The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 15, No. 4, pp.19-24, Aug. 31, 2015. pISSN 2289-0238, eISSN 2289-0246

http://dx.doi.org/10.7236/JIIBC.2015.15.4.19
JIIBC 2015-4-3

보안 전술과 Broker 아키텍처 패턴간의 호환성 분석

Compatibility Analysis Between Security Tactics and Broker Architecture Pattern

김순태*

Suntae Kim*

요 약 보안은 소프트웨어 개발에 있어서 주요한 관심사중 하나이다. 보안 전술(Security tactics)은 아키텍처 수준에서 보안의 문제를 해결하기 위한 재사용 가능한 빌딩 블록으로, 소프트웨어 시스템의 구조를 수립하기 위한 일반적인 해결책인 아키텍처 패턴(Architectural Pattern)과 자주 함께 사용된다. 하지만, 아키텍처 패턴에서의 아키텍처 전술은 패턴이나 전술 참여자들이 구조적/행위적으로 어떻게 함께 설계되어야 하는지에 대한 구체적인 이해 없이 보통 개념적으로만 이해되고 만다. 본 논문에서는 대표적인 아키텍처 패턴중 하나인 Broker패턴에서 이와 함께 사용 가능한 보안 전술을 모델 기반으로 표현하고, 실세계에서 어떻게 이 개념이 적용되었는지에 대한 구체적인 사례를 소개한다.

Abstract Security has been a major concern in software development. Security tactics are reusable building blocks providing a general solution for recurring security concerns at the architectural level. They are often used together with architectural patterns which provide a general solution for architecting software systems. However, use of security tactics in architectural patterns has been understood only conceptually without concrete understanding of how their involved elements should be structurally and behaviorally co-designed. In this paper, we present model-driven analyses of security tactics in the context of Broker architectural patterns and provide evidences of the analyses in real world case studies.

Key Words : Broker Architecture Pattern, Security Tactics, Compatibility Analysis

Ⅰ. 서 론

보안 아키텍처 전술(Security Architectural Tactic)은 소프트웨어의 보안 품질을 다루기 위한 아키텍처 수준의 재사용 가능한 빌딩 블록으로[1], 대표적인 예로는 Detecting Attacks, Recovering from Attacks 등이 있다. 보안 전술은 Broker, Layer 패턴과 같은 소프트웨어의 구조를 수립하기 위한 기반을 제공하는 소프트웨어 아키텍처 패턴과 함께 사용된다[2]. 보안 전술을 아키텍처 패턴에 적용하기 위해서는 불필요한 복잡성이나, 보안의 취약성이 발생하지 않도록 신중한 설계가 필요하다. 하지만, 많은 경우에 개발자들은 자신의 과거의 경험이나 직관에 의존하여 전술과 패턴을 통합하여 보안의 취약성을 만들어 낸다. 이는 전술과 패턴 모두 구체적인 명세 없이 개념적으로 정의되었으며, 통합을 위한 가이드라인이나, 기존에 통합되어 적용된 사례의 제시의 부족으로 인하여 이러한 문제가 발생한다[3].

본 논문에서는 보안 전술과 대표적인 분산 아키텍처

*정회원, 전북대학교 소프트웨어공학과
접수일자 2015년 7월 28일, 수정완료 2015년 8월 7일
게재확정일자 2015년 8월 7일

Received: 28 July, 2015 / Revised: 7 August, 2015 /
Accepted: 7 August, 2015
*Corresponding Author: stkim@jbnu.ac.kr
Dept of Software Engineering, Chunbuk National University, korea

패턴인 Broker 패턴의 구조와 행위에 대하여 분석 및 명세하고, 어떻게 보안 전술이 보안 문제를 해결하기 위하여 Broker 패턴에 장착될 수 있는지에 대하여 소개 한다. 그리고 실제 Broker 패턴이 적용된 사례인 HornetQ[4]에서 보안 전술이 실제로 어떻게 적용되었는지에 대한 증거(Evidence)를 제시한다.

본 논문은 다음과 같이 구성하였다. 2장에서는 보안 아키텍처 전술과 그 관계에 대하여 소개하고, 3장에서는 보안 전술과 연동한 Broker아키텍처 패턴에 대하여 각 전술과 패턴과의 연관성을 소개한다. 4장에서는 보안 전술을 장착한 Broker 아키텍처 패턴의 적용 사례인 HornetQ를 보여주며, 마지막으로 5장에서는 본 논문을 마무리한다.

II. 보안 아키텍처 전술

보안은 허용된 사용자에게 시스템의 서비스를 제공하면서, 악의적인 사용자로부터 시스템을 보호하는 품질중 하나이다. 보안 전술은 이러한 시스템의 보안 품질을 높이기 위한 아키텍처 수준의 접근 방법으로, 이는 그림 1과 같이 Resisting Attacks, Detecting Attacks, 그리고 Recovering From Attacks으로 구분이 가능하다.

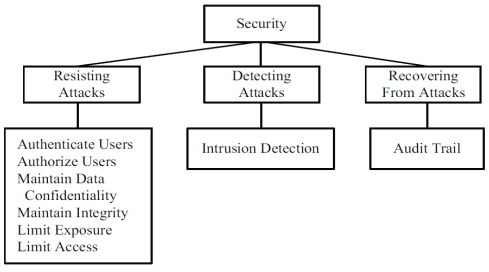


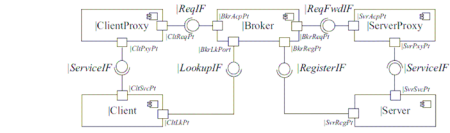
그림 1. 보안 아키텍처 전술
Fig. 1. Security Architectural Tactics

Resisting Attacks은 사전에 악의적인 시스템의 공격을 차단하기 위한 방법으로 이중 Authenticate User는 ID/Password와 같이 사용자의 Credential을 확인하기 위한 방법이고, Authorize User와 같이 허용된 사용자만이 정해진 리소스에 대한 접근을 허용하는 것이다. Maintain Data Confidentiality 전술은 허용되지 않은 사용자에게 정보의 노출을 막기 위한 방법이며, Maintain Integrity는 비 인가된 정보의 수정을 차단하기 위한 방법이다. Limit Exposure는 시스템의 사용량을 제안하여 과도한 시스템 사용을 줄여 리소스 손실을 낮추기 위한 방법이며, Limit Access는 보안 Safeguard를 활용하여 시스템 접근을 통제하기 위한 방법이다.

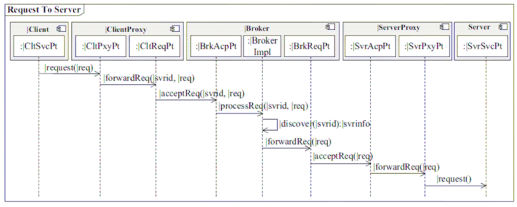
Registing Attacks는 사전에 공격을 차단하기 위한 방법이지만, Detecting Attacks 기법은 시스템이 공격을 받은 후에 이를 확인하는 방법으로 Intrusion Detection 방법이 있다. 이는 정상적인 사용 패턴과 침입자의 사용 사례를 로그 분석과 사용 패턴 비교를 통하여 알아내는 방법이다. 또한 Attack후에 시스템을 복구하는 방법인 Recovering From Attack방법은 Audit Trail 전술이 있으며, 이는 사용 로그를 저장하고 이를 Replay하는 방법이 있다. 각 전술의 구조/행위적 의미에 대한 분석은 [5]를 기반으로 사용한다.

III. 보안 전술과 연동한 Broker 아키텍처 패턴

보안 아키텍처 전술을 아키텍처 패턴과 결합하여 자주 사용된다. 본 장에서는 분산 아키텍처 패턴에서 가장 대표적인 Broker 아키텍처 패턴과 보안 전술과의 호환성을 분석하고 각 보안 전술과 Broker 패턴과의 연동 방안에 대하여 소개한다. Broker패턴은 분산 컴포넌트를 검색하고, 이 컴포넌트의 메소드를 호출하고 결과를 얻기 위한 과정을 지원하는 패턴으로 그림 2와 같은 구조와 행위를 갖는다.



(a) Structure



(b) Behaviors

그림 2. Broker 아키텍처 패턴
Fig. 2. The Broker Architecture Pattern

Broker패턴은 서비스를 요청하는 Client, 서비스를 제공하는 Server, Client의 요청을 Broker에 전달하기 위한 Client Proxy, Broker로부터 Server의 호출을 지원하기 위한 Server Proxy, 그리고 Server목록과 Client로부터 Server의 Lookup을 제공하는 Broker 컴포넌트로 구성된다. 각 컴포넌트들은 각각 Provided Interface와 Required Interface가 존재하며, 이 Interface를 구현하는 Port가 존재한다 (UML2.0 참조[6]). 또한 각 컴포넌트의 이름 앞에는 'I'가 존재하는데, 이는 역할(Role)을 의미한다. 그림 2(b)는 Client 컴포넌트에서 Server 컴포넌트를 호출하는 일련의 과정을 보여준다.

실세계에서 Broker 패턴은 다양한 미들웨어에서 적용되었다. 다양한 JMS(Java Messaging Service)와 같은 MOM (Message Oriented Middleware)에서 적용된 바 있으며, Java RMI(Remote Method Invocation)에서도 Broker패턴을 기반으로 하고 있고, CORBA(Common Object Request Broker Architecture)에서도 Broker 패턴을 사용하고 있다. 최근에는 Web Service에서도 Broker 패턴을 기반으로 구축되었다.

1. Authentication Tactic연동 Broker 패턴

그림 3은 Authenticate Users 보안 전술이 연동된 Broker 패턴의 구조를 보여준다. 그림의 붉은색 점선이 보안 전술이 연동된 부분이다. 그림에서, Client Proxy-Broker, Broker-Server Proxy사이에서 Authenticate Users전술이 사용될 수 있으며, 여기에서는 두 Proxy가 Broker에 접근하기 위한 적절한 Library 인증 키를 갖고 있는지 확인하기 위한 방법으로 사용될 수 있다. 이때, Broker안의 Authenticator는 Library Key를 내부에서 유지하여, 각 Proxy의 접근이 타당한지 확인한다.

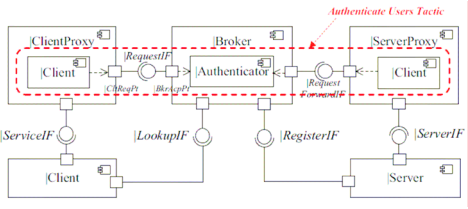


그림 3. Authenticate Users 전술 장착 Broker 패턴
Fig. 3. Broker Pattern Equipped with Authenticate Users Tactic

비록 JMS 명세서에는 Proxy에 대한 타당성 검증에 대한 제약을 두고 있지 않으나, ActiveMQ, IBM MQ, HornetQ에서는 모두 Library Key를 유지하고, 이에 대한 타당성에 대하여 실행시 검증한다.

2. Authorization Tactic 연동 Broker 패턴

Authorize Users 전술은 Broker에의해 관리되는 Server가 제공하는 다양한 서비스에 대한 접근 제어를 하기 위하여 사용된다. 그림 4는 Authorize User 전술이 장착된 Broker 패턴의 구조를 보여준다. Broker내의 Reference Monitor는 Reference와 Reference가 접근 가능한 Resource에 대한 연동 정보를 갖고 있으며, Client Proxy는 자체적 보유하거나 Client로 부여받은 Reference를 Broker로 전달하여 해당 Resource로의 접근을 시도한다.

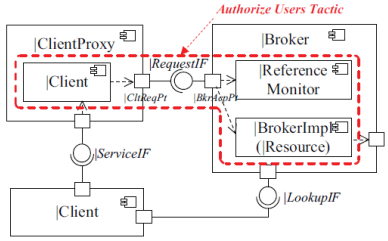


그림 4. Authorize Users 전술 장착 Broker Pattern
Fig. 4. Broker Pattern Equipped with Authorize Users Tactic

그림 5는 Authorize Users 전술 연동으로 인한 Request To Server의 행위의 변경 사례를 보여준다. Broker의 포트인 BrkAcpPt에서는 특정 Resource를 접근하기 전에 항상 ReferenceMonitor를 접근하여 허용 가능한 Permission을 갖는지 확인하고, 허용 가능할 때만 Resource에 접근한다.

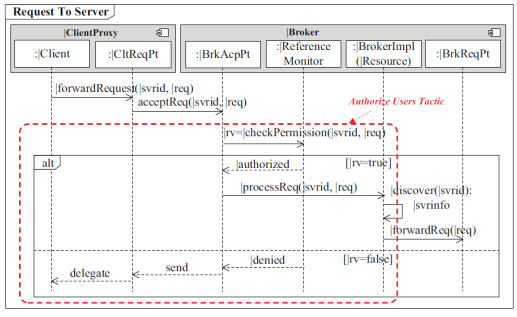


그림 5. Authorize Users 전술 장착 Broker 패턴의 행위 모델
Fig. 5. Behavioral Model for the Broker Pattern equipped with Authorize Users tactic

3. Confidentiality Tactic 연동 Broker 패턴

Broker 패턴에서 Maintain Data Confidentiality 전술은 Broker와 Proxy사이의 보안성 있는 데이터 전송을 위하여 사용될 수 있다. 이때, 양쪽 종단에 있는 Port에서는 데이터를 전송하기 전 Encrypt하고 데이터를 수신 후에 Decrypt하여 데이터의 노출(Disclosure)를 차단한다. 그림 6은 Maintain Data Confidentiality 전술을 장착한 Broker패턴의 구조와 행위를 보여준다.

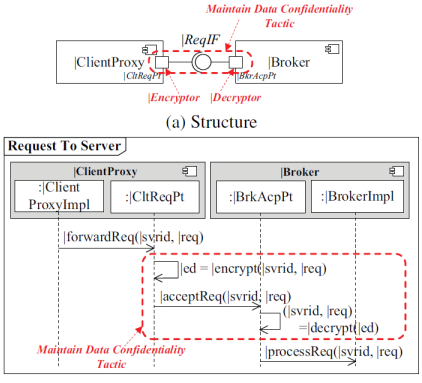


그림 6. Maintain Data Confidentiality 전술 장착 Broker 패턴의 구조 및 행위 모델

Fig. 6. Structural and Behavioral Model of the Broker Pattern equipped with Maintain Data Confidentiality tactic

4. Audit Trail Tactic 연동 Broker 패턴

Audit Trail 전술은 시스템의 보안 정책에 문제가 되는지 확인하기 위하여 시스템 사용 정보를 기록/감사하는 전술이다. 이 전술에서의 핵심 컴포넌트인 LogAuditor는 Broker 패턴에서 Broker와 Proxy 컴포넌트내에 위치할 수 있다. 그림 7(a)에서는 Broker 패턴에서의 Audit Trail 전술의 구조적인 위치를 보여주고, 그림 7(b)에서는 Client Proxy에서 Broker로의 메시지 전송 및 Broker에서의 메시지 수신 직후 request정보를 저장하는 시스템의 행위를 보여준다.

5. 기타 보안 전술과 Broker 패턴과의 관계

위에서 소개한 4가지 보안 전술 이외의 보안 전술(예, Limit Exposure, Intrusion Detection)은 Broker 패턴과 연동하여 사용된 사례를 찾기가 어렵다. 이는 선언적으로 사용할 수 없다고 할 수는 없으나, Broker 패턴을 주로 적용하는 실제 사례에서는 발견하지 못하였다. 예를 들면 Limit Exposure 전술을 Broker 패턴에 적용한다면, Client의 Broker에 대한 접근수를 줄이게 되는데, 이는 Broker가 가진 가능한 많은 Client에게 서비스 하는 시스템의 본래 취지에 반하게 된다. 또한 Intrusion Detection이나 Limit Access 전술은 Broker 패턴에 적용가능하나, 이들은 Web기반 시스템에서 보다 흔하게 발견된다.

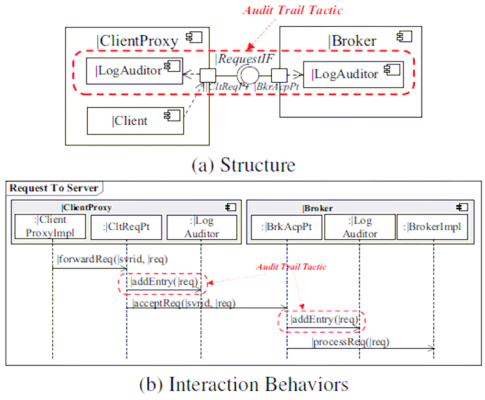


그림 7. Audit Trail 전술 장착 Broker 패턴의 구조 및 행위 모델

Fig. 7. Structural and Behavioral Model of the Broker Pattern equipped with Audit Trail tactic

Ⅳ. HornetQ: 보안 전술 장착 Broker 패턴 적용 사례 연구

이번 장에서는 전장에서 소개한 Broker패턴과 연동 가능한 보안 아키텍처 전술이 실제 시스템에서 적용된 사례를 소개한다. 이를 통하여 개념적으로 소개된 아키텍처 전술이 어떻게 실체화 되는지를 이해하고, 패턴과 연동된 아키텍처 전술에 대하여 사례를 통하여 증명하도록 한다. 사례연구를 위하여 우리는 JBoss HornetQ 2.4.0[4]을 사용하였다. HornetQ는 JMS명세를 구현한 오픈 소스 MOM 시스템으로, 이는 종단에 존재하는 Client 간의 신뢰성 있는 동기/비동기 메시지 전송을 지원하는 미들웨어 시스템이다. 신뢰성 있는 비동기 메시지 전송의 예는 핸드폰의 문자메시지 서비스로, 상대방 핸드폰이 켜져있는 것과 관계없이, 반드시 메시지는 전송된다. 이러한 서비스를 엔터프라이즈 컴포넌트간의 메시지 전송의 목적으로 만든 것이 JMS이며, 이중에 하나가 JBoss의 HornetQ 메시지 미들웨어이다.

그림 7은 HornetQ의 아키텍처를 보여준다. 각 컴포넌트의 Broker패턴에서의 역할은 << >>내에 표시하였다. HornetQ는 두 클라이언트 역할인 JMSMessageSender, JMSMessageReceiver가 존재하고, 둘 다 HQJMSLib를 공통적으로 사용한다. HQJMSLib는 Broker에서 ClientProxy와 ServerProxy의 역할을 수행하며, 여기에서 HQJMSServer와 통신을 담당한다. JMSServer는 Broker 컴포넌트의 역할을 수행하고, 여기에서 신뢰성 있는 메시지 전송을 위한 메시지 저장소와 분배기의 작업을 수행한다. HQJMSServer내에는 보안을 담당하는 HornetQSecurityManager가 존재하고, 로그정보를 기록하는 HQLogAuditor컴포넌트가 존재한다.

그림 7에는 HornetQ 아키텍처에서 어느 부분이 보안 아키텍처 전술이 사용되었는지를 빨간색 점선으로 하이라이트 하였다. HornetQ의 아키텍처에서는 JMSMessageSender와 JMSMessageReceiver 컴포넌트에 Authenticate Users 보안 전술이 사용되며, HQJMSLib와 HQJMSServer사이에는 Library의 타당성을 확인하기 위하여 Library에 Magic Number를 삽입하여 인증을 수행한다.

또한, 모든 네트워크를 통한 전송에서 메시지 내용의 외부노출을 차단하기 위하여 Maintain Data Confidentiality 보안 전술이 사용된다. HornetQ는 Data Confidentiality를 위하여 SSL/TLS 기반 통신을 지원하는 비동기 전송 지원 미들웨어인 Netty[7]를 사용한다. HQJMSLib에서 메시지를 보내기 전, 그리고 HQJMSServer에서 메시지 수신 직후에 HQLogAuditor를 통하여 모든 메시지를 저장한다.

마지막으로 HQJMSServer가 시작할 때 이 서버는 사용자의 id와 password, 그리고 각각마다 access할수 있는 서비스 수준을 로딩하여 HornetQSecurityManager에 저장한다(그림 중간 하단의 Database 참조). 이는 JMSMessageSender의 인증과 접근 수준을 제안하기 위하여 사용된다.

그림 8은 JMSMessageSender에서 JMS의 API(Application Programming Interface)인 ConnectionFactory를 사용하여 사용자 계정(id/password)을 통한 인증의 샘플코드를 보여준다. 여기에서는 JNDI(Java Naming and Diectory Interface)를 통하여 저장된 ConnectionFactory를 얻어오고(Line 6), 여기에서 Connection객체를 얻어올 때 id/password를 전달한다 (Line 7). 이 코드는 그림 7의 JMSMessageSender의 Authenticate User 전술의 적용예를 코드 수준으로 보여준다.

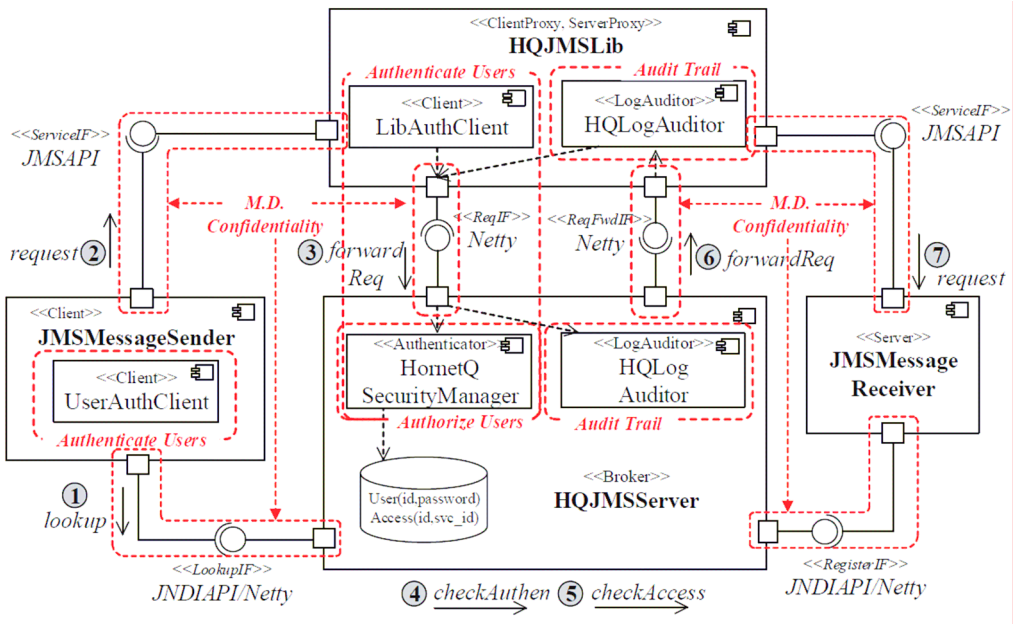


그림 7. HornetQ JMS 서버 아키텍처
Fig. 7. HornetQ JMS Server Architecture

```
class JMSApp {
 Topic topic = null;
 Connection conn = null;

 public void authenticate(){
   ConnectionFactory cf = (ConnectionFactory)initialContext.lookup("/
    ConnectionFactory");
   conn = cf.createConnection("andrew", "hornetq1");
 }

 public void sendAndWaitResponse(long waitingTime){
   topic = (Topic)initialContext.lookup("/myTopic");
   Session session = conn.createSession(false, Session.AUTO_ACKNOWLEDGE);
   MessageProducer producer = session.createProducer(topic);
   TextMessage msg = session.createTextMessage("My Messages");
   producer.send(msg);
   MessageConsumer consumer = session.createConsumer(topic);
   TextMessage receivedMsg = (TextMessage)consumer.receive(waitingTime);
 }
}
```

그림 8. JMS ConnectionFactory를 사용한 사용자 인증 코드
Fig. 8. A Code Snippet for User Authentication Using JMS ConnectionFactory

V. 결 론

본 논문에서는 Broker패턴과 보안 아키텍처 전술 간의 호환성 분석을 소개하였다. 또한, 이를 입증하기 위하여 실세계에 존재하는 사례 아키텍처 수준과 코드 수준으로 소개하여 패턴과 전술간의 호환성에 대한 증거로 제시하였다. 이는 패턴과 보안 전술을 혼용해야하는 개발자에게 구체적이고 실질적인 가이드라인으로 활용이 가능하다. 향후 연구로는 다양한 아키텍처 패턴과 보안 전술간의 관계와 호환성에 대한 연구 및 공통 품질을 위한 프로덕트라인과의 통합 연구[8][9]를 진행하고자 한다.

References

[1] L. Bass, P. Clements, and R. Kazman,, Software Architecture in Practice, 3rd Edition. Addison Wesley.2012

[2] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, Pattern-Oriented Software Architecture: A System of Pattern. Jonh Wiley, 1996.

[3] F. Buschmann, L. Bass, and R. Nord, Understanding Architectural Patterns in Terms of Tactics and Models 2007. http://www.sei.cmu.edu/library/abstracts/news-at-sei/architect200708.cfm.

[4] JBoss HornetQ 2.4.0 http://hornetq.jboss.org/.

[5] S. Kim, D. Kim, L. Lu, and S. Park, “Quality-Driven Architecture Development Using Architectural Tactics”. Journal of Systems and Software, pp1211 - 1231. 82(8), 2009.

[6] UML2.0, http://www.omg.org/spec/UML/2.0/

[7] Netty Project, http://netty.io.

[8] K. Lee, “Design Technology of Product Line Architecture for Software Globalization”, JIIBC, Vol. 13, No. 2, pp83-92, 2013

[9] M. Hwang, K. Lee, S. Yoon, “Software Development Methodology for SaaS Cloud sService”, JIIB, Vol. 14, No. 1, pp61-67, 2014

저자 소개

김 순 태(정회원)

- 2014년 ~ 현재 : 전북대학교 소프트웨어공학과 조교수
- 2007년 2월 ~ 2010년 8월 : 서강대학교 컴퓨터공학과 (석사 및 박사)
- 2003년 2월 : 중앙대학교 컴퓨터공학과 (학사)
- 2002년 11월 ~ 2004년 8월 : 소프트웨어크래프트 컨설팅 선임컨설턴트