find ways of developing and evolving dependable software systems dynamically. Runtime Adaptation is a critical feature for dependable application systems to cope with changes in the operating conditions and to meet the required dependability. AOP can improve the modularity of self-adaptive software by encapsulating the crosscutting concerns into aspects cleanly. Therefore, in this paper we take the advantages of AOP and self-adaptive technique, and propose an AOP-based dynamic adaptation approach. We show how to increase system dependability by runtime adaptation from the perspective of system internal state and external environment. Meanwhile, we illustrate our approach by applying it to a simplified on-line train ticket reservation system.

Index Terms—AOP, dynamic adaptation, dependable software, separation of concerns.

I. INTRODUCTION

Modern software systems live in highly dynamic environments, making the systems more and more complicated, error-prone and unstable. This phenomenon has brought new problems and challenges to the dependability of software systems and has led the software engineering community to investigate innovative ways of developing, deploying, managing and evolving dependable software systems [1], [2].

Software fault-tolerance technology is an important method to guarantee the dependability of the systems. But, the traditional fault-tolerance methods have several disadvantages: costs are too high that multiple redundant implementation versions are needed; low modularization makes modeling and predictive analysis hard; unable to adapt to various and complex run-time environments. In recent years, researchers in the field of dependable software get down to associating their research work with self-adaptive software systems. Comparing to traditional redundancy and diversity based fault-tolerance methods, self-adaptive systems can dynamically adjust their behaviors and structures, thus improve reliability at run time. This leads to a more flexible and effective way towards software dependability [1].

However, current works address dynamic adaptation often loses sight of the existence of system crosscutting concerns (e.g. safety and quality etc.), causing the system hard to maintain and evolve. While Aspect-Oriented Software Development (AOSD) can ease this kind of problem. With the application of Aspect-oriented Programming (AOP) [3], the crosscutting concerns can be cleanly encapsulated into aspects and modeled as components, and improve the modularity of software. Therefore, we can implement dynamic adaptation by weaving aspects into the target system at runtime.

In this paper, we present an AOP-based autonomic adaptation approach to increase system dependability by combining AOP and self-adaptive technology. We illustrate our approach by applying it to a simplified on-line train ticket reservation system, and show how to implement a dependable system which can adapt itself at runtime to be continuously available under varying conditions (e.g. the system internal state, external environments, system requirements, etc.).

The remainder of this paper is organized as follows: Section II provides some background knowledge of dynamic adaptation and AOP technology. Section III presents the motivating example at first, and then describes how to increase system dependability by AOP-based runtime adaptation. Section IV some related works are discussed. Finally, conclusion, evaluation and future work are presented in Section V.

II. PRELIMINARIES

Our research work is founded on dynamic adaptation and AOP technology. In this section, we recall some key concepts of these two research domains, used as basis for our approach.

A. Dynamic Adaptation

Self-adaptive software system aims to adjust itself to changes during its operation. These changes may stem from the software system’s self (internal causes, e.g., failure) or its environments (external events, e.g., increasing requests from users). Such a system is required to monitor itself and its context, detect significant changes, decide how to react, and act to execute such decisions. These processes depend on adaptation properties [4].

Adaptation properties are often called as self-* properties. IBM is one of the leading organizations developing self-adaptive software systems. In the IBM architectural blueprint [5], IBM introduces a set of four properties that a self-adaptive system needs to possess. They are:

Self-Configuring: is the capability of reconfiguring automatically and dynamically in response to changing environments.

Self-Healing: is the capability of discovering, diagnosing,
and reacting to disruptions.

Self-Optimizing: is the capability of tuning system automatically to meet end-user or business needs.

Self-Protecting: is the capability of anticipating, detecting and protecting against threats from anywhere.

If a system possesses all of the four features, we call it a fully self-adaptive system. However, in real application environment, a software system usually just need to be partially self-adaptive, which is developed with a sub-set of these features.

B. AOP Technology

Separation of Concerns (SoC) is an emerging methodology and a good design principle for separating a computer program into distinct sections, such that each section addresses a separate concern. And AOP is one of the state-of-the-art SoC techniques.

In AOP approaches, “join point” and “aspect” are very important concepts to modularize crosscutting concerns. A join point is a well-defined point in the code at which our concerns crosscut the application. Typically, there are many join points for each concern. An aspect specifies a set of join points in the target application where the normal execution is altered and describes the additional behavior that should be executed. The aspect logic is woven into the target application by aspect weavers.

In existing AOSD approaches, different alternatives to tackle the separation of concerns issue, even though the final goal is always the same. These approaches differ mainly with regard to when aspects are applied (before, after or around), the join points where aspects are applied, where the weaving information is placed, whether the weaving process is static or dynamic [6].

Static AOP is a relatively mature technique. Such as AspectJ [7], a compile-time, Java-based AOP language, extends Java with three new programming features: pointcuts, advice, and aspects. Recent years, there are many dynamic AOP techniques being developed. Dynamic AOP (e.g. Spring AOP [8], AspectWerkz [9]) extends static AOP by providing dynamic weaving mechanism, without stopping the whole system. A complete description of AOSD technologies can be found in [10].

III. THE APPROACH

In this section, we introduce the motivating example at first, and then discuss how to increase system dependability by AOP-based dynamic adaptation, from the perspective of system internal state and external environment.

A. Motivating Example

Here we introduce a typical on-line Train Ticket Reservation System (TTRS) to illustrate our approach. TTRS is a kind of application that must be highly dependable, especially during the Spring Festival travel period. The simplified software architecture of TTRS is shown in Fig. 1.

To keep the example manageable, we make a few simplifying assumptions. We assume that the software architecture of TTRS mainly consists of five composite components. They are Reservation Business, Reservation GUI, Account Manager, Database Manager and Bank Delegate. Reservation Business encapsulates all the business functions that related to reservation processing. For example enables online-customers to query the train routes and tickets information, provides payment service, etc. Reservation GUI is the user interface component. Account Manager deals with the registration and management of users accounts. Database Manager is responsible for implementing the functions associated with database access. Bank Delegate is in charge of communicating with various banking systems.

In order to support dynamic adaptation, we model the system in terms of components and connectors. A component is an encapsulation of a computational unit. And it specifies its provided services and required services by its provide interfaces and require interfaces. Connectors mediate interactions among components by establishing the rules governing component interaction and specifying any auxiliary mechanisms required [11].

![Fig. 1. The simplified software architecture of TTRS.](image)

B. AOP-Based Runtime Adaptation

In this section, we describe how to implement runtime adaptation by using AOP technology.

Generally speaking, self-adaptive software system modifies its own structure or behavior to correct faults, to improve performance or other attributes, or to adapt the product to a changing operating environment [12]. So we will exemplify our approach from these points of view.

At first, let us illustrate how to implement self-tuning to response to the changing system internal states. Consider such an adaptation scenario: there are two message notification components GMC and TMC. GMC is employed to display vivid graphical information to online-users when the load on the server is normal. TMC is used to display pure textual information when the server is highly loaded. Under normal condition, the system instantiates and loads GMC to provide better service. However when the system comes under heavy load, e.g. during peak times, we assume that the system will instantiate and load TMC for improving the request-response time. And if the system load drops to a normal state again, the system needs to adjust itself back to graphical mode dynamically. From this scenario, the system should have the self-adaptive ability to ensure it can switch back and forth from textual mode to graphical mode automatically. That is to say, the system should load and
unload GMC and TMC dynamically for balancing server utilization with service response time.

The corresponding statechart diagram of the adaptation process is illustrated in Fig. 2. GMC is loaded and TMC is unloaded dynamically when the system is performing the `displayGrapMessage()` function. Otherwise, the system will load TMC and unload GMC when it is performing the `displayTextMessage()` function.

![Fig. 2. The statechart diagram of the adaptation process.](image)

Notice that these two message notification components affect some business functions, e.g. register/login an account, pay for orders, and reserve tickets. Therefore, both of GMC and TMC have relationships with other basic components of the system. This means that they crosscut the others components. For making the system easier to maintain and adapt, we employ AOP technology to solve this problem. Concretely, system crosscutting concerns are encapsulated into aspect beans and implemented as aspects components. An aspect bean is an extension of the Java Bean component that specifies crosscutting behavior in a reusable manner.

Fig. 3 shows part of the XML-based beans configuration files under the graphical mode (Fig. 3a) and the textual mode (Fig. 3b) respectively. It’s very efficient to implement the dynamic adaptation scenario we mentioned above, just by modifying and reloading the configuration file at runtime.

![Fig. 3. The XML-based beans configuration file.](image)

This redirection aspect does not specify any concrete e-bank online payment components. As we mentioned in section III, the connectors mediate interactions among components. Therefore, the specific mapping relationships are implemented by various connectors. Here, we omit the implementation code of the connectors due to the space limitation.

IV. RELATED WORK

We focus on implementing dynamic and self-adaptive software to increase system dependability. Our approach has been inspired by many other approaches.

For example, in [13], [14] J. Kramer and J. Megee proposed a dynamic architectural reconfiguration approach where self-managed software architecture is one in which components automatically configure their interaction in a way that is compatible with an overall architectural specification and achieves the goals of the system. They defined a three layer reference model consisting of component control layer, change management layer and goal management layer. And the component control layer is to reconfigure the software components and ensures application consistency and avoids undesirable transient behavior. At the change management layer, decentralized configuration management is required which can tolerate inconsistent views of the system state, but still converge to a satisfactory stable state. Finally, some form of on-line planning is required at the goal management layer.

D. Menasce et al. [15], [16] proposed a model-driven framework (SASSY) for runtime self-architecting of distributed service-oriented software systems. This
framework automatically generates candidate software architectures and selects the one that best serves stakeholder-defined, scenario-based quality-of-service (QoS) goals. Self-architecting occurs during initial system deployment and at runtime, thus making systems self-adaptive, self-healing, self-managing, and self-optimizing.

In [17] Junrong Shen et al. established a platform-independent model to address the fundamental issues of dynamic update mechanisms in the form of a special kind of architectural connector. In their work, the model specifies the constituent components of a dynamic update mechanism, the behaviors of these components and the interactions between them. All these support deep understanding of existing dynamic update mechanisms. And they demonstrated how to use the model to guide the implementation of dynamic update in a J2EE application server. In our approach, we borrow the redirection idea from their Dynamic Update Connector (DUC) to implement self-healing dynamically.

Aspect-Oriented community also proposed some adaptability works to weave aspects into running software system. These works are mainly developed in Java or C#.

For instance, [18] presented a component and aspect model that combines CBSD and AOSD disciplines. This model introduces aspects as special connectors between components. It achieves a high degree of independence between components and aspects, which makes them more reusable. Therefore, based on this model, developers can build complete application systems in a short time.

In [19] Eddy Truyen and Wouter Joosen adopt the well-accepted reference model of autonomic computing [20] as the foundation for studying self-adaptive frameworks and demonstrate where and how AOP can be applied in the architecture of self-adaptive systems. In [21], Tesanovic et al. proposed a novel concept of aspectual component-based real-time system development (ACCORD) and applied it successfully in the development of a real-time database system.

V. CONCLUSION, EVALUATION AND FUTURE WORK

This paper has discussed how dynamic adaptation can be applied to dependable software system. Unlike traditional adaptation approaches, in which the adaptation logic is tangled with the business logic, we separate the dynamic adaptation concern from the business function code by using AOP. It complies with the "separation of concerns" principle and makes the system easier to maintain and reuse, which is a fundamental property for developing large-scale complex software systems.

In our approach, we use the join points to specify where the adaptation should take place, and define the rules to specify the conditions when an adaptation should occur manually. To some extent, this is a static solution for implementing an adaptive system. Although it is feasible for a bare-bones system, it is not possible to specify all the adaptation rules for a large-scale complex system.

Therefore, in order to manage the increasing complexity of software systems and uncertainty of their environments, we need to introduce feedback control theory into our future work. By exploring feedback control loops within existing self-adaptive systems, it will deal with the continuously changing environment and emerging unpredictable requirements much better.

ACKNOWLEDGMENT

This work is supported by the Innovation Program of Shanghai Municipal Education Commission under Grant No.11YZ251, and the Shanghai University Scientific Selection and Cultivation for Outstanding Young Teachers in Special Fund under Grant No.ZZegd12008.

We would like to thank to Professor Xing Zhenchang (School of Computer Engineering, Nanyang Technological University, Singapore) for his commendable comments on this paper.

REFERENCES


Shan Tang was born in Taojiang County, Hunan Province, China on 30th March 1982. She received her B.S. degree in computer science and technology from National University of Defense and Technology, Changsha, China in 2004. She received her Ph.D. degree in computer software and theory from Fudan University, Shanghai, China in 2011. Currently, she is a lecturer of the school of computer and information at Shanghai Second Polytechnic University. Her research interests include software component and architecture, self-adaptive software system, and dependable computing. Dr. Tang is a member of CCF (China Computer Federation) and IACSIT (International Association of Computer Science and Information Technology).

Shan Tang

Liping Li was born in Xiangtan County, Hunan Province, China on 17th March 1976. She received her M.S. degree in software engineering from Tongji University, Shanghai, China in 2006. She received her Ph.D. degree in computer application technology from Shanghai University, Shanghai, China in 2011. She is currently an associate professor of the school of computer and information at Shanghai Second Polytechnic University. Her research interests include software engineering, software testing and formal methods. Dr. Li is a member of CCF (China Computer Federation).

Liping Li

Bin Zhu was born in Ninghai County, Zhejiang Province, China on 11th April 1976. He received his M.S. degree in computer application technology from Shanghai University, Shanghai, China in 2004. He received his Ph.D. degree in computer application technology from Shanghai University, Shanghai, China in 2011. Currently, he is an associate professor of the school of computer and information at Shanghai Second Polytechnic University. His research interests include software engineering and formal methods. Dr. Zhu is a member of CCF (China Computer Federation).

Bin Zhu