Towards Sustainability-Oriented Development of Dynamic Reconfigurable Software Systems

Shan Tang, Liping Li, Wenjing Yang, Jianxin Xue
School of Computer and Information
Shanghai Second Polytechnic University
Shanghai, China
{tangshan, liliping, wjyang, jxxue}@sspu.edu.cn

Abstract—Sustainability should be supported by modern software engineering methods to guarantee reliability of the running systems. Dynamic reconfiguration is an important technology implementing the sustainability goal. However, building dynamic reconfigurable software system cost-effectively and in a predictable manner is a major engineering challenge. Aiming at solving this problem, this paper combines AOSD approach and feedback control theory to explore how sustainability goal can be obtained through dynamic reconfiguration. At the same time, a case study is employed to illustrate and evaluate our approach.

Keywords—dynamic reconfiguration; AOP; feedback loop; sustainability; runtime

I. INTRODUCTION

Variability is an essential characteristic of software systems. Providing high-quality software in the face of uncertainties (such as dealing with new user needs, changing availability of resources, and faults that are difficult to predict) raises fundamental challenges to software engineers. For making software system can run continuously and can continuously provide service, the system should dynamically adapt its behavior at run-time in response to changing requirements of users and running environment. This is a fundamental requirement that many modern systems must satisfy [1, 20].

Dynamic reconfiguration is a key technology guaranteeing the sustainability of the systems. In recent years, dynamic reconfigurable software systems have been researched and applied in different application domain, and have made a lot of achievements. However, building dynamic reconfigurable software system cost-effectively and there are numerous remaining research problems in developing this kind of systems.

An obvious research problem is that current works address dynamic reconfiguration often loses sight of the existence of system crosscutting concerns (e.g. the quality attribute), causing the system hard to maintain and evolve. Aspect-oriented Programming (AOP) can ease this kind of problem. With the application of AOP [2], the crosscutting concerns can be cleanly encapsulated into aspects and modeled as components, and improve the modularity of software.

Another research problem is: how can we effectively design software so that it can respond to changes in execution environments, without human intervention? Over the past century, feedback control loop has been proposed as an effective model for controlling dynamic behavior of mechanical, electrical, fluid and chemical systems in the corresponding fields of engineering. Inspired by these works, we bring it into the development of dynamic reconfigurable software systems.

In this paper, we take the advantages of AOP technology and feedback control theory to explore how sustainability goal can be obtained through dynamic reconfiguration. In our approach, AOP and feedback loop are regarded as first-class entities. A simplified on-line train ticket reservation system is employed to illustrate our approach throughout this paper.

The remainder of this paper is organized as follows: Section II provides some background knowledge. Section III presents how to implement system’s sustainability goal from the perspective of dynamic reconfiguration by applying Aspect-oriented Software Development (AOSD) approach and feedback control theory. Section IV introduces the implementation work and evaluates our approach. And some related works are discussed in section V. Finally, conclusion and future work are presented in section VI.

II. BACKGROUND KNOWLEDGE

Our research work is founded on MAPE-K control loop, AOSD approach and feedback control theory.

A. MAPE-K Control Loop

Dynamic reconfiguration systems modify themselves at runtime in order to control the satisfaction of their requirements under changing environmental conditions. Therefore, they are required to monitor themselves and their context’s environment, detect significant changes, decide how to react, and act to execute such decisions. All of these behaviors are typically realized using a control loop, which is by means of four components that are responsible for the primary functions of dynamic reconfiguration: Monitor, Analyze, Plan, and Execute, often referred to as the MAPE loop [3].
Figure 1 exemplifies an autonomic controller based upon the MAPE-K loop (monitor, analyze, plan, and execute), whereby data collected by probes is utilized to produce a model of the system and its environment. Through the analysis of such models, we are able to detect system’s state and its environmental conditions, and diagnose performance problems or to detect failures, deciding on how to resolve the problem (e.g., via dynamic load-balancing or healing), and acting to effect the planning decisions made [4].

In existing AOSD approaches, there are 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 [5].

Recent years, there are many dynamic AOP techniques being developed. Dynamic AOP (e.g. Spring AOP [6], AspectWerkz [7]) extends static AOP by providing dynamic weaving mechanism, without stopping the whole system. And our implementation is based on Spring AOP. A complete description of AOSD technologies can be found in [8].

C. Feedback Control Loop

The feedback control loop (closed loop) is the cornerstone structure for building controllers for different kinds of dynamic software systems [9]. Controllers are used to maintain specified properties of the outputs of the target system at (sufficiently near) the given reference values called the set points.

Figure 2 shows a classic feedback loop structure for a target system. This feedback loop performs dynamic control by comparing the target system output to the control objective given as desired state (the set points), yielding the control error (delta), and then adjusting the controlling input based on the output from a sensor, to make the target system to behave closer to the set points.
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 DisImg and DisTxt 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 evolve, 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.

Figure 4 shows part of the XML-based configuration file for graphical display mode. It’s very efficient to implement the dynamic reconfiguration scenario we mentioned above, just by modifying (the values of “bean id” and “ref”) and reloading the corresponding configuration file at runtime.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
  <bean id="showGraphicalMessage" class="org.springframework.aop.support.NameMatchMethodPointcutAdvisor">
    <property name="names" ref="DisImg" />
    <property name="mappings">
      <list>
        <value>UserRegister</value>
        <value>OrderPayment</value>
        <value>TicketReservation</value>
      </list>
    </property>
  </bean>
</beans>
```

Figure 4. The XML-based configuration file for graphical display mode

Then, we describe how to implement dynamic reconfiguration to cope with the problems caused by external environment. Suppose that there are various e-bank online payment service components provided by different banks (e.g. BOC, ICBC, ABC, CCB, etc.). Usually, online-users would like to choose a specific e-bank online payment component (e.g. ICBC) to pay for their order.

As all we know that banks server often need to upgrade their service. If one bank server need to be upgraded on someday, and cannot provide payment service to their customers during the upgrade. It is unacceptable for the online users that they couldn’t finish their payment after they submit their ticket order due to the service unavailable caused by upgrade, particularly in the case of emergency. In this situation in order to ensure the TTRS application system can continue to provide payment services for all the users, we need to improve sustainability by reconfiguring the target system at runtime to adapt to the environmental failure.

As an online third-party payment platform, AliPay can deal with payment transactions for more than one hundred banks and financial institutions. Assume that AliPay also supports this bank which is in the process of being upgraded. Therefore, the service requests to the upgrading bank will not be refused if we can redirect all the requests to AliPay dynamically. In this scenario, this kind of redirection crosscuts all the e-bank online payment components that AliPay supports. So, we can encapsulate it into a redirection aspect component.

In our work, we model the system in terms of components and connectors for supporting dynamic reconfiguration. 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 [10].

Figure 5 shows the interactions among e-bank online payment components, redirection aspect, and AliPay server. In this interaction diagram, each e-bank online payment component uses services provided by AliPay, and calls the services through connectors, which forward the request to AliPay via the redirection aspect and pass the result to e-bank online payment components. During the process of updating the e-bank server, all the requests from its online-user are buffered and redirected to AliPay to handle sequentially. When the update is finished, all the buffered requests will be redirected back to this e-bank server if these requests have not been handled yet.

```
Figure 5. The interactions among e-banks, redirection aspect, and AliPay server
```

B. Feedback Loop based Dynamic Reconfiguration

As the previously mentioned scenario, the value of the concurrent accesses threshold (concurrAcce_Threshold) is a key factor to decide when and how to reconfigure the target system at runtime. In most of the existing approaches, this value is set statically at design time that is very difficult to guide the system to reach an optimal state.

If the value is estimated too high, it will cause the system overloads or even collapses. While if the value is estimated too low, it will lead to a waste of system resources. Therefore, we need to set this value dynamically according to the runtime environment. Here we employ feedback control loop to help us to implement that the system can run reliably and the system resources can be used efficiently.

```
Figure 6. The feedback control loop with architectural components of the MAPE-K loop
```

Figure 6 shows the feedback control loop with architectural components of the MAPE-K loop. This diagram establishes the
mapping relationship of functional elements between feedback loop and MAPE-K loop.

Monitor is responsible for sensing information that is relevant to dynamic reconfiguration, including the target system’s internal variables corresponding to quality attribute, and also the external context. In our work, we use probes to collect data from the executing system and its context about its current state.

Analyzer based on the high-level system requirement and the sensed information which is transferred from monitor, compares target system performance and desired state, determines whether a reconfiguration must be triggered, and infers the control error (delta) and forwards this error to Dynamic Configurator.

Planner selects strategies and generates the reconfiguration plan according to the error information. In this case, the plan refers to how to set the value of concurrAcce_Threshold. Meanwhile, planner computes the necessary control actions to be instrumented into the target system.

Executor interprets and executes each of the operations specified in the plan and instruments them into the target system. Thereby adapts target system to obtain the desired state.

IV. IMPLEMENTATION AND EVALUATION

We have designed our experimental study to evaluate our work. Our experiment environment is Apache Tomcat 8.0.5 running on a server with Intel(R) Core(TM) i7 CPU and 6G memory.

We implement our AOP-based dynamic reconfiguration approach based on Spring framework. And we adopt the Interface ServletContextListener and the Class TimerTask to implement the feedback control loop. We use JMeter to perform stress test and simulate the runtime environment. During the test, a lot of business operations are recorded in the Tomcat’s configuration file (server.xml). We analyze the recorded data and produce more runtime state information, like the response time, the total number of requests, and the system’s throughput, etc.

In our case study, we compared the performance of the original system which does not have the dynamic reconfigurable ability and the system which has been instrumented with our dynamic reconfiguration approach based on the original system. From the analysis data, we found that the response time of the dynamic system is a bit slower than the original one when the concurrent access less than 200. But the dynamic system has an obvious better performance than the original system when the concurrent access exceeds 500. It shows a significant improvement for highly concurrent application system.

V. RELATED WORK

This work has been inspired by many other approaches. For instance, [11] 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 [12] Eddy Truyen and Wouter Joosen demonstrated where and how AOP can be applied in the architecture of self-adaptive systems. In [13], 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.

In [14], Shen JR et al. 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.

D.Sykes et al. proposed a dynamic architectural reconfiguration approach in [15] 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.

D.Menasce et al. [16-17] 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 [18], Yuriy Brun et al. explored feedback loops from the perspective of control engineering and within existing self-adaptive systems in nature and biology. At first, they investigated feedback loops as a key aspect of engineering self-adaptive systems. Then they outlined basic principles of feedback loops and demonstrated their importance and potential benefits for understanding self-adaptive systems. Finally, they described control engineering and biologically inspired approaches for self-adaptation.

In [1], A.Filieri et al. explored how continuous reconfigurations through dynamic binding can be obtained as the solution of a discrete-time feedback control problem. They regarded software system as a broken down into constituent blocks having no hard-wired connectors. Connector is created dynamically when the binding between two blocks is established. They applied control theory to automatically derive how the bindings must evolve over time.

[19] applied AI planning and implemented a general closed control loop for selecting a configuration and deciding how to change the system at runtime. Their approach worked well for small examples, but they encountered a critical complexity limit, depending on the number of used “objects” (resources) and the number of “exists” operators.

Danny Weyns et al.[20] investigated the question whether external feedback loops provide more effective engineering solutions than internal mechanisms. And they claimed that external MAPE loops do simplify the design in terms of control flow primitives for the processes. The significant reduction of
control flow complexity increases understandability of the design, and can improve maintainability and testability of the system. The use of external MAPE loops reduces fault density. Reduced fault density increases the quality and reliability of the software design, and adds to customer satisfaction. And external MAPE loops realize a separation of concerns, which yields easier to understand designs, having a positive effect on productivity.

VI. CONCLUSION AND FUTURE WORK

In this paper, we presented an approach which combines AOSD technology and feedback control theory to implement a dynamic reconfiguration software system to achieve system’s sustainability goal.

We have discussed how to implement dynamic reconfiguration under different application scenarios by using AOP and feedback control theory respectively. The main advantages include that: Firstly, our work complies with the “separation of concerns” principle. We separate the dynamic reconfiguration concern from the business function code by using AOP. It makes the system easier to maintain and reuse. Secondly, we use feedback control technology to set the values of key variables dynamically. Comparing to most of existing methods which set the values statically at design time, it can reflect the changing environment in real time and provide a more reasonable way to guide the system to run in an optimal state.

However, we just study how to reconfigure system at runtime driven by a single object. In real-world applications, we often need to take into account multiple objects at the same time, and even include conflict objects (e.g. security vs response time). Therefore, we will research runtime tradeoff about different quality objects and functional objects in our future work.

ACKNOWLEDGMENT

This work is supported by the Innovation Program of Shanghai Municipal Education Commission under Grant No.11YZ251, the Startup Funding of SSPU Scientific Research under Grant No.A20XK11X003 and No.EDG14XQD13, the Shanghai University Scientific Selection and Cultivation for Outstanding Young Teachers in Special Fund under Grant No.ZZegd12008, and the Key Disciplines of Software Engineering of Shanghai Second Polytechnic University under Grant No. XXKZD1301.

REFERENCES


