MOP 기반 모니터링 참조 모델 구축

오정진*, 조상식**
*전남과학대학 자치경찰과
**전남과학대학 게임제작과
e-mail:jjoh@chunnam-c.ac.kr, sscho@chunnam-c.ac.kr

A monitering reference model construction of MOP foundation

Jeong-Jin Oh*, Sang-Sik Cho**
*Dept of Local Police, Chunnam Techno College
**Dept of Game manufacture, Chunnam Techno College

요 약

본 연구에서는 모니터링 정보의 생성, 처리, 분산 및 표현에 수반되는 사양을 정형화된 주석으로 표현하여 구현이 함께 이루어질 수 있는 하나의 자동 코드 생성 알고리즘을 제안하는데 있다. 일반적인 패러다임은 언어 및 사양 독립적이지만, 제안되는 패러다임은 모니터링 지향 프로그래밍 영역에 속하는 것으로 하나의 프로타입 형태의 모니터링 지향 환경이 구현되고 제안된다. 입력으로 하나의 공식을 취하는 프로그램을 제공하는 방식으로 모니터링 지향 프로그래밍 환경에 대한 새로운 로직을 추가하고 미리 정의된 서식으로 출력이 이루어지도록 하는데 있다.

키워드 : 모니터링, MOP, 프로토타입

1. 서론

본 연구에서는 모니터링 정보의 생성, 처리, 분산 및 표현에 수반되는 사양을 정형화된 주석으로 표현하여 구현이 함께 이루어질 수 있는 하나의 자동 코드 생성 알고리즘을 제안하는데 있다. 주석으로 표현된 정형화된 사양은 구현에 의해 생성된 추적 정보에 대한 검토를 런 타임 시에 진행할 수 있으며, 설계를 통해 상호작용이 이루어질 수 있다. 일반적인 패러다임은 언어 및 사양 독립적이지만, 제안되는 패러다임은 모니터링 지향 프로그래밍 영역에 속하는 것으로 하나의 프로타입 형태의 모니터링 지향 환경이 구현되고 제안된다. 어떤 요구사항들은 특정 논리 공식으로 잘 표현될 수 있는 반면에, 프로그래밍 언어들은 범용성을 추구하고 있다. 이러한 이유로 인해 모니터링 지향 프로그래밍 환경은 목표 프로그래밍 언어 상부에서 모니터링 논리적인 틀을 제공할 수 있는 능력을 가져야 한다. 모니터링 논리에 관한 지식들을 감안했을 때, 모니터링 논리의 정보적인 개념에 관한 정형적인 추상화를 나타낸다. 본 연구에서는 입력으로 하나의 공식을 취하는 프로그램을 제공하는 방식으로 모니터링 지향 프로그래밍 환경에 대한 새로운 로직을 추가하고 미리 정의된 서식으로 출력이 이루어지도록 하는데 있다.

※ 본 연구는 2008년도 교육인적자원부 전문대학 특성화 재정지원사업의 지원에 의하여 연구되었음.

하나의 MOP 환경을 설계하고 촉진하는데 따른 여러 가지 인자들을 논의할 필요가 있다. 특히, 현재 지원되고 있는 모니터링 로직들에 관한 공개 소프트웨어 기반의 프로토타입이 제안될 필요가 있다. 또한 시스템 개발 및 분석에 있어서 계약에 의한 설계, 객체지향 측면 및 런 타임 증명과 같은 다른 영역들과의 관계 설정은 매우 중요하다. 본 연구에서는 이러한 측면들에 관련된 사항들을 취급하며, 이에 앞서 정보의 표현은 많은 중간 단계에서 나타날 수 있기 때문에 일반 모니터링 서비스에서 요구되는 것으로서 조합될 수 있는 활동 집합으로 모니터링 모델을 나타낸다.

본 연구의 제안 방식은 언어와 로직 독립적이며, 객체지향 측면에서 다른 시스템 개발 영역과 자연스럽게 통합될 수 있다. 특히 Eclipse 환경에서 자바로 구현되는 하나의 프로토타입은 확장된 정규적인 식일 뿐만 아니라 과거 시점과 미래 시점을 선형 시제 논리로 표현한 요구사항들을 지원하는 자동 코드 생성의 로직엔진이라 할 수 있다.

본 연구의 구성은 2장에서 관련 연구로 시스템 요구사항들이 MOP 관점에서 어떻게 접근되어야 할 것인지를 고찰하고, 3장에서는 MOP 환경 구축을 위한 모니터링 참조 모델을 소개하고, 통합적인 모니터링 정보 관점에서 주석 형태로 표현되는 정형화된 사양과 구현을 결합하는 로직엔진으로서 새로운 틀을 제시하였으며, 4장에서는 MOP 로직 엔진의 설계와 구현을 다루었으며, 마지막으로 본 연구의 결론을 5장

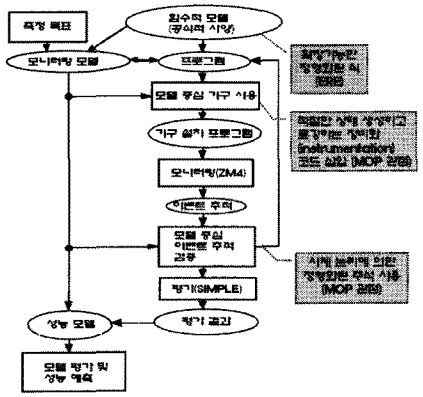
에 기술하였다.

2. 관련연구

2.1 모델 지향 모니터링

현재 사용되고 있는 모니터링 시스템의 유형과 작동 원리 분산 시스템에서 프로그램들의 성능과 기능적 행위들에 대한 관측과 분석을 허용하는 모니터링 시스템에 관하여 연구되었다[1]. 모니터링 시스템은 성능 평가, 튜닝과 디버깅에 사용되며, 이에 관련된 접근 방식은 모델 지향적 모니터링으로 불린다. 모델 지향 모니터링에서 이벤트 지향 모니터링과 이벤트 기반 모니터링은 동적 행위의 같은 추상 즉, 이벤트에 의존하기 때문에 하나의 방법론으로 결합된다. 그래프, 패트리 네트 혹은 큐잉 모델들을 사용한 프로그램 행위에 관한 하나의 이벤트 기반 정형적인 모델은 하나의 모니터링 모델을 드라이브하기 위해 사용된다.

모니터링 모델은 원하는 추상 레벨에서 프로그램 행위를 상술하기 위해 같은 이벤트 집합을 사용하는 기능적 모델의 부분집합이다. 어떤 도구들의 지원 하에서 체계적인 프로그램 도구화, 이벤트 인식, 이벤트 추적 정보 검증 및 새로운 성능 모델 생성을 위해 사용된다. <그림 1>은 MOP 관점에서 모델 지향 모니터링 시스템의 구성을 나타낸 것이다.



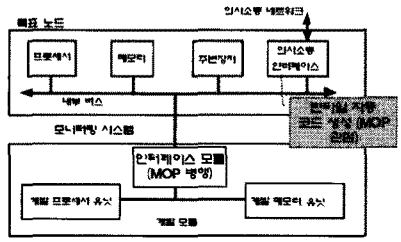
<그림 1> MOP 관점에서 모델 지향 모니터링 시스템 구성

2.2 실시간 다중 프로세서 구조

다중 프로세서는 실시간 모니터링을 목표로 하는 non-invasive 시스템을 들 수 있다[2]. 이러한 시스템은 하드웨어 모니터링 기법과 testing 및 디버깅 목적으로 개발되었으며, 다음과 같은 가정을 기반으로 하는 접근방식이 사용된다.

① 모니터링되는 분산 시스템은 클라이언트/서버 구조로 서버는 모든 클라이언트들 간에 상호작용을 제어한다.
② 메모리 보호를 갖는 캐시 혹은 가상 주소법은 사용할 수 없다.
③ 의사소통은 프로세스들 간에 상호작용으로만 사용되며, 노드들 간에 프로세스 이주는 모니터링 중에 허용되지 않는다.
④ 순환 호출은 사용될 수 없다.

<그림 2>는 2가지 주요 성분들로 구성되는 모니터링 시스템의 구조이다. 인터페이스 모듈은 사용자에 의해 미리 정의된 집합을 근거로 목표 시스템의 내부 상태들을 제어하는 부분으로 MOP 환경 조성이 가능한 부분이다.



<그림 2> 모니터링 시스템 구조

2.3 MOP를 통한 기존 모니터링 모델의 경량화

기존 모니터링 시스템 모델들은 목적에 따라 크게 생성, 처리, 분산, 표현 등 4가지 활동들을 중심으로 하고 있으며, 이들 활동들에 관한 MOP 관점에서 접근 가능한 영역들은 <표 1>과 같이 나타낼 수 있다[3].

<표 1> MOP 관점에서 모니터링 참조 모델

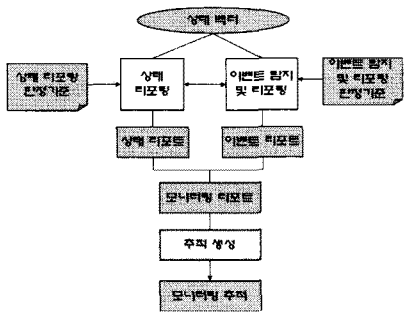
모니터링 정보 생성	- 상태 리포트 - 이벤트 탐지와 리포트 - 추적 생성(MOP 관점에서 런 타임 시에 수행)
모니터링 정보 처리	- 병합, 다중 추적 생성 및 검증 (MOP 관점에서 시제 논리에 따른 자동 코드 생성) - 데이터베이스 업데이트 - 조합, 여과, 분석
모니터링 정보 분산	- 분산 서비스 예약자 등록 - 정보 선택 판정기준 사양 (MOP 관점에서 정형화된 주석 처리)
모니터링 정보 표현	- 텍스트 출력, 타임 처리 다이아 그램 - 이벤트 및 상태 애니메이션 - 추상 레벨의 사용자 제어, 갱신을 위한 정보 위치 및 시제 논리의 사용자 제어 (MOP 관점의 정형화된 사양 사용) - 다중 동시 관점 - 상호작용 메시지 내용물 가시성

3. MOP 기반 모니터링 참조 모델

3.1 모니터링 정보의 생성

모니터링 데이터는 상태와 이벤트 리포트(event report)의 형태로 생성된다. 리포트 수열은 모니터링 추적(monitoring

trace) 생성에 사용될 수 있다. 상태 보고하기(reporting), 이벤트 탐지 및 추적 생성은 <그림 3>과 같이 나타낼 수 있다. 상태 리포트에 관한 정형화된 사양은 상태 벡터로부터 값들의 부분 집합을 포함하며, 다른 관련 정보를 포함할 수 있다.



<그림 3> 모니터링 리포트 및 추적 생성

3.2 모니터링 정보의 처리

모니터링 서비스는 모니터링 요구 사항들의 부합을 위한 여러 가지 방식으로 결합될 수 있는 블록 구축과 같은 어떠한 기능적인 유닛 혹은 단위들을 제공할 수 있다. 시간 주기에 걸쳐서 시스템 활동의 상이한 논리적 관점들을 제공하기 위해서 모니터링 추적 정보가 여러 가지 방식으로 구축되고 순서화 되어야 한다. 모니터링 추적 정보들이 처리되는 방식을 결정하기 위한 선택 기준들로 사용될 수 있는 하나의 보고서 속성들은 다음과 같은 것들을 포함해야 한다.

① 시제 논리에 의한 시간스탬프 생성, 리포트 우선순위 혹은 유형
② 보고 개체의 식별, 우선순위, 유형
③ 리포트가 참조하는 관리 객체의 식별자 혹은 유형
④ 리포트 도착지 식별 혹은 유형

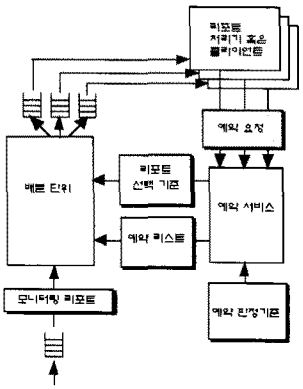
3.3 모니터링 정보의 분산 및 표현

객체들에 의해 생성된 모니터링 정보는 이러한 정보의 서로 다른 사용자에게 배분되어야 할 것이다. 이러한 보고서들의 도착은 사용자, 관리자, 다른 모니터링 객체 혹은 처리 개체들일 수 있다. 분산 계획은 아주 간단한 것으로부터 아무 복잡하고 특화된 것에 이르는 범위를 가진다. 하나의 고정된 계획의 예는 모든 사용자들에 대한 보고들의 브로드캐스팅이다. 이 경우에 모니터링 서버는 모든 사용자들에게 전향될 것이다. 특화되고 복잡한 분산 계획은 예약 원리에 의한 것일 수 있기 때문에 MOP 접근 방식이 가장 바람직하다.

생성, 처리 및 수집된 모니터링 정보는 고객의 특수 목적에 따라 적절한 서식으로 표현되어야 한다. 적절한 사용자 인터페이스는 사용자에게 정보 출력 방식뿐만 아니라 다음과 같은 사항들을 처리할 수 있어야 한다.

① 시스템에 의해 생성된 많은 모니터링 데이터
② 이러한 정보의 여러 가지 추상 레벨
③ 모니터링 데이터에 의해 표현된 시스템 활동의 고질적인 병렬적 성질
④ 정보가 생성 및 표현되는 속도와 비율

<그림 4>는 모니터링 보고 배분의 MOP 접근을 나타낸 것이다.



<그림 4> 모니터링 보고 배분의 MOP 접근

사용자에게 정보를 출력하는 여러 가지 방식이 사용될 수 있다. 여기에는 일부 기존 도구들의 유용한 특징들이 고려되어야 한다. 여러 가지 표현 기법에 관해 우선적으로 병렬적인 디버깅 시스템으로의 디버깅 데이터 출력 방식이 사용될 수 있다. 유사 기법들은 구성, 성능, 보안, 회계 등에 관련된 모니터링 정보의 여러 가지 유형들의 출력이다.

가장 보편적인 출력 형태는 모니터링 정보의 간단한 텍스트 표현이다. 여기에는 강조 및 색상을 포함한다. 이벤트들은 Traveler 와 같은 시제적인 순서라기보다는 인과관계로 출력한다. Jade 모니터링 시스템은 자체 과정 의사소통, 과정 생성과 죽이기와 같은 프로세스 레벨 이벤트들을 관측하고, 이벤트 상술을 위해 한 두 라인의 텍스트를 출력하는 텍스트 콘솔을 제공한다. 이벤트 시작 과정의 이름, 이벤트 타입, 이벤트 주제가 되는 과정의 이름, 가능하면 첫 라인을 나타내준다. 만일, 이벤트가 프로세스들이 통신하는 것이라면, 메시지 내용이 출력의 두 번째 라인으로 등장한다. 적합한 새김 눈, 하이라이트, 색칠은 가시화의 표현력을 증가시키는데 사용될 수 있으며, 또한 여러 가지 추상적인 수준에서 모니터링 정보를 구별시킨다.

이러한 기법의 강점은 모니터링 데이터 표현에 어떠한 특수 장치도 필요하지 않다는 점이며, 이러한 정보를 텍스트 형태로 변환시키기가 간단하다는 점에서 MOP 접근방식이라 할 수 있다. 그렇지만, 이러한 기법은 병렬적인 시스템 활동들을 나타내는 데는 적합하지 않다. 현재 MOP와 관련하여 Eclipse의 활용은 간단한 텍스트 형식의 표현 방식과 다른 보다 복잡한 표현 기법이 함께 사용되고 있다. 병렬적 시스템

의 상태는 2차원 다이아 그램으로 표현된다. MOP 환경의 구성은 시스템의 현재 상태, 그러한 상태를 유도하는 이벤트들의 순서를 나타내주며, 이러하여 시간에 관해 행위들의 패턴들을 출력할 수 있다. 시간 단위는 이벤트 출현 혹은 실시간 주기일 수 있다. 1~2개 캐릭터들이 객체 혹은 객체 그룹과 관련된 가능한 각 이벤트들을 나타내는데 사용될 수 있다. Java-MOP에 의한 시제논리의 장점은 모니터링 정보가 간단한 텍스트 화면상에서 제시될 수 있다는 것이다.

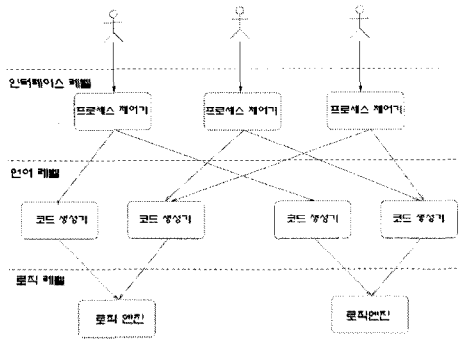
4. MOP 로직 엔진의 설계와 구현

모니터링 정보 생성, 처리, 분산 및 표현 활동에 관련된 사양들을 MOP 관점에서 정형화시키기 위한 제반 작업들을 나타낸다. 이러한 일련의 작업들은 일종의 MOP 기반 모니터링 참조 모델의 구축이라 할 수 있다. 문제는 이러한 참조 모델을 가능한 경량급의 정형화된 방식으로 구현할 수 있는 새로운 플러그인 형태의 로직엔진 개발이다. 이러한 로직 엔진의 개발과 관련하여 선형 시제 논리를 통한 정형화된 사양 구축과 확장된 정규식 표현을 갖는 플러그인 프로토타입을 제시한다. 제시된 프로토타입은 런 타임 시에 사양과 구현이 결합되는 방식으로 자동 코드를 생성할 수 있다.

4.1 MOP 로직 엔진의 구조

MOP 로직엔진의 설계를 위해 필요한 특징들은 크게 5가지 부문으로 나눌 수 있다. 첫 번째는 역할 분리에 관한 것이고, 두 번째는 언어 및 논리가 독립적이어야 하며, 세 번째는 자동화, 확장성 및 유연성이다. 네 번째는 인라인과 오프라인 모니터링 선택에 관한 것이고, 마지막은 경량급의 정형화된 방식이다.

〈그림 5〉는 MOP 환경 구조를 나타낸 것이다.

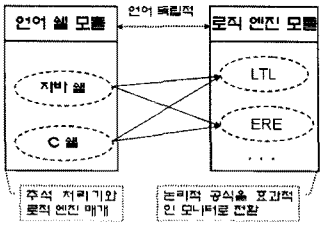


〈그림 5〉 MOP 환경의 구조

3개 레벨의 모듈들을 고려할 수 있는데, 그것은 인터페이스, 언어 및 로직 레벨이다. 하위 레벨에서 모듈들은 인접된 상위 레벨에서 모듈들에 의해 재사용될 수 있다. 인터페이스 레벨에서 모듈들은 프로세스 제어기라 할 수 있으며, 언어 레벨에서 모듈들은 코드 생성기들이라 할 수 있다. 로직 레벨에서 모듈들은 로직 엔진이 된다. 프로세스 제어기들은 3가지 주된 기능들을 가진다. 첫 번째는 주석들로부터 정형화된 사양들을 추출해서 대응하는 코드 생성기들에 그것들을 발송하는 것이다. 두 번째는 코드 생성기들의 출력을 수집해서 호스트 언어의 실행 가능한 코드로 변환시키는 것이다. 세 번째는 사용자, 텍스트 혹은 그래픽 기반으로 인터페이스를 제공하는 것이다.

4.2 사양 중심의 모니터 통합을 위한 틀

사양을 정형화시키기 위해서는 새로운 논리적인 틀이 제안될 필요가 있다. 시스템의 새로운 성분들로서, 특히 로직 플러그인으로 새로운 논리적인 정형화가 필요하다. 구체적으로 하나의 로직 플러그인은 〈그림 6〉과 같은 기본 모듈 구조를 가질 필요가 있다.



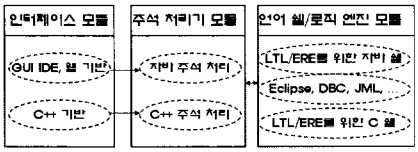
〈그림 6〉 사양 정형화를 위한 기본 모듈 구조

모듈들 간에 프로토콜을 표준화시킴으로써 새로운 모듈들이 쉽고 독립적으로 추가될 수 있다. 특정 프로그래밍 언어와 사양 정형화를 위해 생성된 모니터는 언어 쉘 모듈에 의해 생성되며, 주석 처리기와 로직 엔진의 중간 모듈이라 할 수 있다. MOP 환경에서 로직 엔진의 구성은 MOP의 핵심적인 특징이 된다. 로직 엔진 모듈은 로직 공식을 효과적인 모니터링이 될 수 있도록 하는 고유 변환기가 된다. 로직 엔진 모듈의 출력 형태는 추상적인 의사코드로서 언어 쉘 모듈에 의해 특정 목표 코드로 변환된다.

자바 모델링 언어인 JML은 자바를 위한 하나의 사양 언어로서 자바 클래스 및 인터페이스를 위한 설계들을 상술하는데 사용되며, 런 타임 디버깅과 정적 분석을 위한 기반을 제공한다. JML 선언들은 고정된 논리적인 정형화를 사용하여 특수한 주석들을 표현하기 때문에 쉽게 MOP 환경을 구축할 수 있도록 한다. 계약에 의한 설계로 지칭되는 DBC는 Eiffel 환경에서 특히 잘 지원되는 소프트웨어 설계 방법론이다. DBC는 사양들이 런 타임 점검들로 컴파일 되는 선언 및 불변적인 것들로서 프로그램과 결합될 수 있도록 한다. 자바를 위한 목적으로 JASS 및 jContractor와 같은 언어로 제안된 DBC 확장들이 있다. DBC는 로직 플러그인에 있어서 일정한 서식으로 접근하기 때문에 MOP 환경을 위한 특수한 경우라 할 수 있다. Eclipse 환경에서는 로직 엔진 모듈은 언어 쉘 모듈의 한 부분으로 통합될 수 있다.

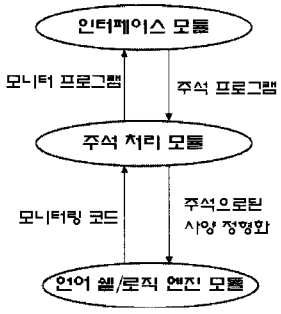
사양 정형화의 기본 구조는 주석들을 정의하고, 처리하고 디버깅할 수 있는 인터페이스 모듈 및 특정 모니터링 프로그

래밍 언어로 구체화되는 주석 처리기 모듈들과 통합될 수 있다. 주석 처리기 모듈은 하나의 프로그램 스캐너로서 특수한 모듈로 처리될 주석들과 주석 처리 내용이 원하는 구조가 되지 못했을 경우에 오류 보고서를 작성하는 것을 구별한다. 주석 처리기는 사양들을 추출해서 대응하는 로직 플러그인으로 급송하는 형태가 된다. 그런 다음에 생성된 모니터링 코드를 수집해서 구성 속성들에 따라 모니터링 될 프로그램 내에 그것을 통합시킨다. <그림 7>은 윈도우 운영체제에서 C++, 유닉스 운영체제에서 자바라는 관점에서 사양 중심의 모니터 통합을 위한 하나의 논리적 틀을 제시한 것이다.



<그림 7> 사양 중심의 모니터링 통합 틀

인터페이스 및 주석 처리기는 빈번하게 변경되거나 갱신되지는 않는다. 언어 쉘 및 로직 엔진 모듈은 로직 플러그인을 포함하기 때문에 빈번하게 추가, 제거 및 갱신될 수 있다. 새로운 사양 언어를 적용하기 위해서는 새로운 로직 플러그인을 개발하거나 제공되는 웹 저장소를 사용할 필요가 있다. 이러한 작업들은 MOP 개념에 해당된다고 할 수 있다. 새로운 정형화된 공식을 적용하는 주석들은 일종의 버튼을 사용하여 밀어 넣는 개념으로 처리될 수 있다. 인접한 모듈 간에 프로토콜은 유연성 및 확장성의 최대 효과를 거두고 디버깅 및 테스팅을 용이하게 하기 위해 아스키 텍스트를 기반으로 한다. 이 구조에서 성분들은 기본적으로 아스키 텍스트를 받는 개별 프로그램으로 구현되기 때문에 서로 다른 언어들을 사용하여 각자 역할을 하는 방식으로 구현될 수 있다. 이러한 측면에서 하나의 주석 처리 프로그램이 변환되는 과정은 <그림 8>과 같이 수정될 수 있다.



<그림 8> 주석 프로그램의 변환 과정

하나의 로직 플러그인은 정형화된 사양이지만, 그것의 출력 서식은 추상적이다. 따라서 그것이 갖는 내부 모니터 통합 기법에 관계없이 하나의 MOP 지원 틀로서 귀속되기 위해서는 다음과 같은 2가지 구성 성분들이 필요하다. Java-MOP 프로토타입의 경우에는 JAVA의 특성에 따라 선언과 초기화가 엄격히 구분되고 있으며, 모니터링 사양 본체 부분에서 조건 부분을 별도로 취급하고 있다. 모니터링 시스템의 특성에 따라 불가피하게 미래 시제 논리의 사양이 주석으로 등장하는 경우에는 성공 조건을 별도로 취급할 필요가 있지만, 이것 또한 처음 조건이 충족되면 참이기 때문에 이에 관한 별도의 차원을 설정할 필요는 없다.

① 선언 부분 : 모니터 상태를 저장하는 변수들로 구성되며, 이들 중에 일부는 모니터링 시작을 위해 사용된다. 이들은 목표 프로그래밍 언어와 모니터 구성 속성에 종속된다.
② 모니터링 사양 본체 : 모니터 중단이 있을 때마다 실행되는 모니터 메인 부분으로 실패 조건들을 포함한다. 요구사항들이 위반되는 경우에 예외처리 혹은 시스템을 안정 상태로 놓거나 프로그램에 새로운 기능을 붙이는 복구 기능을 갖도록 해야 한다.

모니터링 로직 플러그인의 설계와 구현은 대단히 어려운 작업으로 여러 가지 기존 가능성들과의 균형이 요구되며, 전적으로 새로운 형태의 알고리즘 개발이 필요하다. 현재 대표적인 형태는 모든 모니터 상태들이 명확히 생성된 경우에 따른 로직 플러그인 형태로 오직 하나의 현재 모니터 상태만이 저장된다. 여기에 따른 주된 문제점은 진행 중인 다음 상태의 생성에 관한 것이다. 이러한 측면에서 미래 시점과 과거가 연관되는 계량적이고 지식적인 시제 논리와 이에 대응하는 확장된 정규적인 식의 구성이 요구된다. 이에 관련하여 유한상태머신(FSM)이 고려될 수 있다. 로직 플러그인 구성은 프로그래밍 언어 쉘과 로직 엔진이 결합된 하나의 FSM이 가장 바람직하다.

4.3 로직 엔진 알고리즘

MOP 환경에서 모듈라 방식으로 요구사항들을 표현하기 위한 새로운 논리적 틀을 이상적으로 붙일 수 있다. 정형화된 요구사항들은 실행 가능한 코드로 변환될 필요가 있기 때문에 논리적 틀들은 MOP의 확장성을 용이하게 하기 위해서 표준화된 인터페이스를 제공해야 한다. 하나의 논리 모듈에 관한 입력은 분명히 논리식이지만, 그것의 출력 서식은 덜 뚜렷하게 된다. 모니터링 논리 모듈의 출력이 하나의 MOP 환경에 접속될 수 있도록 하기 위하여 다음과 같이 확장 하였다.

① 선언 : 다음 모니터링 단계를 위해 프로그램이 상술하는 변수들로서 부분 리스트가 유지되어야 한다. 이러한 변수들은 목표 프로그래밍 언어에 종속되는 적절한 위치로 MOP 환경에서 삽입될 필요가 있다. 초기화 단계는 모니터링 시작을 위한 변수를 준비하고, 프로그램 실행 중에 직면하게 되는 첫 모니터링 중단 지점에 이를 때까지 오직 한 번 실행되도록 한다.

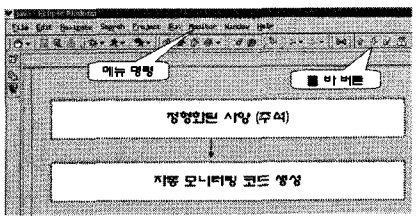
② 모니터링 본체 : 모니터링 본체는 모니터의 주요 부분으로 모니터링 중단에 도달되면 언제든지 실행되어야 한다. 인라인과 오프라인, 동기화와 비동기화와 같은 요구사항 문제에 관해 바람직한 모니터링 타입에 종속되면서, 모니터링 본체는 프로그램의 지속적인 허용에 앞서 실행되거나, 포크 모양으로 분기되면서 병렬로 실행되거나 혹은 다른 머신에서 처리로서 실행될 수도 있어야 한다.

③ 0/1 조건 : 모니터링 요구사항들이 충족되고 있는지 상술하기 위한 조건으로 1은 더 이상 모니터링 될 이유가 없는 것이다. 미래 시간 시제 논리의 문맥에서 하나의 예로 조건1은 "궁극적으로 1"라는 형태의 요구사항이 모니터링 되고 있는 중이고, 1이 포착되기 위해 관측되고 있다고 할 때, 조건은 1이 될 것이다. 자체 변경 코드를 지원하는 하나의 실행 환경의 문맥에서 전적으로 런 타임 오버헤드를 줄이기 위해 조건이 1일 때, 모니터가 완전히 제거될 수 있다. 추적정보가 요구사항을 벗어날 때 0의 조건이 나타난다. 이 조건이 맞는다고 할 때, 사용자 제공의 복구 코드가 실행될 것이다. "복구"는 극소량의 양념과 같은 역할을 해야 한다. 그 이유는 이러한 코드가 하나의 예외 혹은 시스템을 안전 상태로 놓아야 할 뿐만 아니라 프로그램에 새로운 기능을 부착시켜야 하기 때문이다.

이러한 차원들은 과거 및 미래를 나타내는 선형 시제 논리로 이것들을 구현해야 할 뿐만 아니라 확장된 정규 식들로 구현해야 한다. 이리하여 이들 3개 로직들은 MOP 프로토타입으로 지원될 수 있다.

4.4 Eclipse 기반 프로토타입 구현

GUI 툴의 사용은 MOP 환경에 보다 친절한 사용자 인터페이스를 제공할 수 있다. 본 연구에서 GUI 툴은 Eclipse 플랫폼에서 플러그인 형태로 구현된다. Eclipse는 강력하고 확장 가능한 자바 IDE를 포함하고 있다. <그림 9>는 Eclipse의 AJE 섹션을 사용하여 제안된 로직엔진을 플러그인 형태의 프로토타입으로 구현한 결과의 초기 화면이다.



<그림 9> 로직 플러그인 형태의 프로토타입

MOP를 위해 사용되는 부분은 Monitor 메뉴와 툴 바 버튼들이다. 프로토타입은 정형화된 시제 논리의 사양들을 포함한 주석뿐만 아니라 자동 생성되는 코드들을 포함한다. 사용자들은 주석을 통한 항해, 모니터링 코드 생성 혹은 단순 클릭 혹은 핫 키를 사용한 생성 코드 작업이 가능하다. 단적으로 Eclipse에 의해 제공된 다른 특징들과 함께 MOP의 통합 환경을 구축할 수 있다.

5. 결 론

본 연구에서는 MOP 접근이 용이한 모니터링 참조 모델을 정의하였고, 참조 모델을 가능한 경량급의 정형화된 방식으로 구현할 수 있는 새로운 플러그인 형태의 로직엔진이라 할 수 있는 하나의 프로토타입을 제안하였다.

OSI 표준들은 구현 문제의 이슈가 될 수 있는 관리 정보의 표현에 관해 어떠한 것도 정의하지 않고 있지 않다. 모든 관리 측면을 지원하기 위한 하나의 잠재적인 서비스로서 모니터링 분산 시스템에 관한 MOP 접근방식이 절대적으로 필요하다. 또한 정형화된 사양의 모니터링 서비스는 시스템 개발 중에 디버깅에 유용하며, 그것은 애플리케이션 자체의 한 부분으로 필요하다.

본 연구에서 제안된 프로토타입은 선형 시제 논리를 통한 정형화된 사양 구축과 확장된 정규식 표현을 가지며, 런 타임 시에 사양과 구현이 결합되는 방식으로 자동 코드를 생성할 수 있다.

향후 본 연구에서 제안된 프로토타입의 일반화를 위해서는 무엇보다도 비선형 시제 논리의 전개를 위한 새로운 틀이 제안될 필요가 있다. 또한, 모니터링 정보의 생성, 처리, 분산 및 표현에 관한 MOP 접근과 관련하여 보다 통합적인 Eclipse 환경 구축에 관한 연구가 필요할 것이다.

참고문헌

[1] R. Hofmann, R. Klar, N. Luttenberger, B. Mohr, G. Werner, 'An Approach to Monitoring and Modeling of Multiprocessor and Multicomputer Systems', In T. Hasegawa et al., editors, Proc. of the International Seminar on Performance of Distributed and Parallel Systems, pages 91-110, Kyoto, 7-9 Dec. 1988.

[2] J. P. Tsai, K. Y. Fang, H. Y. Chen. 'A Non-interference Monitoring and Replay Mechanism for Real-time Software Testing and Debugging', IEEE Trans. Software Eng., 16, 8, 897-916, 1990

[3] D. C. Marinescu, J. E. Lump, T. L. Casavant, H. J. Siegel, 'Models for Monitoring and Debugging Tools for Parallel and Distributed Software', Journal of Parallel Distributed Computing 9, 2, pp. 171-184, 1990,.

[4] 김정렬, 'PLC활용과 모니터링', ,테크미디어, 2002

[5] 타라 칼리셰인 , '정보 트래핑 : 원하는 정보를 자동으로 수집하는 웹 모니터링 기법 (에이콘 웹 프로페셔널 시리즈 10)', 에이콘출판, 2007