이벤트 파싱 엔진의 구조 설계와 성능 분석

윤태웅, 민덕기

건국대학교 컴퓨터·정보통신공학과

{taewoong, dkmin}@konkuk.ac.kr

Architecture Modeling and Performance Analysis of Event Rule Engine

Taewoong Yun, Dugki Min

Department of Computer Science and Engineering, Konkuk University

Abstract

In operating distributed systems, proactive management is one of the major concerns for better quality of service and future capacity planning. In order to handle this management problem effectively, it is necessary to analyze performances of the distributed system and events generated by components in the system. This paper provides a rule-based event parsing engine for proactive management. Our event parsing engine uses object hooking-based and event-token approaches. The object hooking-based approach prepares new conditions and actions in Java classes and allows dynamically exchange them as hook objects in run time. The event-token approach allows the event parsing engine consider a proper sequence and relationship among events as an event token to trigger an action. We analyze the performance of our event parsing engine with two different implementations of rule structure; one is table-based and the other is tree-based.

Key Words : Event Parsing Engine, Event Correlation, Rule-based Parsing Engine, Performance Analysis

* 건국대학교 컴퓨터.정보통신공학과

1. 서론

수많은 서버들로 이루어진 분산 시스템을 관리하기 위해서는 먼저 현재 시스템에 대한 정확한 분석이 필요하다. 시스템에 대한 분석을 위해서는 분산 시스템을 이루는 각각의 서버들 혹은 서버내의 각각의 컴포넌트들이 발생시키는 이벤트를 모니터링 하는 시스템이 필요하다[1]. 시스템에서 발생한 이벤트를 통해서 현재 시스템의 성능 분석은 물론 시스템의 오류 발생 시 오류발생의 원인 분석을 할 수 있다. 또한 발생하는 이벤트를 통해서 오류 발생을 예측하여 시스템의 에러를 방지하는 방법인 Proactive Management[7]가 가능하게 된다. 이러한 이유로 다양한 시스템의 컴포넌트들이 발생 시키는 이벤트 파싱을 위한 유연한 이벤트 파싱 엔진이 필요하다.

본 연구에서는 유연한 이벤트 파싱 엔진을 위한 접근방법으로 기존의 룰 기반의 접근 방법을 변형한 새로운 접근방법을 제시한다. 룰의 조건과 액션을 후크 객체화한 후크 객체 기반 접근방법과 룰에 해당하는 이벤트를 찾아낼 때 보다 효과적인 검색을 지원하는 이벤트 토큰 기반 접근방법을 사용한다. 또한 이벤트 파싱 엔진의 구조를 위해서는 이벤트를 찾아내는 룰의 저장방식을 테이블과 트리 방식으로 비교 분석하여 효과적인 이벤트 파싱 구조를 제시한다

본 논문은 2장에서 이벤트 시스템의 3계층의 구조와 역할에 대해서 설명하고, 3장에서는 기존의 이벤트 파싱 접근방법의 장단점을 비교 분석한다. 4장에서는 논문에서 제시하는 접근방법에 대해서 설명하고 5장에서는 4장에서 제시하는 접근방법을 사용한 이벤트 파싱 엔진의 구조에 대해서 설명한다. 6장에서는 룰의 저장방식에 따른 이벤트 파싱 엔진의 성능 분석결과를 보이고 마지막 7장에서는 결론과 향후 추가되어야 할 사항으로 마무리한다.

2. 이벤트 시스템 구조

이벤트 기반의 시스템이란 시스템의 각 컴포넌트에서 특정한 상황에 발생하는 이벤트가 미리 정의되어 있고 컴포넌트와 컴포넌트들이 이벤트를 주고받는 시스템을 말한다. 즉 서로 연관되어 있는 컴포넌트들은 상대편의 컴포넌트로부터 받은 이벤트와 자신의 상태를 비교해서 작동하는 것이다.

이벤트 시스템의 구조는 그림1 과 같이 세 개의 계층으로 나뉘어 진다. Message Communication은 네트워크를 통해서 메시지를 보내는 계층이고 Notification Server, Channel Server는 각 노드에서 발생하는 이벤트를 처리하고 다른 노드로 이벤트를 전달하는 계층이다. Notification Server를 통해서 서로 떨어져 있는 컴포넌트들의 Direct 통신이 가능하며 서로 직접적으로 연결될 수 없는 컴포넌트들은 Channel Server를 통해서 간접적으로 Indirect통신을 하게 된다. 이벤트 파싱을 위한 룰 기반의 엔진은 Rule Engine, Correlation 계층에 존재한다. Rule Engine, Correlation 계층은 하나의 노드 혹은 하나의 컴포넌트, 또는 전체 시스템에서의 이벤트 발생원인 및 관계 분석과 분석된 결과를 다시 시스템에 적용시키는 역할을 담당한다.

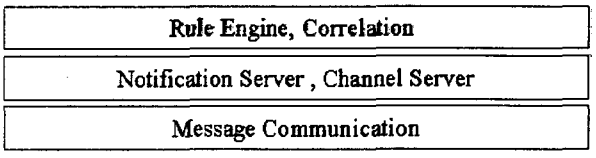

그림 1 이벤트 시스템의 구조

3. 관련연구

3.1. Casual-based 모델 접근방법

Casual-based 모델 시스템[3]은 시스템에서 일어 날 수 있는 다양한 이벤트와 원인을 저장하고 있다가 어떤 이벤트가 발생 시 발생한 이벤트의 원인을 찾아내는 모델이다. 이 모델

은 세 개의 부분으로 되어 있으며 Dynamic Fact Repository, Event Knowledge Model, 그리고 Correlation으로 구성된다. Dynamic Fact Repository는 네트워크나 시스템의 정보를 저장하고 있고, Event Knowledge Model은 시스템에서 일어날 수 있는 다양한 이벤트와 원인을 저장하고 있다. 이러한 이벤트와 원인의 인과 관계는 트리, 그래프, 룰, 유한 상태머신 등으로 표현 할 수 있다. 마지막으로 Correlation은 Knowledge Model을 가지고 시스템 모니터로부터 들어온 이벤트를 통해서 시스템으로부터 발생한 이벤트의 원인과 발생 장소를 분석해 낸다. 다른 접근방법 중에 일반적이며, 추상적인 접근방법이다. 실제 구현으로 위해서는 3.2,절 및 3.3절에서와 같은 구체적인 접근방법을 사용해야 한다.

3.2. 코드 기반 접근방법

코드 기반 접근방법[3]은 CASUAL MODEL 접근방법의 변형으로 발생할 수 있는 이벤트를 코드(Code)로 미리 정의 한 후 발생하는 이벤트가 미리 정의된 코드표에 있으면 적절한 액션을 취하는 접근방법이다.

장점으로는 코드 값만 비교하므로 알고리즘이 단순하고 실행 속도 또한 빠르다. 단점으로는 다양한 환경에서 일어나는 모든 이벤트를 코드 화 할 수 없기 때문에 사용하기에는 비현실적이다 는 것이다. 또한 다양한 이벤트 파싱을 위한 접근방법으로는 부적합한 면이 있다.

3.3. 룰 기반 접근방법

룰 기반의 접근방법[2,3]은 이벤트 파싱의 방법으로 룰을 정의하고 정의된 룰의 조건에 해당하는 이벤트를 검색해서 룰을 적용하는 방법이다. 이 모델은 세 개의 컴포넌트로 이루어진다. 각 각 User-Interface, Inference Engine, Knowledge(or Rule) Base 이며 도메인 지식을 룰로 표현 가능한 전문가 시스템의 한 타입이다. 룰은 "IF condition THEN action"의 형태를 가진다[8]. 조건에 맞는 이벤트가 발생했을 때 추론 엔진이 조건을 판별하고 액션을 취하는 방식이다. 코드 기반의 방식처럼 이벤트를 하나의 코드로 표현하는 것이 아니라 발생할 수 있는 이벤트의 집합으로 정의한다.

장점으로는 코드 기반 접근방법 보다는 다양한 형태의 이벤트와 룰을 정의할 수 있다는 것이다. 단점으로는 룰에 적용되는지를 판별하는 과정이 비교적 복잡하고, 이벤트의 흐름과 조건을 다 확인해야 하기 때문에 코드 기반 접근방법보다 느리다. 또한 조건과 액션은 스크립트 같은 언어로 정의되어 있으며, 따로 조건과 액션을 정의하는 언어를 알아야 하고 그 기능이 다양하지 못해서 복잡한 조건이나 다양한 액션이 불가능하다.

4. 이벤트 파싱 엔진 접근방법

본 장에서는 이벤트 파싱 엔진을 위한 접근방법과 구현에 있어서의 특징인 후크 객체 기반 접근방법, 이벤트 토큰 기반 접근방법에 대하여 살펴보겠다.

4.1. 후크 객체 기반

후크 객체 기반 접근방법은 룰 기반 접근방법의 변형하여 룰의 조건과 액션을 정의를 제한된 문법의 스크립트 언어로 작성하는 대신에 후크(Hook) 객체[5]를 정의해서 사용한다. 구현에 있어서 후크 객체를 만들어 내기 위해서 객체 지향 언어인 자바 언어를 선택하였다.

후크 객체 기반의 장점으로는 조건과 액션을 정의할 스크립트 언어를 따로 정의 할 필요가 없고, 조건과 액션코드가 인터프러터 방식이 아닌 자바 바이트 코드로 미리 컴파일되어 있는 컴파일러 방식으로 작동되기 때문에 속도가 빠르다는 장점이 있다. 더욱이 자바

언어 내부에 정의되어있는 라이브러리를 사용할 수 있기 때문에 보다 복잡한 조건문과 액션을 처리 할 수 있게 된다. 즉, 다양한 문자열 처리, 숫자 처리, 논리 처리가 가능해 지는 것이다. 또 하나의 장점으로는 실행 가능한 액션코드로 기존의 시스템(자바)에 쉽게 통합할 수 있기 때문에 이벤트를 적용할 시스템에 보다 많고, 강력한 액션을 취할 수 있게 된다는 장점이 있다.

4.2. 이벤트 토큰 기반

이벤트 토큰 기반 방식이란 이벤트 흐름에 맞는 룰을 검색하는 과정에서 룰 안에 정의된 이벤트의 리스트와 조건을 전부를 검색하는 것이 아니라 룰에 적용 될 수 있는 이벤트 리스트만 먼저 검색하는 방식이다. 이벤트를 구별하는 이벤트 타입은 코바의 이벤트 모델의 StructuredEvent[4]의 이벤트 타입에 이벤트 도메인 이름을 포함해서 이벤트 타입으로 정했다. 또한 이러한 이벤트 타입의 연속된 리스트를 이벤트 토큰으로 칭한다. 이벤트 토큰을 검색함으로써 룰에 적용되어야 하는 이벤트의 타입 리스트만 먼저 매칭하는 빠른 검색 방법을 사용하는 것이다. 이 방법은 일반적인 컴파일러가 문자열에서 의미 있는 토큰(Token)을 찾아내는 방법[6]과 동일하게 처리할 수 있다. 또한 정규식 에 있는 연산자 기호(*, ., [])등을 표현 할 수 있어서 더욱 실제 환경에서 발생 할 수 있는 이벤트를 비슷하게 묘사 할 수 있다는 장점이 있다. 본 연구에서는 이벤트 토큰의 다양성 보다는 파싱 엔진 구조의 비교 분석에 초점을 두었기 때문에 정규식 표현을 제외하였다. 그러나 향후 연구에 있어서 정규식 표현을 포함함으로써 보다 다양한 이벤트 표현이 가능하게 될 수 있다.

5. 이벤트 파싱 엔진의 구조

이벤트 파싱 엔진의 구조는 크게 세 개의 패키지로 구성된다. Information Package는 이벤트 파싱 엔진의 룰 정보, 엔진의 정보를 관리하고, Engine Package는 룰 엔진의 핵심 골격을 담당하고, 엔진을 시작, 중지, 하는 역할을 한다. 마지막으로 Parser Package는 Information Package에서 정의된 룰을 가지고 파싱 테이블을 구성한 뒤 실시간으로 발생하는 이벤트를 가지고 적용할 수 있는 룰을 찾는 작업을 한다.

5.1. Information Package

Infomation 패키지는 그림 2와 같이 이루어져 있다. RuleInfo는 하나 이상의 룰을 정의한다. 룰 정의에는 룰 이름, 룰이 두 개 이상 발견 시 적용되는 순서를 정하는 우선순위, "도메인 이름:이벤트 타입이름"을 나태는 이벤트 토큰 이름, 자바 문장으로 이루어진 조건 코드, 자바 문장으로 이루어진 액션 코드로 구성되어 있다. 또한 이벤트 파싱에 필요한 룰 정의를 XML파일에 저장해서 이벤트 파싱 엔진을 사용하려는 곳에서 룰 정보를 삽입, 삭제, 수정함으로써 발생하는 이벤트에 따른 액션이 달라지게 하였다.

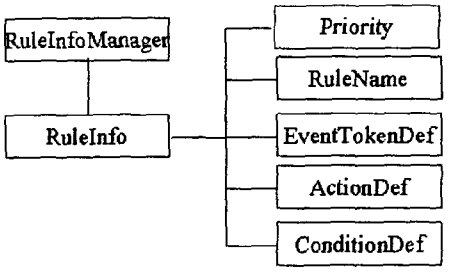

그림 2 Information package 구조

5.2. Engine Package

Engine패키지는 룰 엔진의 핵심 부분이며 그림 3과 같이 구성되어 있다. 룰 엔진은 세 개의 컴포넌트로 이루어져 있다. EventBuffer-Manager는 실시간으로 들어온 이벤트를 관리

하고 특정 시간이 지날 경우 오래된 이벤트는 삭제하는 일을 담당한다. RuleInfoManager는 룰 관련 정보를 관리하며, JavaCodeBuilder는 엔진 최초 가동 시 조건과 액션을 자바 코드로 정의된 부분을 실행가능한 인터페이스 상속 객체로 변경한다. 엔진 실행 시 조건을 비교하거나 액션을 실행 할 때 JavaCodeBulider를 통해서 생성된 비교가능 하거나 실행 가능한 객체를 사용하게 된다.

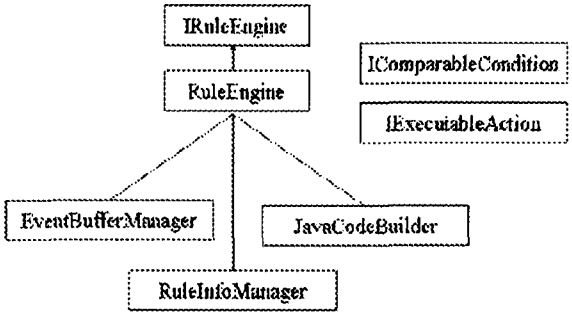

그림 3 룰 파싱 엔진 구조

5.2.1. IComparableCondition Interface

IComparableCondition은 룰의 조건 후크 객체의 조건문 문장을 비교 가능한 객체로 변환 시 상속받는 인터페이스이다. 파라미터 인 Events는 룰에 정의 된 EventTokenList를 의미한다. 인터페이스 안에 정의된 Compare함수에서 파라미터인 Events를 가지고 이벤트의 실제 데이터를 사용해서 조건을 만들어 낸다. 즉, 발생된 이벤트의 데이터를 확인하거나 이벤트의 머리말 정보도 활용 하게 된다. 이러한 인터페이스를 상속받은 객체는 JavaCodeBuilder에 의해서 룰 정의 정보에서 실시간에 비교해 볼 수 있는 객체로 변환된다.

5.2.2. IExecutableAction Interface

IExecutableAction 인터페이스는 룰의 액션 처리 문장을 실행 가능한 객체로 변환 시 상속 받는 인터페이스이다. 룰에 의해서 적용되어야 하는 액션 코드인 자바 프로그램 코드가 정의된다. 액션 코드는 자바에 정의된 함수 호출이 가능하고 원격메서드호출(RMI)이나 소켓을 사용하면 외부에 있는 시스템이나 외부 서버에 영향을 줄 수도 있다. 이 인터페이스를 상속받은 객체는 JavaCodeBuilder에 의해서 실시간에 실행 가능한 액션 객체로 변환된다.

6. 룰의 저장방식에 따른 성능 분석

이벤트 파싱 엔진의 구현에 있어서 룰을 저장하는 방식을 테이블과 트리로 나누어서 구현하였다. 테이블 저장방식은 룰의 이벤트 토큰을 벡터에 저장하고 여러 개의 룰을 해쉬테이블에 저장하는 방법이다. 테이블 저장방식은 파싱 과정에서 매칭 될 수 있는 룰 즉, 벡터를 하나씩 선택하면서 매칭해보는 방법을 사용한다. 트리 저장방식은 룰의 이벤트 토큰을 입력되는 이벤트에 따른 검색과정을 미리 트리로 구현해 놓는 방법이다.

테스트는 테이블 기반의 룰 엔진과 트리 기반의 룰 엔진을 룰의 개수와 이벤트의 개수에 따른 비교 분석, 룰 엔진 운영시 발생하는 이벤트의 개수와 룰 매칭률에 따른 비교 분석을 하였다. 테스트 시스템은 팬티엄3-1G, 512M, Windows 2000 Server, JDK 1.3.2를 사용하였다.

6.1 룰의 개수에 따른 파싱 시간 비교

룰의 개수에 따른 파싱 시간 비교는 실제 Runtime시가 아닌 파싱 엔진의 순수한 파싱 시간만을 테스트 하였으며 그림 4와 같다. 이벤트의 타입과 룰에 존재하는 이벤트 토큰의 길이를 5개로 고정하고 룰의 개수를 5에서 200개 까지 늘리면서 트리 방식과 테이블 방식을 테스트 하였다.

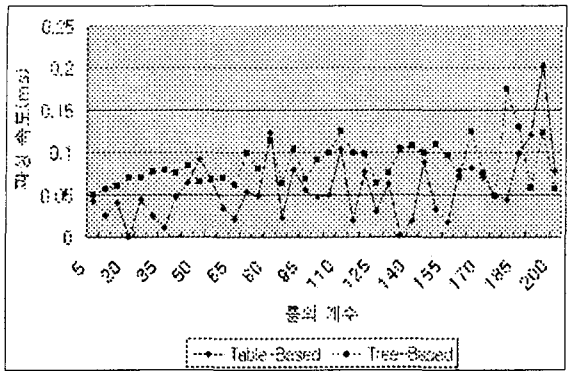

그림 4 룰의 개수에 따른 파싱 속도

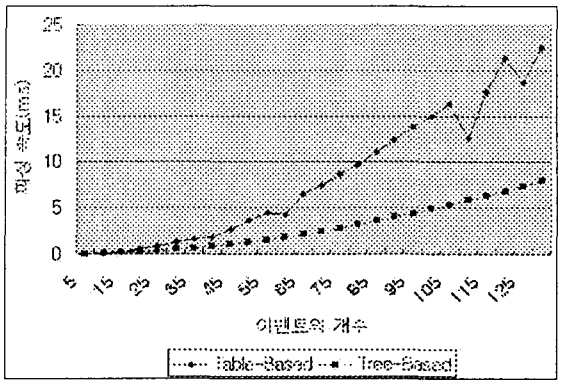

그림 5 이벤트 개수에 따른 파싱 속도

테스트 결과 룰의 개수가 적을 때는 테이블 방식의 파싱 속도가 낮게 나오나 룰의 개수가 커질수록 트리 방식의 파싱 속도가 낮게 나온다. 룰의 개수가 적을 때는 테이블 방식의 저장방법이 단순한 알고리즘 때문에 빠르지만 룰의 개수가 커짐에 따라서 룰을 하나씩 비교해보는 테이블 방식보다는 트리 방식의 성능이 더 빠르다는 것일 알 수 있다.

6.2 룰에 정의된 이벤트 토큰의 개수에 따른 파싱 시간 비교

그림 5는 그림 4와 같은 상황에서 룰의 개수를 5개로 고정하고 이벤트 개수와 룰의 이벤트 토큰을 5에서 125까지 늘려나가면서 파싱 속도만을 테스트한 결과이다. 이벤트의 개수가 증가함에 따라서 테이블 방식의 룰 엔진은 파싱 속도가 트리 방식보다 2,3배 이상으로 거 걸리는 것을 알 수 있다. 트리 방식의 특성상 이벤트 매칭을 함에 있어서 매칭 되는 이벤트의 리스트가 트리의 경로에 해당하기 때문에 경로 상에 있지 않는 이벤트가 발생했을 경우 룰에 맞지 않는 것을 테이블 방식보다 빠르게 알 수 있다. 따라서 이벤트의 개수가 많아짐에 따라서 트리 방식이 테이블 방식보다 효과적으로 파싱 할 수 있다.

6.3 Rumtime시 이벤트 개수에 따른 평균 Latency 비교

그림 6은 이벤트 파싱 엔진을 실제 사용하는 RunTime환경에서 룰 500개, 룰의 이벤트 토큰 사이즈 50, 이벤트 타입 10개 일 때 쓰레드를 사용해서 이벤트를 계속해서 발생시켜 하나의 이벤트가 룰 엔진의 버퍼에서 대기하다가 파싱 처리되는 평균 Latency를 보인다. 이벤트의 개수가 증가함에 따라서 테이블 방식은 파싱 시간이 증가함에 따라서 이벤트가 입력버퍼에 존재하는 시간이 커져서 평균 Latency가 급격히 증가하게 된다.

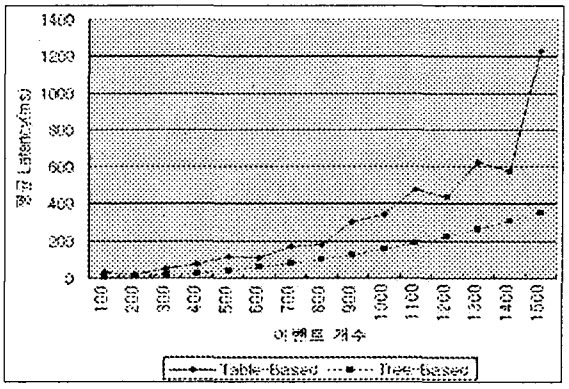

그림 6 이벤트 개수에 따른 평균 Latency

6.4 RumTime시 룰에 정의된 이벤트 토큰의

매칭률에 따른 평균 Latency 비교

그림 7은 그림 6과 같은 실행 상황에서 룰에 정의된 이벤트 토큰의 최소 매칭 비율을 조절함에 따른 이벤트의 평균 Latency를 나타낸다. 최소 매칭 비율은 룰의 이벤트 토큰을 난수를 기반으로 이벤트 리스트를 미리 생성해 놓고 이벤트를 발생 시킬 때 일정한 비율의 이벤트를 미리 생성한 난수 기반의 이벤트를 리스트를 입력함으로써 룰 매칭 과정에서 강제로 매칭 될 수 있는 조건을 만들어서 테스트 하였다.

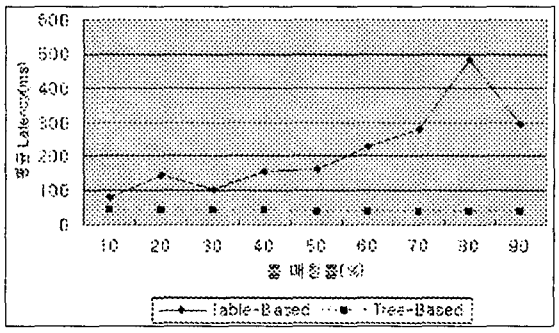

그림 7 룰 매칭률에 따른 평균 Latency

트리방식의 경우 매칭률이 증가함에 따라 Latency가 감소한다. 그 이유는 이벤트 버퍼 내에 존재하는 시간이 줄어들기 때문이다. 테이블 방식은 매칭률의 증가에 따른 Latency가 감소하는 량보다 테이블 방식의 파싱 처리시간 이 더 큰 영향을 끼치기 때문에 Latency가 증가하게 된다.

7. 결론

본 논문에서는 시스템에 이벤트가 왜 필요한지, 이벤트 기반의 시스템은 어떻게 이루어져야 하는지에 대해서 설명하였다. 논문에서 제시하는 후크 객체 기반, 이벤트 객체 기반의 기존의 룰 기반 접근 방법에서 변형된 새로운 방법으로 룰 엔진을 설계하였다. 룰 엔진을 설계함에 있어서의 이슈인 룰의 이벤트 토큰 저장방식을 테이블 방식과 트리 방식으로 나누어서 설계 및 구현하였다. 테이블 방식과 트리방식의 룰 엔진을 룰 개수, 이벤트 개수, 룰 매칭률에 따른 이벤트 파싱 시간과 Latency를 비교 분석하였다. 결과적으로 이벤트 파싱을 위해서는 테이블 방식 보다는 트리 방식의 구조가 더 좋은 성능을 보였다.

참고문헌

[1] Yongseok Park, "Event correlation " ,IEEE Potentials , Volume: 20 Issue: 2 , Apr/May 2001 ,Page(s): 34 -35.

[2] Appleby, K., Goldszmidt, G., Steinder, M., "Yemanja - a layered event correlation engine for multi-domain server farms ",Integrated Network Management Proceedings, 2001 IEEE/IFIP International Symposium on , 2001 ,Page(s): 329 -344.

[3] Robert D. Gardner & David A. Harle," Methods and Systems for Alarm Correlation " ,IEEE, 1996.

[4] CORBA Notification Specification 1.0.1, August 2002

[5] Mark Grand, "Patterns in Java Volume 1 : A Catalog of Reuseable Design Patterns Illustrated with UML", wiley computer publishing, 1998.

[6] Elliot Berk, "JLex: A Lexical Analyzer Generator for Java(TM)" available at URL: http://www.cs.princeton.edu/~appel/modern/java/JLex/

[7] Yoonhee Kim, Hariri, S., Djunaedi, M., "Experimental results and evaluation of the proactive application management system(PAMS)", Conference Proceeding of the IEEE International , Feb 2000, Page(s): 76 -82.

[8] Jakobson, G., Weissman, M., "Alarm correlation ", IEEE Network , Volume: 7 Issue: 6 , Nov 1993 , Page(s): 52 -59.

● 저자소개 ●

윤태웅

2001 건국대학교 공과대학 컴퓨터공학과 학사

2003 건국대학교 정보통신대학 컴퓨터.정보통신공학과 석사

2003~현재 건국대학교 정보통신대학 컴퓨터.정보통신공학과 박사과정

관심분야: 분산 시스템, 소프트웨어 공학

E-Mail : taewoong@konkuk.ac.kr

민덕기

1986 고려대학교 공과대학 산업공학과 학사

1991 미시건 주립 대학 컴퓨터공학 석사

1995 미시건 주립 대학 컴퓨터공학 박사

1995~현재 건국대학교 정보통신대학 컴퓨터공학부 교수

2000~현재 소프트웨어 컴포넌트 포럼 기술분과 위원장

관심분야: 분산 및 병렬 시스템, 분산 객체 및 컴포넌트 기술, 미들웨어, 소프트웨어 시스템 아키텍쳐, 시스템 성능 분석, 웹 기반 분산 컴퓨팅, 웹 서비스

E-Mail : dkmin@konkuk.ac.kr