A Service-Oriented Non-intrusive Software Fault-Tolerant Programming Model

Shuanghui Yi *, Rong Li
National Key Laboratory for Complex Systems Simulation, Beijing, China.

* Corresponding author. Email: flyysh@126.com
Manuscript submitted January 13, 2019; accepted March 7, 2019.
doi: 10.17706/jsw.14.8.350-355

Abstract: Only through good design can we obtain the high dependability of the software. How to establish the high dependability of the software from the design is the current problem to be solved. Existing object-oriented programming methods and techniques cannot adapt to service-oriented credibility design requirements. This paper will propose a non-intrusive software fault-tolerant programming model based on the research of the fault-tolerant ability of service-affecting service. By establishing service fault-tolerant design and development model, the flexible compilation of trusted attributes is realized.

Key words: Fault-tolerance, programming model, service-oriented.

1. Introduction

When developing and designing fault-tolerant software (or extending a system that is not fault-tolerant), developers need to focus on the code that is used to implement fault-tolerant logic (that is, non-functional code, which is often business-independent and "cross-cuts" a number of functional modules), making the fault-tolerant system intrusive, increasing the development difficulty and maintenance costs of the system. On the other hand, when specifying different fault-tolerant strategies or performance indicators, the behavior of each service component (especially the component with fault-tolerant logic) in fault-tolerant software may be completely different, while the traditional object-oriented analysis and design methods lack fault-tolerant software. The unified description mechanism of the system in the design stage reduces the manageability and reusability of fault-tolerant logic [1], [2].

2. Programming Model Framework

Through analysis, we believe that the root cause of the above problems lies in the lack of common programming-level fault-tolerant model support based on existing platforms and technologies. This project introduces AOP and replica technology into the design phase of fault-tolerant software, and proposes a non-intrusive fault-tolerant software model with software redundancy replica as fault-tolerant mechanism.

The model describes the quality of service and performance indicators of the software system with a declarative fault-tolerant strategy. Through the aspect-oriented UML extension mechanism [3], the structure of the fault-tolerant aspect and the interaction between the replicas are accurately characterized in the design phase, and the interception and proxy mechanisms are implemented at runtime. The specified fault-tolerant aspect is dynamically woven into the software system, ultimately achieving the goal of eliminating or minimizing the intrusiveness of the fault-tolerant logic in the system to the business logic.

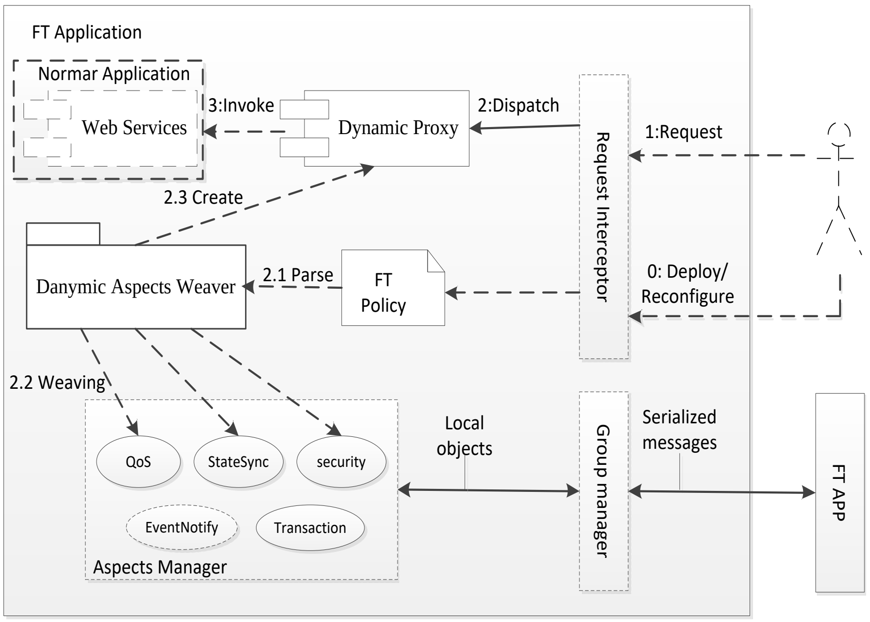

Fig. 1. Non-intrusive fault tolerant software programming model.

Eliminating or reducing intrusion can be divided into three steps: aspect separation, aspect realization, and aspect weaving. For our fault-tolerant model, the aspect separation is mainly reflected in the fault-tolerant strategy described by various fault-tolerant indicators and service quality; the aspect implementation is to encode the various fault-tolerant logic; and the aspect-cut weaving is the fault-tolerance described by the aspect. The logic "adds" to the business logic contained in the system, ultimately making the software system fault-tolerant. The model is shown as Fig. 1.

The typical working scenario of the model is as follows:

1) The service requester invokes a service request. In general, the requester is the client of requesting a service, or it may be a member of a cluster formed by multiple redundant copies of software.
2) The service request is intercepted by the Request Interceptor and forwarded to the proxy (Dynamic Proxy). The request interceptor plays the role of a dispatcher and is the only visible interface that the fault-tolerant application (cluster member or service group) exposes to the outside world. Its role is: 1) to uniformly control and manage all requests that should be sent to the service (such as initialization, authentication, permission checking, etc.); 2) decoupling the dependencies between requester and executor. When the service requester requests access to the service's business logic, we set up a proxy for each service of the original application for non-intrusive fault tolerance purposes. Proxy is an intermediary between requests interceptors and original services, and is woven into fault-tolerant logic to ensure various performance metrics and quality of service. On the first invoking(access to service), the proxy has not yet been created, then: 3) Dynamic Aspects Weaver loads and parses the fault tolerance policy (FT Policy) for application configuration, which defines various quality of service and fault detection /recovery mechanism at a higher level (generally described in XML format), as shown in Fig. 1. 2) According to the fault-tolerant strategy, the dynamic aspect weaver matches the relevant aspects in the Aspects Manager, and weaves the logic contained in these aspects into the corresponding services, and finally creates the proxy service, as shown in Fig. 1(2.2, 2.3).
3) The proxy invokes the business logic of the original service according to the service request. The proxy does not contain the business logic code of its corresponding original service, but only the forwarding code. The real performer of the business logic is still in the original service. While

completing the business logic, the proxy service is also responsible for completing fault-tolerant logic (such as event notification, quality of service, state synchronization, etc.) and other non-critical logic (logs, transactions, security, etc.) whose execution details are completely transparent to the service requester.

In terms of fault tolerance mechanism, the model adopts the idea of redundant copies, that is, multiple copies of the key services contained in the same software form a cluster, and each copy is a member of the cluster. How to ensure the state consistency among the members of the cluster is not described in this paper.

The implementation of the non-intrusive fault-tolerant software programming model faces three key questions: How is the fault-tolerant strategy expressed? How to build the model of fault-tolerant aspect? How is fault-tolerant logic woven? In response to these three questions, the research team initially considered the following technical approaches to solve them.

3. Problem Solution

3.1. Declarative Fault Tolerance Strategy

Based on the characteristics of the current mainstream object-oriented programming language, the better way to eliminate intrusion is declarative (or configuration) programming, that is, let the software system express its various performance indicators or quality of service in a non-programming way, then load and analyze these requirements at runtime and dynamically "weave" them into the system. The fault-tolerant strategy describes the various service quality and performance indicators that the fault-tolerant system should be able to achieve, and the adjustment behavior and adaptation mechanism that the system should take actively when the error occurs or the related indicators are not met. The pre-designed declarative fault-tolerant strategy is as follows:

```
<? xml version="1.0" encoding="UTF-8"?>
<policy xmlns="http://www.bise.cn/schema/policy"
        xmlns:ft="http:// www.bise.cn/schema/ft"
        xmlns:qos="http:// www.bise.cn/schema/qos"...>
  <ft:cluster target="*.*.*（*）">
    <transport>bise.cn.transport</transport>
    <config>udp.xml</config>
    <jndi>topic/FT_APP</jndi>
    <properties>
      <property name="msg_max_bytes" value="10240"/>
      <property name="skip_suspected_members"value="true"/>
      <property name="processing_delay" value="0"/>
      <property name="excluding_self" value="false"/>
    </properties>
    <view>
      <coordinator>first_active</coordinator>
      <fd_interval>400</fd_interval>
      <fd_timeout>8000</fd_timeout>
      <merge>auto</merge>
    </view>
  </ft:cluster>
  <qos:template name="crud">
    <response>
```

```
        <target method=="*.biz.*Impl.add*（*）|upd*（*）|del*（*）"readonly="false">
        <cascade level="2"/>
        <trigger timeout="1000" operation="rollback"/>
        <tx advice="crud_tx_advice"/>
        </target>
      </response>
      <load_balancing target="*.sevice.*.*（*）">
        <trigger>
          <property name="cpu" value="80" logic="and"/>
          <property name="memory" value="60" logic="or"/>
        </trigger>
        <strategy excluding="true">most_lazy</ strategy>
        <default>last_active</default>
      </load_balancing>
    </qos:template>
  </policy>
```

During the system operation period, the fault-tolerant policy can be replaced at any time. The system periodically polls its version and status and parses it to meet different service quality and performance indicators.

3.2. Fault Tolerant Aspect Modeling

The modeling of fault-tolerant aspects is essentially a process of describing the structure and behavior of key services and their replicas related to fault-tolerant logic. Different from the traditional object-oriented design method, the aspect-oriented design (AOD) method emphasizes extracting and describing the aspect in the design stage to improve the reciprocity of the aspect. Our model treats each fault-tolerant aspect of a service-oriented software system as a set of roles, and defines each role as a set of properties. Taking into account the lack of UML language widely used in industry, the modeling capability of the aspect is intended to describe the fault-tolerant aspect using the UML extension mechanism [4] combined with the Object Constraint Language (OCL). The UML extension mechanism of the fault-tolerant aspect designed by the research group is shown in the figure below.

Fig. 2 shows the structure of the replica state synchronization aspect (StatusSynch) and the interaction between replicas. Member is a redundant copy of the service that constitutes a group. Multiple members form a group with a subject/observer relationship. This relationship is implemented by the StatusSynch aspect.

It can be seen that the StatusSynch aspect actually acted as an agreement between Group and Member. The StatusSynch aspect describes its properties and behaviors in a formal way, such as member joining/leaving, state returning/synchronization, etc., in addition to defining the prerequisite/postconditions for each behavior. After the StatusSynch aspect is initialized by the Aspect Manager, the specified target module is weaved according to the information specified in the policy file. Thereafter, while the user's operation request to the target module is processed, the fault-tolerant logic written in the StatusSynch aspect is responsible for finding the context information of the Group and completing the related operations such as state synchronization[5]. Therefore, the original target module does not realize the existence of the StatusSynch aspect and the Group, thus eliminating the aforementioned intrusiveness.

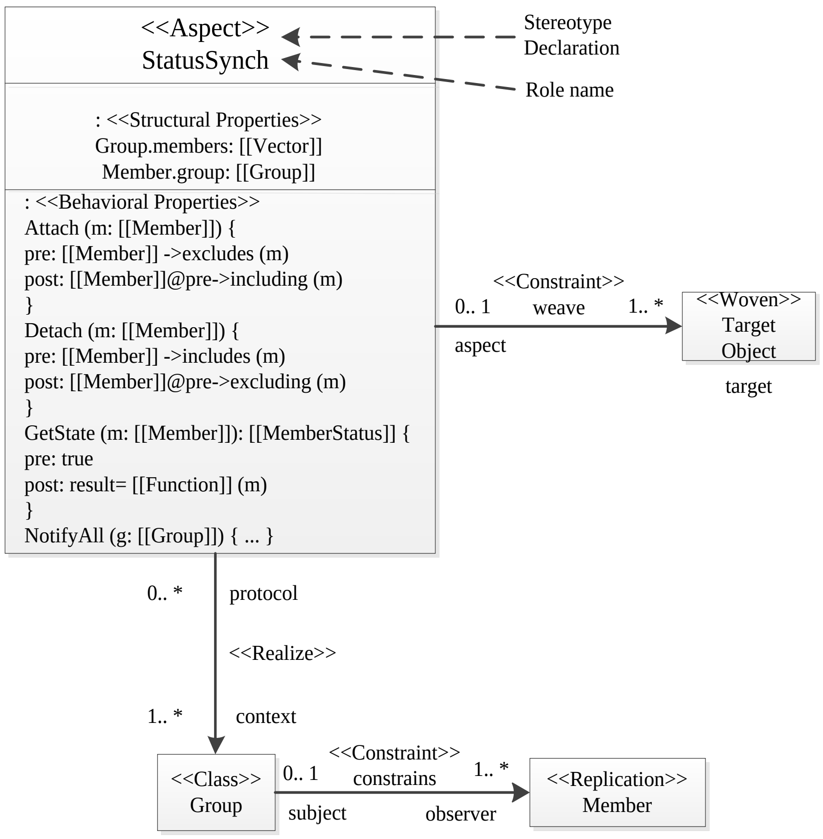

Fig. 2. Fault-tolerant aspect modeling method based on copy mechanism.

3.3. Fault-Tolerant Aspect Weaving

At present, there are two ways to implement aspect weaving: one is static weaving, that is, embedding additional code by modifying source code or compiled binary file at compile time, post compile time or load time. The other is dynamic weaving, which implements code embedding at runtime through techniques such as proxies. The static weaving provides better performance, but lacks the flexibility to adjust the behavior of the aspect at any time during the runtime, which is critically useful for applications that need to flexibly adjust the fault-tolerant strategy based on changing requirements and high-level business goals at any time. Therefore, the research team takes the approach of dynamic weaving to achieve fault-tolerant logic dynamic injection.

Taking the Java platform as an example, the JDK introduced the dynamic proxy mechanism from version 1.3. The essence of dynamic proxy mechanism is to introduce a mediator (ie, proxy) between the client and the service, which decouples the dependencies of them ,and can dynamically add certain behaviors (such as fault-tolerant logic) to the proxy at runtime. However, dynamic proxies using JDK must have a premise that the class being proxied must implement one or more interfaces, but for most software services (which may be a class), this premise is often not met. For services that do not implement any interfaces, we can implement dynamic weaving of additional logic through an interception mechanism. The specific method is: firstly, we should dynamically generate a subclass of the target class, then override the non-final method of the target class, and set a callback for it. After that, the method calls to the target class will be transferred to the user-defined interceptors (interceptors), so that the fault-tolerant logic can be dynamically woven to the appropriate location of the target class.

4. Conclusion

The paper introduces AOP and replica technology into the design stage of fault-tolerant software, and proposes a non-intrusive fault-tolerant software model with software redundancy replica as fault-tolerant mechanism. The model describes the quality of service and performance indicators of the software system with a declarative fault-tolerant strategy. Through the aspect-oriented UML extension mechanism, the structure of the fault-tolerant aspect and the interaction between the replicas are accurately characterized in the design phase. The interception and proxy mechanisms are implemented at runtime, and the specified fault-tolerant aspect is dynamically woven into the software system, ultimately achieving the goal of eliminating or minimizing the intrusiveness of the fault-tolerant logic in the system to the business logic.

References

[1] Avizienis, A., *et al*. Concepts and taxonomy of dependable and secure computing. *IEEE Trans On Dependable and Secure Computing, 1(1)*, 11-33.

[2] Gamma, E., *et al*. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

[3] Zhao, W. (2007). BFT-WS: A byzantine fault tolerance framework for web service. *Proceedings of the Middleware for Web Service Workshop*.

[4] Jean, C. L., *et al*. (2008). Modelling Interdependencies between the electricity and information infrastructures.

[5] Victor, B., Paolo, D., & Sima Asgari. High Dependability Computing Program, The Unified Model of Dependability, 20742 Technical Report CS-TR 4601-UMIACS-TR-2004-43,*Computer Science Department University of Maryland College Park*, Maryland June 2004.

Shuanghui Yi received a master's degree in computer science and technology in 2003. He had been engaged in distributed computing research for nearly 10 years, software testing for more than 3 years and information system architecture design for more than 5 years at the Beijing Institute of Systems Engineering. At present, he is the deputy director of the Second Research Department of the National Key Laboratory for Complex Systems Simulation, engaged in system design work in the laboratory.

His team was one of the earliest teams engaged in CORBA distributed system research in China. He proposed the CORBA security protocol implementation technology based on micro-kernel, which effectively solved the problem of communication performance loss caused by security encryption.

Currently he is engaged in model-based system modeling and simulation technology research.

Rong Li graduated from the School of Computer Science, Northwest Polytechnic University, He has a master's degree in engineering. He is currently working in the State Key Laboratory of Complex System Simulation, Beijing Institute of Systems Engineering. He is an associate researcher. He has been engaged in basic research and engineering technology development in computer software, knowledge database, system modeling and simulation, artificial intelligence and other professional fields for a long time.