International Journal of Web Technology
Volume: 04 Issue: 01 June 2015 Page No.58-62
ISSN: 2278-2389

Developing Fault Tolerance of Web Applications through Architectural Patterns

Md Umar Khan[1], T.V. Rao[2]

[1]Associate Professor, Computer Science and Engineering Dept.Prakasam Engineering CollegeKandukur, A.P, India
[2]Professor& HOD in CSE Department, PVP Siddartha Institute of Technology Kanuru, Vijayawada, AP, India
Email-v.umar2003@gmail.com, tv_venkat@yahoo.com

Abstract-Modern enterprise web applications need to have reliable functionality. Fault tolerance is one of the reliability factors which can be achieved using modern structuring techniques blended with design patterns. In this paper we improve our architectural pattern XWADF to incorporate fault tolerant strategies in terms of design patterns to make web applications fault tolerant besides achieving scalability and high availability. We proposed two Fault Tolerant (FT) design patterns and incorporated into XWADF so as to make design level decisions that help in developing fault tolerant web applications. We implemented two FT design patterns namely Fault Tolerant Exception Handling Pattern (FTEHP) and Fault Tolerant Security Pattern (FTSP) using Aspect Oriented Programming (AOP) which separates cross cutting concerns from pure business logic thus getting rid of code pollution. The first Design Pattern takes care of faults pertaining to common runtime errors while the second Design Patterns takes care of security related faults. The FTSP is implemented on top of Secure Socket Layer (SSL). We applied the enhanced XWADF to two existing web applications namely Hospital Management System (HMS) and Library Management System (LMS). The empirical results revealed that our Developing Fault Tolerance of Web Applications through Architectural Patterns make applications fault tolerant besides making them scalable with high availability

Keywords- Dynamic web applications, fault tolerance, design pattern, architectural pattern

I. INTRODUCTION

Web applications are widely used by businesses to reach global audience. E-commerce and other commercial applications that leverage business performance need to be given highest priority since their performance leads success. Having said this it is essential to make such applications fault tolerant. Fault is a deviation from the normal behavior of the application. It is also known as a malfunction. Faults might occur for many reasons such as software, hardware, network and human errors [4]. Building a fail-safe web application is challenging task. Apart from measures taken at server level with respect to fault tolerant behavior of web server the application level fault tolerant structure or strategies play an important role to achieve fail-save standard. Fault tolerance is also known as graceful degradation which helps the application to achieve operational continuity. It can recover fast from unknown or unexpected faults. Rather than failing, the fault tolerant web application continues working, most probably, in less than ideal fashion until the fault is overcome. At many levels faults may occur. The faults at application level is focused in this paper and let the application withstand the potential faults. Troger [7] described the different fault tolerance phases such as activation of fault, detection of error, process of error and treatment of fault. These phases are self explanatory and intuitive in nature.

The architectural pattern XWADF that we proposed and improved in [2], [3] to improve performance of web applications with features such as availability and scalability is further extended in this paper in order to make the architecture fault tolerant.

Our contributions in this paper are as follows.

- We identified strategies or methods that can withstand faults in web applications at application level. These strategies are embodied into some design patterns to make web applications fault tolerant.
- We enhanced our architectural pattern XWADF further to incorporate the design patterns as underlying structures in the architectural pattern. Thus our architectural pattern will become useful tool to build highly robust web applications that exhibit desirable quality attributes such as fault tolerance besides having scalability and availability.
- We applied the enhanced architecture to two case study web applications built using Java Enterprise Edition. The fault tolerance feature of these web applications is evaluated with standard metrics

The remaining part of this paper is as follows. Section II reviews literature related to fault tolerance and techniques or strategies that let web applications to be fault tolerant. Section III provides brief overview of our architectural pattern XWADF. Section IV presents the design patterns with underlying fault tolerant strategies which are part of the enhanced XWADF. This is actually the aim of this paper. Section V presents experimental results and evaluation of fault tolerance of the two case study web applications namely HMS and LMS while section VI represents paper conclusion.

II. RELATED WORK

Fault tolerance is highly desirable feature of any software product. Enterprise web applications are no exception. The basic traits of fault tolerance system includes fault detection, fault containment which is essential to prevent its propagation, fault isolation, no single point of failure and the availability of contingency plans in place as reversible models [5]. At hardware level it is possible to overcome faults using replication, redundancy, and diversity. Replication refers to have multiple systems which are identical. Redundancy of identical systems can help in switching to alternative systems in case of failure. Diversity refers to have multiple implementations that can tolerate fault [6]. Troger [7] explored object oriented design patterns for fault tolerant application development. Fault tolerant systems employ patterns such as observers and monitors. Such patters are useful to both tasteful and stateless systems. The fault tolerance and security are non functional requirements of application. According to Troger fault tolerant patterns are architectural , detection, error recovery, error mitigation and fault treatment patterns. Error detection patterns can detect the faults or failures while the error recovery patterns help in operation continuity. In other words, error recovery patterns are responsible to ensure that system continues functioning in spite of some faults and getting rid of the effects of faults if any by creating new state. Error mitigation patterns are responsible to mask errors and compensate the system in other way while the fault treatment patterns prevent recurring of such faults and take corrective measures.

Web applications are prone to runtime faults and need fault tolerant mechanisms at application design level. Harrison et al. [8] show the modern structures that can be employed to build integrated, fault-tolerant applications in the real world. They employed a multi-level exception handling to make the applications fault tolerant. They also applied coordinated atomic actions in order to handle abnormal situations. They modeled travel agency web site for demonstrating the modern structures for fault tolerance.

It is evident that the application has pre-defined structured along with legacy components. The wrappers created and strategically placed can handle abnormal behavior at runtime and ensure that the system exhibits operational continuity [9]. Garcia and Toledo [10] presented architecture for web services to be tolerant. However, in this paper we assume no usage of web services and built an architectural pattern that demonstrates fault tolerance with the help of design patterns. From the experiments of it is understood that fault tolerance has its toll on system performance. Therefore it has to be used sparingly. With fault tolerance and with faults in the system caused more response time. Gawand, Mundada and Swaminathan [11] explored software architectural design for fault tolerance followed by fault tolerance redundancy patterns, reflective state pattern, Tri-Module redundancy pattern and control-monitor safety pattern. The safety measures thy considered in their research include modularity, fault avoidance, fault tolerance, technology transparency and ease of modifications. They identified system failures as of two types namely random and systematic. The former is the failure caused by some degradation in the system while the latter represents a fault due to the flow in the underlying system.

Failure avoidance is achieved through substitution. Failure detection is achieved using time outs, timestamp, condition check and comparison. Failure containment is achieved using redundancy, recovery, masking and barrier. Finally failure is either avoided or handled with the help of this hierarchy of safety tactics. In this paper we are inspired by this hierarchy of safety measures. Richar III and Tu [12] studied fault tolerance practically in Java applications. They observed that fault tolerance is indispensable in both sequential and distributed software systems. They described Java design patterns for achieving fault tolerance. The patterns include Idenpotent for stateful applications, pattern for fault tolerant C/S applications [24] and a pattern for bag-of-tasks applications. Garcia, Beder and Rubira [13] explored fault tolerant object oriented systems. They advocate design decisions pertaining to exception handling for making applications fault tolerant. They explored exception handling in sequential system and also concurrent systems with a banking application as case study. The components in their fault tolerant architectures include exception, handling of exception strategy, concurrent exception handling action.

Collection of design patterns such as exception pattern, handler pattern, strategy pattern and concurrent exception handling pattern are used in for making software application fault tolerant. The names of patterns reflect the purpose of patterns as they are intuitive in nature. In this paper we use a design pattern using AOP (Aspect Oriented Programming) [21] paradigm for complete exception handling thus we separate concerns in the application. Duncan and Pullum [14] employed three level class frameworks as pattern for making applications fault tolerant. By separating their classes into base classes, data type dependent classes and data type specific utilities as three layers they achieved fault tolerance. Again in each type of classes there are two types of components employed namely executive components and building block components.

The executive components make use of fault tolerant techniques which are independent of data types. The building block components bestow utilities pertaining to fault tolerance besides giving application specific code libraries [14]. This approach is not used in this paper as our framework needs other design patterns. A generic approach is possible to have complex fault tolerant applications. Xu, Bandell and Romanovsky [15] focused on building a generic approach that could handle the residual faults in complex software systems. The generic model is actually a multi-layered reference architecture which comprises of two things namely configuration method and an architectural pattern. Their work is based on [16]. The FT component used has both normal activity handling and exception handling component.

The interface exceptions are propagated to the exception handling part of the FT component where it has many handlers for exceptional responses, abort exceptions and failure exceptions. This architecture for making applications FT is known as multi-level reference architecture. Pre-defined classes and runtime libraries are used to implement the FT. Ermagan et al. [17] proposed fault tolerant software oriented architectures which are model driven by nature. They applied the architectures to a trading system. The pattern implemented was named as "rich service" pattern. The FT approach used includes defining failure hypothesis, failure detection and failure mitigation. The interaction pattern along with deadlines proposed in [18] is used while the mitigation strategies explored in [19], [20] are used in the FT approach. In this paper we follow design patterns based approach to implementing fault tolerant architecture. In fact we enhance our XWADF [2] architecture for incorporating FT patterns

III. OVERVIEW OF ARCHITECTURAL PATTERN XWADF

Our architectural pattern XWADF which is based on MVC is presented here for reference. As we are enhancing the architecture using fault tolerant design patterns it is presented here. This proposed architectural pattern consists of different design patterns and is used for improving performance of web applications. As viewed in Figure 1 our architectural solution to high performance web applications presented here has been improved in [3] further for incorporating design patterns pertaining to scalability and availability. In this paper it is further extended to include fault tolerant design patterns that make our architecture robust to faults as well besides exhibiting high scalability and availability

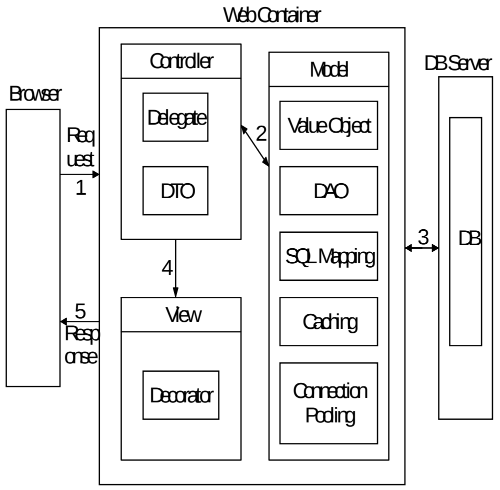

Figure 1 – Illustrates overview of our architectural pattern XWADF

IV. ARCHITECTURAL PATTERN WITH FT DESIGN PATTERNS (ENHANCED XWADF)

Our architectural pattern presented in the preceding section has been improved to incorporate FT design patterns. Interestingly fault tolerance and security come under non-functional requirements. For clean coding strategy we adapt AOP paradigm to implement the FT design patterns. Our FT design patterns cover both runtime abnormal situations and also security attacks. Therefore we implement two FT patterns namely Fault Tolerant Exception Handling Pattern (FTEHP) and Fault Tolerant Security Pattern (FTSP). Before describing the implementation of these FT patterns we would like to introduce the new paradigm (programming) AOP.

A. *Aspect Oriented Programming (AOP)*

This is a new programming approach which based on OOP. It complements OOP by allowing separation of concerns. To state differently, it helps Java developers (AOP is available for other languages as well) to separate business logic from other concerns or aspects. Especially non functional requirements can be implemented using AOP. Thus separation of functional and non-functional requirements is possible. Since FT and security are non functional requirements, they are implemented using AOP. Thus the application Business Logic code is kept clean as it does not pollute with cross cutting concerns like security and FT. Figure 2(a) and Figure 2(b) shows the difference between using AOP with and without using AOP in Java programming As can be seen in Figure 2 (a) the coding of non functional requirements such as logging, security, and profiling are mixed in an object along with Business Logic(BL). This is polluting the code. In Figure 2 (b) the separation of such concerns from pure BL is shown. At runtime the cross cutting aspects are mixed with pure BL without polluting the source code. This weaving is achieved using a GOF pattern [25] known as Decorator.

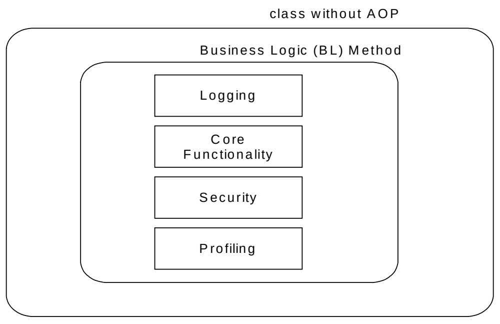

Figure 2 (a)

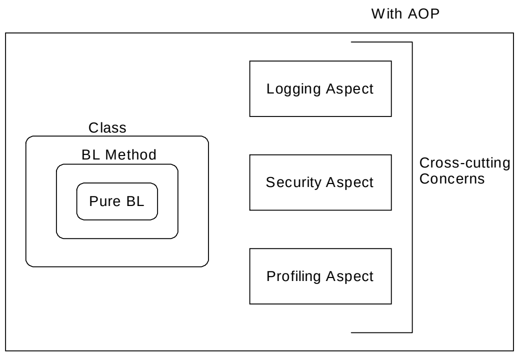

Figure 2(b)

Figure 2 (a), (b) – Illustrates how AOP is capable of separating cross-cutting concerns

B. *Fault Tolerant Exception Handling Pattern (FTEHP)*

This pattern is implemented using Java language in AOP paradigm. The AOP framework used is AspectJ [23] which is a full-fledged AOP framework available. We defined an aspect by name FTExceAspect which performs activities such as fault detection and isolation, fault recovery and fault treatment. It also takes necessary steps to prevent fault propagation. Instead of causing the system fail, this pattern will give priority to operational continuity.

c. *Fault Tolerant Security Pattern (FTSP)*

This pattern is responsible to handle security faults. It takes care of general security mechanisms [22] such as authentication, authorization, confidentiality, and non repudiation. The functionality is implemented using an aspect developed in Aspect J. The name of aspect we implemented is FTSecAspect. The underlying protocol used in this pattern is Secure Sockets Layer (SSL). The SSL is configured by modifying server.xml found in Tomcat’s configurations directory.

V. ENHANCED XWADF WITH PROPOSED DESIGN PATTERNS

Our FT patterns namely Fault Tolerant Exception Handling Pattern (FTEHP) and Fault Tolerant Security Pattern (FTSP) are incorporated into XWADF so as to make the architecture with FT features. The resultant architecture is as shown in the following Figure 3Building fault tolerant features into the architectural pattern of web application is very important so as to enable developers to follow the best practices. Thus the enhanced XWADF architecture when followed yields fault tolerant web applications. The two patterns for FT namely

FTEHP and FTSP are specialized reusable components that can be reused in every web application instead of reinventing the wheel. Thus our architectural approach ensures highly fault tolerant web applications

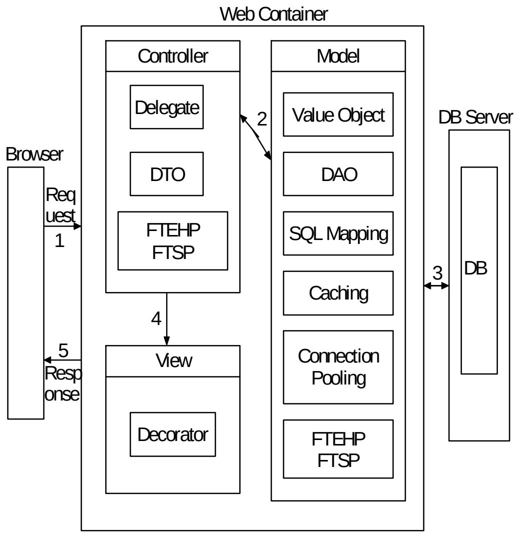

Figure 3 – Illustrates the overview of modified XWADF with FT capabilities

.

VI. EXPERIMENTAL RESULTS AND EVALUATION

A. Experimental Setup

The environment used for experiments include a PC that consists of 4 GB RAM, Core 2 dual processor running on Windows 7 operating system. Java/JEE platform is used for building case study applications. The scalability of applications is tested using two metrics namely latency and throughput. The testing is done using a testing tool known as LoadUIWeb 2. Also we can find number of failures against operational time.

B. Case Study Applications

Two existing web applications [16] are used for experiments. Hospital Management System (HMS) from health care domain and Library Management System (LMS) from education domain were considered to adapt enhanced XWADF. The applications prior to adaption of the proposed architectural pattern and after are subjected to automated testing in terms of CRUD [18] operations. The database used is MYSQL with replication support. The HMS database has one lack records while the LMS has 50000 entities.

C. Evaluation of FT

FT of HMS and LMS were evaluated using traditional FT metric as given below. This metric is taken from [26].

FT = MTBF

Where MTBF stands for Mean Time between Failures. The MTBF is computed as follows.

MTBF = Total operating time / No. of failures encountered

Based on this metric the FT is computed for HMS and LMS applications. For 24 hours, the applications are monitored and the number of failures is considered. Then that operational time is scaled to 1 week and 1 year. The results are as shown in Table 1 and Table 2.

Table 1 Ft Of Hms

FT Measure (Without XWADF)			FT Measure (With XWADF)		
Number of Failures	Operational time	FT Measure (%)	Number of Failures	Operational time	FT Measure (%)
23	1800	78	19	1800	94.7

As can be seen in Table 1, it is evident that the HMS with enhanced XWADF availability is more in the orders of magnitude when compared with the one without XWADF.

Table 2 Ft Of Lms

FT Measure (Without XWADF)			FT Measure (With XWADF)		
Number of Failures	Operational time	FT Measure (%)	Number of Failures	Operational time	FT Measure (%)
25	2000	80	21	2000	95.23

As can be seen in Table 2, it is evident that the LMS with enhanced XWADF availability is more in the orders of magnitude when compared with the one without XWADF.

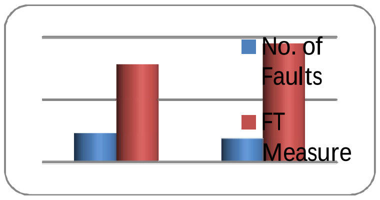

Figure 4 – FT measure of HMS (1800 operational time in seconds)

As shown in Figure 4 our architectural approach to web application development could yield high FT with XWADF. This is evident in case study applications such as HMS.

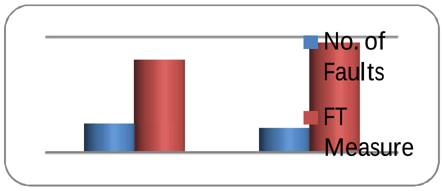

Figure 5 – FT measure of LMS (2000 operational time in seconds)

As shown in Figure 5 our architectural approach to web application development could yield high Ft with XWADF. This is evident in case study applications such as LMS.

VII. CONCLUSIONS AND FUTURE WORK

In this paper we studied fault tolerance its characteristics with respect to software products including web applications. We focused solutions based on architectural or design patterns for making applications fault tolerant. We enhanced our architectural pattern XWADF [2] further to incorporate FT patterns. The FT patterns we implemented are Fault Tolerant Exception Handling Pattern (FTEHP) and Fault Tolerant Security Pattern (FTSP). The former handles any kind of runtime errors while the latter is specialized for handling security related faults. The implementation of these patterns is done using AspectJ as AOP framework. This is done to have clear separation of concerns and avoid pollution. It also promotes code reusability as aspects can handle cross cutting concerns like security and exception handling. These two patterns are incorporated into XWADF [2] so as to make it more robust in terms of fault tolerance besides its existing functionality for making applications highly scalable and available. We tested our architectural pattern and the underlying FT design patterns using two case study web applications namely HMS and LMS. The empirical results are encouraging. As future work we will enhance XWADF further to promote maintainability of web applications by incorporating design patterns into the architectural pattern.

REFERENCES

[1] Umar Khan, Dr. T.V. Rao. (2014). A Survey of Design Patterns for Improving Quality and Performance of Web Application Design. ,p1-10
[2] Umar Khan, Dr. T.V. Rao. (2014). XWADF: Architectural Pattern for Improving Performance of Web Applications. ,p1-9.
[3] Umar Khan, Dr. T.V. Rao. (2014). Web Application’s Reliability Improvement Through Architectural Patterns. ,p1-18.
[4] Paul Krzyzanowski. , Distributed Systems. p1-18.
[5] Wikipedia (2014), "Fault Tolerance". Available online at: http://en.wikipedia.org/wiki/Fault_tolerance
[6] Dr. Kalpakis. (2003). Fault Tolerance in Distributed Systems. CMSC. p1-15.
[7] Dr. Peter Tröger. (2010). Dependable Systems Fault Tolerance Patterns. Fault Tolerance Patterns. p1-35
[8] Neil B. Harrison,Paris Avgeriou,Uwe Zdun. (2010). On the Impact of Fault Tolerance Tactics on Architecture Patterns. ACM. p1-10.
[9] Alexander Romanovsky, Panos Periorellis,Avelino F. Zorzo. Structuring Integrated Web Applications for Fault Tolerance ,p1-9.
[10] Diego Zuquim Guimarães Garcia, Maria Beatriz Felgar de Toledo. (A Fault Tolerant Web Service Architecture. p1-8.
[11] Hemangi Gawand, R.S. Mundada, P.Swaminathan. (2011). Design Patterns to Implement Safety and Fault Tolerance. International Journal of Computer Applications. 18 (2), p1-8.
[12] Gloden G.Richard. (1998). on patterns for practical Fault tolerant software in java. RD. p1-7.
[13] Alessandro F. Garcia,Delano M. Beder,Cecflia M.F. Rubira. (2000). An Exception Handling Software Architecture for Developing Fault-Tolerant Software. IEEE. p1-10.
[14] Ralph Duncan’ and Laura Pullum. (2001). Obj ect-Oriented Executives and Components for Fault Tolerance. IEEE. p1-8.
[15] J. Xu,B. Randell and A. Romanovsky. (2002). A Generic Approach to Structuring and Implementing Complex Fault-Tolerant Software. IEEE. p1-8.
[16] P.A. Lee and T. Anderson. Fault Tolerance: principles andpractice, Second Edition, Springer-Verlag, 1990.
[17] Vina Ermagan, Ingolf Krüger, Massimiliano Menarini. (2008). A Fault Tolerance Approach for Enterprise Applications. IEEE. p1-10.
[18] I. Krüger, R. Grosu, P. Scholz, and M. Broy, “FromMSCs to statecharts”, In Proceedings of the IFIP Wg10.3/Wg10.5 international Workshop on Distributedand Parallel Embedded Systems, F.J. Rammig, Ed.,Kluwer Academic Publishers, Norwell, MA, 1999., pp.61-71.
[19] A. Liu, Q. Li, and M. Xiao. “A Declarative Approach to Enhancing the Reliability of BPEL Processes”, In Proceedings of the IEEE international Conference on Web Services (ICWS'07), IEEE Computer Society,Washington, DC, USA, 2007, pp. 272-279.
[20] T. Mikalsen, S. Tai, and I. Ravellou, “Transactional attitudes: Reliable composition of autonomous Web services”, In Workshop on Dependable MiddlewareBased Systems (WDMS), Washington, DC, USA, 2002.
[21] Ken Wing Kuen Lee. (2002). An Introduction to Aspect-Oriented Programming. Software envelopment of E-Business Applications. p1-16.
[22] Ahmad A. Hassan,Waleed M. Bahgat. (2009). A Framework for Translating a High Level Security Policy into Low Level Security Mechanisms. IEEE. 0 (0), p1-8.
[23] Tao Xie, Jianjun Zhao, Darko Marinov, David Notkin. (2006). Detecting Redundant Unit Tests for AspectJ Programs. IEEE. p1-10.
[24] S. Chinnappen-Rimer and G.P.Hancke. (2003). An XML Model for Use Across Heterogeneous Client-Server Applications. IEEE. p1-5.
[25] Motoshi Saeki. (2000). Behavioral Specification of GOF Design Patterns with LOTOS. IEEE. 0 (0), p1-8.
[26] Cristian Grecu1, Lorena Anghel, Partha P. Pande, André Ivanov, Resve Saleh. (2007). Essential Fault-Tolerance Metrics for NoC Infrastructures. IEEE. p1-6.