실시간 객체 지향 모델을 위한 시나리오 기반 구현 합성

김 세 화[†] · 박 지 용[††] · 홍 성 수[†††]

요 약

내장형 시스템이 제공하는 기능이 다양해 지고 그 구조가 복잡해짐에 따라, 이들 시스템을 설계하는 데에 객체 지향 설계 방법론이 널리 사용되고 있다. 객체로 설계된 시스템을 대상 하드웨어에서 수행시키기 위해서는 객체들로부터 태스크 집합을 유도해야 하는데, 여기에 몇 개의 태스크가 존재하며 각 태스크가 어떤 객체들로 도착한 어떤 이벤트를 처리하느냐에 따라 시스템의 응답성이 크게 좌우된다. 그럼에도 불구하고 객체와 태스크의 상이함 때문에 최적의 태스크 집합을 유도하는 것은 매우 어려운 일이며, 그로 인해 지금까지는 여러 태스크 집합을 반복적으로 시도해 보는 것이 보편적인 방법이었다. 본 논문에서는 이 문제를 해결하는 Scenario-based Implementation Synthesis Architecture(SISA)를 제안한다. SISA는 객체로 설계된 시스템에서 태스크 집합을 유도하는 방법, 그리고 이를 지원하는 개발 도구와 런타임 시스템 아키텍처를 총칭한다. 이를 이용하여 개발된 시스템은 가능한 적은 개수의 태스크들로 이루어져 있으면서도 시스템의 각 이벤트에 대한 응답 시간이 최소임이 보장된다. 우리는 UML 2.0을 모델링 언어로 사용하는 개발도구인 RoseRT를 확장하여 SISA를 구현했으며, 기 개발된 산업용 PBX(사설 교환기) 시스템에 이를 적용했다. 이 시스템의 성능 평가 결과, 지금까지 알려진 최선의 태스크 유도 방식을 이용하여 개발되었을 때에 비해, 시스템의 최대 응답 시간이 평균 30.3% 단축된다는 것을 확인할 수 있었다.

키워드 : 객체 지향 실시간 시스템 설계, 내장형 소프트웨어 개발 방법론, 자동화된 다중 태스킹 코드 합성, 객체 지향 모델링 도구, 통합 모델링 언어(UML)

Scenario-Based Implementation Synthesis for Real-Time Object-Oriented Models

Saehwa Kim[†] · Jiyong Park[††] · Seongsoo Hong[†††]

ABSTRACT

The demands of increasingly complicated software have led to the proliferation of object-oriented design methodologies in embedded systems. To execute a system designed with objects in target hardware, a task set should be derived from the objects, representing how many tasks reside in the system and which task processes which event arriving at an object. The derived task set greatly influences the responsiveness of the system. Nevertheless, it is very difficult to derive an optimal task set due to the discrepancy between objects and tasks. Therefore, the common method currently used by developers is to repetitively try various task sets. This paper proposes Scenario-based Implementation Synthesis Architecture (SISA) to solve this problem. SISA encompasses a method for deriving a task set from a system designed with objects as well as its supporting development tools and run-time system architecture. A system designed with SISA not only consists of the smallest possible number of tasks, but also guarantees that the response time for each event in the system is minimized. We have fully implemented SISA by extending the RoseRT development tool and applied it to an existing industrial PBX system. The experimental results show that maximum response times were reduced 30.3% on average compared to when the task set was derived by the best known existing methods.

Key Words : Object-oriented Real-time System Design, Embedded Software Development Methodology, Automated Multi-tasking Code Synthesis, Object-Oriented Modeling Tools, Unified Modeling Language(UML)

1. 서 론

급격한 기술 융합의 결과로 최근의 내장형 시스템은 다양한 기능을 동시에 제공하며 매우 복잡한 구조를 가지고 있다. 그래서 이들 시스템을 개발하는 과정에 ROOM[30], OCTOPUS[3], COMET[12]과 같은 객체 지향 설계 방법론을 도입하고자 하는 노력이 많이 있어왔다. 또한 이러한 방법론들이 널리 사용됨에 따라 IBM Rational RoseRT[16], ARTiSAN Real-Time Studio[2], I-Logix Rhapsody[17], Telelogic Tau[36], 그리고 IAR visualSTATE[15]와 같은 상용 모델링 도구들이 등장하여 이러한 방법론들을 지원하고 있다. 이 도구들은 UML과 같은 객체 지향 모델링 언어를

† 준 회 원 : 서울대학교 전기컴퓨터공학부 박사과정
†† 준 회 원 : 서울대학교 전기컴퓨터공학부 박사과정
††† 정 회 원 : 서울대학교 전기컴퓨터공학부 부교수
논문접수 : 2005년 6월 8일, 심사완료 : 2005년 9월 13일

사용하여 시스템을 설계할 수 있게 하고, 설계된 모델을 시뮬레이션을 통해 분석할 수 있게 해 줄 뿐만 아니라, 수행 가능한 코드를 자동으로 생성하여 준다. 이 도구들은 시간에 따라 진화하는 복잡한 실시간 내장형 소프트웨어를 효과적으로 유지, 보수할 수 있게 해줌으로써 널리 사용되고 있다.

이렇게 설계된 시스템을 대상 하드웨어에서 작동시키기 위해서는 객체들로부터 시스템이 수행될 태스크들의 집합을 유도하는 과정이 필요하다. 이 때, 유도된 태스크 집합이 몇 개의 태스크들로 이루어져 있으며, 각 태스크가 어떤 객체로 도착하는 어떤 이벤트들을 처리하는가에 따라 시스템의 응답 시간을 비롯한 실시간 성능이 크게 좌우된다. 그러나 이 과정이 이렇게 중요함에도 불구하고, 객체와 태스크의 상이함 때문에 최적의 태스크 집합을 유도하는 것은 매우 어려운 일이며, 그로 인해 지금까지는 여러 태스크 집합을 반복적으로 시도해 보는 것이 일반적인 해결책이었다. 본 논문에서 우리는 최적의 응답 시간을 보장하는 태스크 유도 방식, 그리고 이를 지원하는 개발 도구와 런타임 시스템 아키텍처를 제안한다. 먼저 기존의 태스크 유도 방식들을 살펴보고 이들의 한계점을 논한다. 그 다음 우리의 접근법을 설명한다.

1.1 연구의 동기

내장형 시스템을 위한 객체 지향 설계 방법론들에서 시스템은 능동 객체라 불리는 정형화된 객체들의 네트워크로 설계된다. 능동 객체란 서로 혹은 시스템 외부와 메시지를 주고 받음으로써만 통신하는 객체이다. 능동 객체는 내부에 상태 기계를 가지고 있어서 받은 메시지의 종류와 현재 상태에 따라 액티비티라고 불리는 핸들러 루틴들 중 하나를 선택하여 수행함으로써 그 메시지를 처리하는 방식으로 동작한다. 액티비티는 수행 중에 새로운 메시지를 생성하여 이를 다른 능동 객체에 전달할 수 있다. 개발자는 프로그래밍 언어(C, C++, Java등)를 사용하여 액티비티가 수행할 코드를 작성한다.

기존에는 태스크 집합을 유도할 때 능동 객체의 단위로 유도하는 방식을 사용해 왔다[3, 8, 9, 12, 30]. 이 방식은 기본적으로 시스템의 능동 객체의 수만큼의 태스크들을 생성시키고, 한 능동 객체의 모든 액티비티들을 같은 태스크가 수행하도록 한다. 그런데 이렇게 유도된 태스크 집합은 태스크들의 개수가 너무 많아질 수 있기 때문에, 이 방식에서는 필요할 경우 개발자들이 이를 수정하여 여러 능동 객체들을 그룹지워 각 그룹의 액티비티들을 하나의 태스크가 수행할 수 있도록 하고 있다. 이러한 특징들을 고려하여 우리는 이 태스크 유도 방식을 능동 객체 기반 태스크 유도 방식이라 명명한다. 이 방식은 직관적이며 구현이 쉽기 때문에 널리 사용되어 왔지만, 다음과 같이 시스템의 응답 시간에 불필요한 시간 지연 요소를 발생시키는 것으로 분석되어[11, 31, 34] 최적의 태스크 유도 방식이 아님이 알려져 왔다.

- 시스템이 응답하기 전에 블로킹이 과도하게 발생한다.
- 시스템이 응답하기 전에 태스크 문맥 전환이 과도하게 발생한다.

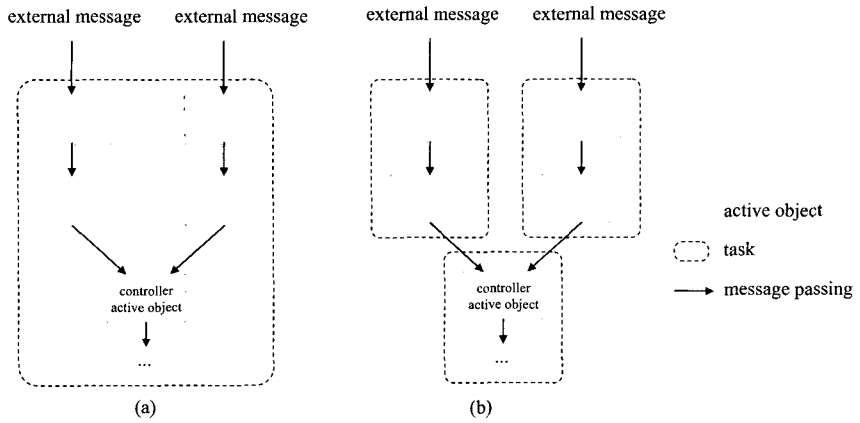
이러한 요인들이 발생하는 과정을 개괄적으로 살펴보면 다음과 같다. 상태 기계의 구현을 간단하게 하기 위해, 각 태스크는 현재 수행중인 액티비티가 끝날 때까지 새로운 메시지를 처리할 수가 없다는 제약 조건을 가지고 있다[11, 26, 31, 34]. 그래서 두 태스크들 사이에 메시지가 전달될 때 대상 태스크에서 이미 어떤 액티비티가 수행 중이면 그 메시지는 즉시 처리되지 못하고 블록 될 수 있다. 이러한 태스크 간 메시지 전달은 시스템이 외부에서의 이벤트를 응답하는 도중에 여러 번 발생할 수 있기 때문에, 결과적으로 시스템은 응답하기 전에 많은 횟수의 블로킹을 경험한다. 더욱이 블로킹은 태스크 문맥 전환을 동반하기 때문에 이에 따르는 추가적인 시간 지연도 함께 겪는다. 이러한 지연 요소들은 개발자가 작성한 액티비티의 코드에 의해 발생한 것이 아니라 태스크 유도 방식이 충분히 최적화 되어 있지 않아 초래된 것들이다.

이런 시간 지연 요인들이 분석됨에 따라 이들의 발생을 줄이기 위한 많은 연구가 있어왔다. 기존의 연구들이 택한 접근법은 몇 개의 능동 객체들을 효과적으로 그룹지어서 이들의 액티비티들을 같은 태스크가 수행시킴으로써, 한 그룹 안의 능동 객체들이 서로 메시지를 주고받을 때에는 태스크 간 메시지 전달이 일어나지 않도록 하는 것이었다. 그룹짓는 방법의 하나로 [8], [9], [12]에서는 개발자로 하여금 동시에 처리될 필요가 없는 메시지들을 파악하여 이들을 처리하는 능동 객체들을 하나로 그룹짓는 방법을 제안하고 있다. 또 다른 방법으로 [31]에서는 개발자가 각 능동 객체의 메시지들이 견딜 수 있는 최악의 블로킹 시간과 액티비티들의 최악 수행시간을 파악하도록 하여, 항상 최악 블로킹 시간이 보장된다는 것이 확인될 때 두 능동 객체들을 그룹짓도록 하고 있다. 그러나 이러한 방법들에는 여전히 다음과 같은 한계점이 있다.

- 개발자들이 제시된 방법들을 실제로 적용하기가 매우 어렵다.
- 블로킹과 문맥 전환을 철저하게 없앨 수 없기 때문에 최적의 태스크 유도 방식이 아니다.

우선 [8], [9], [12]의 방법에서 개발자들은 동시에 처리될 필요가 없는 메시지들을 파악하기 위해, 능동 객체들이 상태 기계에 따라 메시지를 주고 받는 패턴을 완전히 분석해야 하는데, 이는 자동화된 도구의 도움 없이는 매우 고된 일이다. 또한 [31]에서 요구하는 것처럼 시스템의 모든 메시지가 견딜 수 있는 블로킹 시간을 정하고 모든 액티비티의 최악 수행 시간을 측정하는 것도 무척 어려운 작업이다. 특히 여러 외부 이벤트가 공통의 메시지를 발생시킬 경우, 이 메시지가 견딜 수 있는 블로킹 시간은 현재 발생한 외부 이벤트의 중요도에 따라 런타임에 매번 달라져야 한다. 정적 분석을 위해 이를 하나의 값으로 고정시키기 위해서는 이 중 최악의 값을 사용해야만 하는데, 이는 불필요하게 강한 제약 조건으로 시스템 자원의 낭비를 초래하기 쉽다.

이러한 문제점들을 감수할 수 있다고 하더라도, 기존 방법들은 최적의 태스크 유도 방식이 아니어서 개선의 여지가 있다. 왜냐하면 기존의 방식은 다음과 같이 많은 시스템에



(그림 1) 동시에 수행되어야 하는 메시지 흐름들이 공통의 능동 객체를 접근할 경우 태스크 집합을 만드는 방법들의 예.
(a) 모든 능동 객체를 하나의 태스크에서 수행시킨다.
(b) 공통으로 접근되는 능동 객체를 별도의 태스크에서 수행시킨다.

서 블로킹과 문맥 전환을 발생시킬 수 있기 때문이다. 대부분의 내장형 시스템에서는 (그림 1)과 같이 메인 컨트롤러 역할을 하는 능동 객체가 존재하여 동시에 처리되어야 하는 메시지 흐름들 중 많은 것들이 이 능동 객체로 도착하게 된다[13, 42]. 그런데 앞서 언급했듯이 기존 방식에서는 한 능동 객체로 도착한 모든 메시지들이 한 태스크에서 처리되어야 한다. 따라서 (그림 1)의 (a)와 같이 모든 능동 객체들을 하나의 태스크에서 수행시키는 방법만이 태스크 간 메시지 전달을 제거할 수 있는 유일한 방법이다. 그러나 이렇게 하면 동시에 처리되어야 하는 메시지들이 한 태스크에서 순차적으로 처리되는 병목 현상이 발생하여 오히려 시스템의 응답성이 더욱 저하되는 문제가 있다. 따라서 대부분의 경우 (그림 1)의 (b)와 같이 컨트롤러와 같은 능동 객체는 별도의 태스크로 수행되는 것이 보통이다. 이렇듯 기존 방법들을 사용해서는 블로킹과 이에 따르는 문맥 전환을 완전히 제거하는 것이 불가능한 경우가 많다. 이러한 문제점들 때문에 개발자들은 차라리 경험과 직감에 의존하여 여러 태스크 집합들을 만들고 이들이 원하는 응답 시간을 만족시키는지를 반복적으로 확인하는 임시방편적인 방법을 사용하고 있는 실정이다.

한편, 앞서 언급했듯이 대부분의 개발 도구들은 설계된 시스템 모델에서 수행 가능한 코드를 자동 생성하는 기능을 제공한다. 이 과정에서 추가적으로, 개발자는 자신이 원하는 태스크 집합을 기술하여 시스템이 그 태스크들로 수행되도록 한다. 우리는 이를 태스크 집합의 적용이라고 부른다. 그런데 기존의 개발 도구들에서는 태스크 집합의 적용 과정이 매우 까다로우며 불합리하게 구성되어 있다는 문제점이 있다. 이런 점들 때문에 실제로 개발자들은 코드 생성 기능의 사용을 기피하는 경향이 있어 왔다. 구체적으로 다음과 같은 문제점들이 있다.

- 태스크 집합을 적용하기 위해서 시스템의 논리적인 동작을 기술하는 객체 모델을 수정해야 한다
- 태스크 집합을 적용하는 정형화된 방법이 제공되지 않아서 불편할 뿐만 아니라, 제공되더라도 그 기능이 매우 제한적이다.

태스크 집합을 적용하는 과정은 시스템의 시간적 특성을 모델링 하기 위한 과정이기 때문에, 논리적인 동작 특성을 모델링 하는 객체 설계 과정과는 별도로 구분되어 이루어져야 한다. 그러나 기존의 개발 도구들에서는 이 둘이 명확히 분리되지 않고 전자를 수행하는 도중 이미 설계가 끝난 객체 모델을 다시 수정해야 하는 문제점이 있어왔다. 이러한 재 수정은 위험한 일로, 특히 여러 태스크 집합을 적용하기 위해 반복적으로 재 수정을 해야 할 경우 객체 모델에 오류가 주입될 가능성이 매우 커진다. 또한, 대부분의 개발 도구들은 이런 과정을 위한 정형화된 방법을 제공하지 않거나, 제공하더라도 그 기능이 매우 제한적이다. RoseRT는 전자의 대표적인 예로, 개발자가 스스로 태스크를 생성하는 코드와 태스크가 수행할 액티비티들의 능동 객체를 선택하는 코드의 일부분을 작성하여 객체 모델에 수동으로 포함시켜야 하는 불편함이 있다. Rhapsody는 후자의 예로 능동 객체들을 그룹짓는 정형화된 방법을 제공하기는 하지만, 서로 다른 능동 객체에 속한 하위 능동 객체들을 그룹지을 수 없다는 제한이 있다.

1.2 해결 접근법

우리는 이러한 문제들을 해결하는 Scenario-based Implementation Synthesis Architecture(SISA)를 제안한다. SISA는 객체로 설계된 시스템에서 태스크 집합을 유도하는 기법, 그리고 이를 지원하는 개발 도구와 런타임 시스템 아키텍처를 총칭한다. SISA의 핵심은 우리가 시나리오 기반 태스크 유도 방식이라고 명명한 태스크 집합 유도 방식에 있다. 이 방식은 가능한 적은 개수의 태스크들로 이루어져 있으면서도 시스템의 각 이벤트에 대한 응답 시간이 최소임이 보장되는 태스크 집합을 유도하는 방식이다.

이 방식의 기본 아이디어는 세 가지이다. 첫째, 시나리오의 개념을 도입하여 한 시나리오를 이루는 모든 액티비티들을 같은 태스크에서 수행시키도록 한다. 여기서 시나리오란 [22], [25], [26], [32]에서 도입한 개념으로 시스템의 한 외부 이벤트(메시지)에 의하여 진행되는 액티비티와 메시지들의 연속을 뜻한다. 이렇게 하면 태스크 간 메시지 전달이 전혀

필요하지 않게 되어서 이로 인한 블로킹과 문맥 전환 부하가 사라진다. 둘째, 시나리오 그룹의 개념을 고안하여 이를 태스크 생성의 기본 단위로 한다. 시나리오 그룹이란 같은 외부 메시지에 의해 시작되는 시나리오들의 집합을 말한다. 우리는 시스템에 존재하는 각 시나리오 그룹별로 태스크를 마련하고, 한 시나리오 그룹에 속하는 모든 시나리오들을 같은 태스크로 수행시키도록 한다. 이렇게 하면 각 시나리오 마다 별도의 태스크를 생성할 때에 비하여 태스크의 개수를 크게 줄일 수 있다. 이에 더하여 필요에 따라 여러 시나리오 그룹들을 하나의 태스크에서 수행시킬 수 있도록 하여 태스크의 개수를 더욱 줄일 수도 있다. 개발자가 서로 동시에 수행될 필요가 없는 시나리오 그룹들을 지정하면 SISA는 이들을 하나의 태스크로 묶어서 수행시켜 문맥 전환 부하와 태스크가 차지하는 메모리를 더욱 줄이게 된다. 셋째, Immediate Priority Ceiling Inheritance Protocol(IIP) [5, 18]을 사용하여 위의 기법을 사용하면서 초래될 수 있는 동기화 블로킹을 최소화한다. IIP는 [5]에서 제안된 실시간 동기화 기법으로 이 기법을 사용하면 각 시나리오는 최초의 코드가 수행되기 전에 최대 한 번만 블록될 수 있으며, 코드가 일단 수행되면 블록되지 않는 것이 보장되어 시스템은 최적의 응답 시간을 가진다.

SISA는 시나리오 기반 태스크 유도 방식과 더불어, 이를 지원하기 위한 개발 도구와 런타임 시스템 아키텍처도 제공한다. 우선, 개발 도구의 측면에서 시나리오와 시나리오 그룹들을 자동으로 분석해 주고 개발자가 이를 조작할 수 있도록 하는 기능이 제공된다. 개발자는 능동 객체의 상태 기계를 분석할 필요 없이 시스템에 어떤 시나리오들이 존재하는지를 명확히 파악할 수 있다. 그리고 객체 모델을 수정하지 않고도, 이들이 수행될 우선순위를 부여하고, 동시에 수행될 필요가 없는 시나리오 그룹들을 묶는 작업들을 손쉽게 할 수 있다. 다음으로, 시나리오 기반 태스크 유도 방식을 지원하기 위해 기존의 방식과는 다른 런타임 시스템 아키텍처를 제공한다. 기존에는 메시지를 받을 능동 객체에 따라 이를 처리할 태스크가 정해졌기 때문에 능동-객체-태스크의 매핑 테이블이 사용되었다. 그러나 본 방식에서는 시나리오-그룹-태스크와 시나리오-우선순위의 두 개의 매핑 테이블을 사용하여, 메시지가 속한 시나리오 그룹에 따라 수행될 태스크가 선택되고 런타임에 생성되는 메시지의 종류에 따라 시나리오가 분기될 때 태스크의 우선순위가 동적으로 조절되는 구조의 아키텍처가 사용된다.

우리는 UML 2.0을 모델링 언어로 사용하는 RoseRT를 확장하여 SISA를 구현했다. 또한 우리는 이의 유용성을 보이기 위해 구현된 SISA를 기 개발된 사설 교환기 (PBX) 시스템에 적용하였으며, 그 응답 성능을 평가했다. 이 시스템은 산업용으로 개발된 것으로, 고도로 재구성되는 동적 구조와 전형적인 계층 구조와 같은 복잡한 응용에서 발견되는 많은 특징들을 가지고 있다. 성능 평가 결과, 시스템의 응답시간이 평균적으로 30.6% 감소하였으며, 미션 크리티컬 이벤트에 대한 응답 시간은 최대 88.6%까지 감소함을 확인할 수 있었다.

논문의 구성은 다음과 같다. 2 장에서는 SISA에서 가정하는 시스템 모델에 대하여 설명하며, SISA가 해결코자 하는 문제들을 명확히 규명한다. 3 장에서는 이 문제들을 해결하는 핵심 기법인 시나리오 기반 태스크 유도 방식을 설명한다. 4 장에서는 이 기법을 지원하기 위한 개발 도구와 런타임 아키텍처의 구체적인 내용을 설명한다. 5 장에서는 SISA를 이용하여 개발된 시스템의 성능 평가 결과를 기술한다. 6 장에서 관련 연구에 대하여 소개하고, 마지막으로 7 장에서 결론을 맺는다.

2. 문제 기술

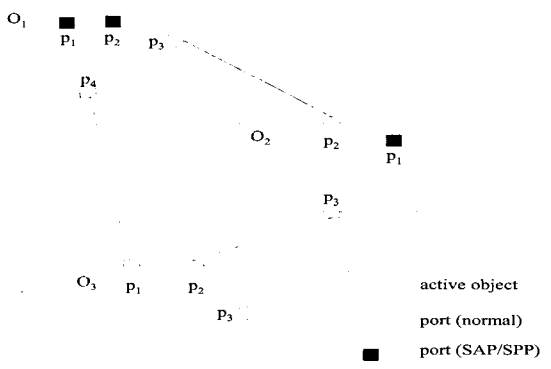
본 장에서는 SISA가 해결하고자 하는 문제들을 명확히 규명한다. 그 전에 앞으로 진행할 논의의 이해를 돕기 위해 SISA가 가정하는 시스템의 모델에 대해 설명한다.

2.1 시스템 모델링

SISA에서 시스템은 두 가지 관점으로 모델링 될 수 있다. 그 중 하나는 능동 객체들의 네트워크로 모델링 하는 것이다. 다른 하나는 시나리오들의 집합으로 모델링 하는 것이다. 전자는 시스템에 어떤 능동 객체들이 존재하는지, 각 능동 객체는 어떻게 동작하는지, 이들의 연결관계는 어떤지, 시스템 외부에서의 이벤트는 어디로 전달되는지 등을 나타내는 모델이다. 우리는 구체적인 모델링 언어로 UML 2.0의 구조화된 클래스(SructuredClass) 다이어그램[26]을 사용하여 이 모델을 설명하기로 한다. 후자는 시스템에 어떤 시나리오들이 존재하며, 각 시나리오들이 어떤 시나리오 그룹에 속하는지, 각 시나리오가 수행될 때 어떤 액티비티들이 어떤 순서로 수행되는지 등을 나타내는 모델이다. 우리는 시나리오 트리라고 명명한 다이어그램을 고안하며, 이를 사용하여 시나리오 그룹과 그 안의 시나리오들을 표현 한다. 이제 이 두 가지 모델링 방식에 대해 순서대로 설명한다.

2.1.1 능동 객체의 네트워크로서의 모델링

이 모델링 방식에서 시스템은 능동 객체들의 네트워크로 모델링 된다. 능동 객체는 메소드 호출 없이 서로 혹은 시스템 외부와 메시지를 주고 받음으로써만 통신하는 객체이다. 각 능동 객체는 포트라는 인터페이스 객체를 통해 메시지를 비동기적으로 주고 받는다. (그림 2)가 이렇게 모델링 된 시스템의 예를 보여주고 있다. 여기에서 큰 사각형은 능동 객체를, 작은 사각형은 포트를 나타낸다. 연결된 포트들은 그 포트들을 통해 메시지가 전달될 수 있음을 나타낸다. 메시지는 전달될 대상 능동 객체, 메시지의 종류, 데이터, 우선순위의 네 가지 항목으로 이루어진다. 이들 중 메시지의 우선순위는 두 개 이상의 메시지가 동시에 능동 객체에 도착할 경우 어떤 메시지를 먼저 처리할 지를 결정하는 데 사용된다. 우리는 우선순위의 숫자가 클수록 높은 우선순위를 나타내는 것으로 정한다. 다른 항목들은 그 의미가 명확하므로 설명을 생략한다.

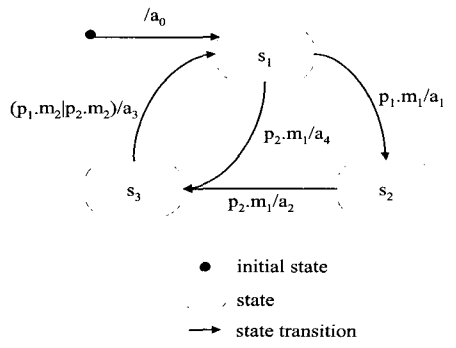


(그림 2) 능동 객체들의 네트워크로 모델링 된 시스템의 예

일반적으로 포트는 능동 객체 사이의 메시지 전송을 위해 사용된다. 그러나 실시간 시스템에서 능동 객체는 센서, 타이머, 모터와 같은 시스템 외부 환경과도 메시지를 주고 받을 수 있어야 한다. 이를 위해 SISA는 ROOM에서의 특수화된 포트인 Service Provision Points(SPP)와 Service Access Points(SAP)의 개념[30]을 도입한다. 전자는 외부 환경에서 메시지를 받기 위한 포트이며, 후자는 외부 환경으로 메시지를 보내는데 사용되는 포트이다. 이 두 종류의 포트들은 (그림 2)에서 일반 포트들과는 다르게 능동 객체의 안에 속이 채워진 작은 사각형으로 표시된다.

한편, 한 능동 객체의 행동은 상태 기계로 모델링 된다. 능동 객체가 포트에서 메시지를 하나 받을 경우, 상태 기계의 천이 규칙에 따라 상태 천이가 발생한다. 어떤 상태 천이가 일어날지는 어떤 포트에서 어떤 종류의 메시지를 받았는가로 결정된다. 상태 천이가 발생하면 그에 해당하는 액티비티가 수행되어 메시지가 처리된다. 액티비티의 코드는 개발자에 의해 작성되며, 메시지에 포함된 데이터를 기반으로 계산을 하거나, 포트를 통해 다른 능동 객체에 메시지를 보내는 등의 작업을 할 수 있다. (그림 3)은 한 능동 객체의 상태 기계의 예를 보여주고 있다. 여기에서 p.m/a는 포트 p로부터 종류가 m인 메시지가 도착할 경우 a라는 액티비티가 수행됨을 나타낸다.

상태 기계의 동작을 이해하는 데서 유의할 점은 액티비티들이 완전 수행 의미 체계(run-to-completion semantics) [26]에 의해 비선점적으로 동작한다는 점이다. 이는 상태 천이가 일어날 조건을 만족시키는 메시지가 도착하였더라도 능동



(그림 3) 한 능동 객체의 행동이 상태 기계로 모델링 된 예

객체가 이미 어떤 액티비티를 수행 중이라면 그 액티비티가 종료된 다음에서야 새로운 액티비티를 수행할 수 있다는 제약 조건이다. 이런 제약 조건을 사용하는 이유는 한 상태 기계의 여러 액티비티들이 공유 자원을 동기화 없이 접근할 수 있도록 하여, 개발자가 액티비티의 코드를 쉽게 작성할 수 있도록 하기 위함이다[26].

2.1.2 시나리오의 집합으로서의 모델링

SISA에서 시스템은 시나리오들의 집합으로도 모델링 될 수 있다. 앞서 언급했듯이, 시나리오는 순서대로 수행되는 메시지와 액티비티들의 연속이며, 같은 외부 이벤트에 의해 시작되는 시나리오들은 하나의 시나리오 그룹으로 집합을 이룬다. 우리는 시나리오 그룹과 그 안의 시나리오들을 표현하기 위해 시나리오 트리를 고안하여 사용한다. 시나리오 트리는 외부 이벤트에 의해 시작하여 액티비티들이 수행되면서 선택적으로 분기되어 여러 개의 시나리오들로 나누어지는 것을 트리 형식으로 표현한 것이다.

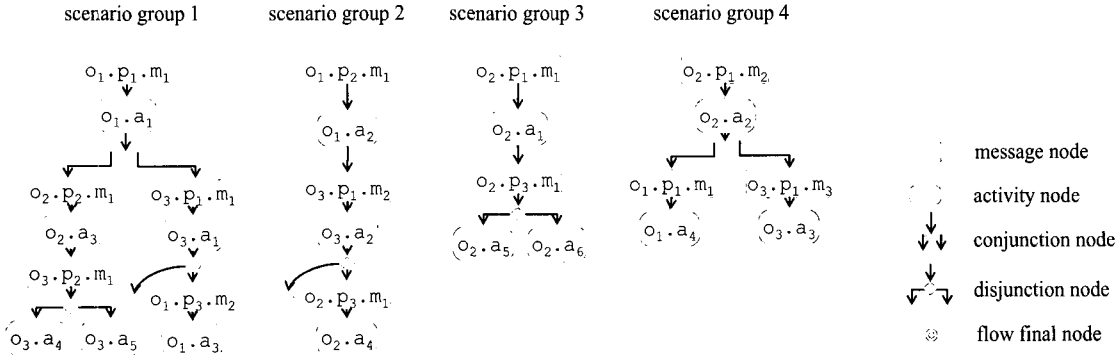
시나리오 트리는 UML 2.0의 액티비티 다이어그램을 확장한 것으로, 여섯 가지 타입의 노드들과 이들을 연결하는 가지(branch)들로 구성된다. 노드들의 종류로는 (1) 액티비티가 수행됨을 나타내는 액티비티 노드, (2) 메시지가 생성되어 전송됨을 나타내는 메시지 노드, (3) 한 시나리오가 두 개 이상의 병렬적으로 수행되는 시나리오들로 나누어 짐을 나타내는 연언 노드, (4) 수행 중에 결정되는 조건에 따라 시나리오가 선택적으로 분기됨을 나타내는 이접 노드, (5) 마지막으로 더 이상의 메시지 발생이 없음을 나타내는 흐름 최종 노드 가 있다. 이렇게 구성된 시나리오 트리에서는 트리 하나가 한 시나리오 그룹을 나타내며, 루트 노드에서 특정 액티비티 노드까지의 한 패스가 한 시나리오를 나타낸다. 또한 각 시나리오는 우선순위를 가질 수 있으며 이는 그 시나리오의 마지막 액티비티 노드에 표시된다.

시나리오 트리를 이용하여 앞의 (그림 2)에서 능동 객체의 네트워크로 표현된 시스템을 모델링 한 예가 (그림 4)이다. 그림에서 시스템은 총 네 개의 시나리오 트리들로 모델링 된다. 각 트리의 루트 노드들은 해당하는 시나리오 그룹이 각각 어떤 SPP에서 어떤 메시지를 받을 때 시작하는지를 알려준다. 시나리오 트리에서 메시지는 o.p.m, 액티비티는 o.a의 형태로 표시되며 o, p, m, a는 각각 능동 객체, 포트, 메시지 종류, 액티비티 이름을 뜻한다.

2.2 문제 정의

우리는 이렇게 모델링 될 수 있는 시스템에서 다음과 같은 문제를 해결하고자 한다.

- 능동 객체로 설계된 시스템이 수행될 태스크 집합이,
- 최적의 응답 시간을 가지며
- 가능한 적은 개수의 태스크들로 이루어지도록 태스크 유도 방식을 정한다.
- 그리고 이를 지원하는 개발 도구와 런타임 시스템 아키텍처를 설계한다.



(그림 4) 시나리오 트리를 이용하여 모델링 된 시스템

먼저 우리는 응답 시간을 시스템이 외부 환경으로부터 메시지를 받은 시점부터 특정 액티비티가 종료하기까지의 시간으로 정의한다. 즉 응답 시간은 하나로 정의되는 것이 아니라 시스템에 존재하는 각 시나리오마다 정의되는 것으로, 이들이 수행을 마치기까지의 시간을 각각 그 시나리오의 응답 시간이라고 부르기로 한다. 그리고 응답 시간이 최적이라는 것은, 둘 이상의 시나리오가 동시에 진행 중일 때, 중요도가 낮은 시나리오 때문에 중요도가 높은 시나리오의 진행이 블록되는 기간이 최소가 된다는 것으로 정의한다. 이렇게 되면 미션 크리티컬 시나리오일수록 응답 시간을 지연시키는 요소가 최소가 된다는 점에서 최적이라고 부른다. 이러한 조건들에 더하여 태스크 집합은 가능한 한 적은 개수의 태스크로 이루어져 있어야 한다. 태스크의 개수를 줄이는 것은 내장형 시스템에서 문맥 전환 부하와 메모리 사용량의 감소를 위해 꼭 필요하다. 마지막으로 이 태스크 유도 방식이 실제로 사용되기 위해서는 개발자들이 이를 손쉽게 이용할 수 있는 개발 도구가 제공되어야 하며, 이를 효과적으로 지원하는 런타임 시스템 아키텍처가 설계되어야 한다.

3. SISA에서의 시나리오 기반 태스크 집합 유도 방법

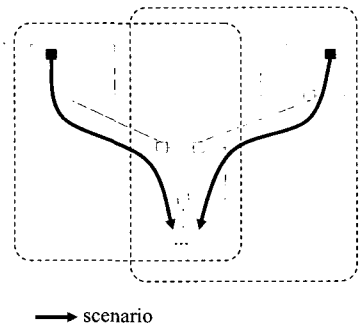
우리는 이러한 문제를 해결하는 SISA를 제안한다. SISA는 객체로 설계된 시스템에서 최적의 응답시간을 위한 태스크 집합을 유도하는 기법, 그리고 이를 지원하는 개발 도구와 런타임 시스템 아키텍처를 총칭한다. 본 장에서는 SISA의 핵심 개념인 시나리오 기반 태스크 유도 방식(scenario-based task set derivation method) 에 대해 설명한다. 개발 도구와 런타임 시스템 아키텍처는 다음 장에서 자세히 다룬다.

시나리오 기반 태스크 유도 방식은 시나리오들의 응답 시간이 각각 최소임이 보장되면서도 가능한 적은 개수의 태스크들로 이루어진 태스크 집합을 유도하는 방식이다. 이 유도 방식의 기본 아이디어는 다음의 세 가지이다. 첫째, 태스크 간 메시지 전달을 제거하기 위해 한 시나리오를 이루는 액티비티들은 같은 태스크로 수행되도록 한다. 둘째, 태스크의 개수를 줄이기 위해 시나리오 그룹별로 태스크를 마련한다. 이에 더하여 몇 개의 시나리오 그룹들을 하나의 태스크로 수행할 수도 있게 하여 태스크의 개수를 더욱 줄인다. 셋째, 동기화 때문에 발생할 수 있는 블로킹을 Immediate Priority Ceiling Inheritance Protocol (IIP)[5]를 사용하여 최소화한다. 이제 이 세 가지에 대해 구체적으로 설명한다.

3.1 태스크간 메시지 전달의 제거

SISA는 태스크 간 메시지 전달을 제거하기 위해 한 시나리오를 이루는 액티비티들은 같은 태스크로 수행되도록 한다. 이렇게 할 경우 한 시나리오는 다음과 같은 방식으로 진행된다. (1) 시나리오가 시작하면, 즉 시스템 외부에서 메시지가 도착하면, 이를 처리하도록 할당된 한 태스크가 선택되어 수행된다. (2) 태스크는 메시지를 받는 능동 객체의 상태 기계를 수행하여 액티비티를 선택하고 수행한다. (3) 액티비티는 수행 중에 메시지를 생성하여 다른 능동 객체로 보낼 수 있다. (4) 액티비티가 끝나면 태스크는 메시지를 받은 능동 객체에 대해 다시 2)를 수행한다. 더 이상 메시지가 생성되지 않을 때까지 (2)-(4)의 과정이 반복된다.

이렇게 하면 연속되어 수행되는 액티비티들은 모두 같은 태스크에서 수행되기 때문에, 어떠한 경우에라도 태스크 간 메시지 전달이 근본적으로 제거된다. 특히 같은 능동 객체에 도착하는 메시지들이라고 할 지라도 서로 다른 시나리오에 속한 경우 서로 다른 태스크에 의해 처리된다. 따라서 (그림 5)와 같이 여러 시나리오에서 공통으로 접근하는 능동 객체가 존재하더라도 서로 다른 외부 메시지에 의해 시발되는 시나리오들은 서로 다른 태스크로 수행된다.



(그림 5) 서로 다른 시나리오에 의해 공통적으로 접근되는 능동 객체가 존재할 때, 각 시나리오는 서로 다른 태스크에서 수행된다. 범례는 (그림 1)과 동일하다.

3.2 태스크 개수의 최소화

위와 같이 한 시나리오를 이루는 모든 액티비티를 한 태스크로 수행한다는 것은 한 시나리오 마다 한 태스크를 마련한다는 것을 암시하고 있다. 그러나 보통 한 시스템에는 아주 많은 수의 시나리오가 존재할 수 있기 때문에 이렇게 하면 태스크의 개수가 많아져서 그에 따르는 문맥 전환 부하나 메모리의 소모가 커지게 된다. 우리는 이 문제를 해결하기 위해 다음의 두 가지 방법으로 태스크의 개수를 줄인다.

- 시나리오가 아닌 시나리오 그룹별로 태스크를 마련하도록 한다.
- 필요할 경우, 여러 시나리오 그룹을 하나로 묶어 한 태스크로 수행하도록 하여 태스크의 개수를 더 줄인다.

먼저 한 시나리오 그룹 안에 있는 모든 시나리오들은 같은 우선순위로 시작하는 시나리오들이다. 따라서 이들 시나리오들을 별도의 태스크로 수행시킨다고 하더라도 이들은 서로 선점할 수 없고 하나씩 수행된다. 즉, 모든 시나리오들을 하나의 태스크로 수행시킬 때와 같이 수행된다. 그러므로 이렇게 하여도 시나리오의 응답 시간은 증가하지 않는다.

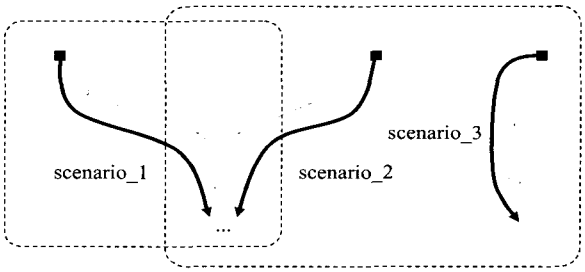
필요에 따라 태스크의 개수를 더 줄이고 싶을 경우, 개발자는 서로 동시에 수행되지 않을 시나리오 그룹들을 명시하여 이들이 하나의 태스크로 수행되도록 할 수 있다. 이러한 시나리오 그룹들의 전형적인 예는 시나리오 그룹들을 시작시키는 메시지들의 시간적 선후 관계가 명확할 경우이다. 예를 들어, 이동형 로봇은 센서로부터는 장애물까지의 거리 메시지를 받으며, 모터로부터는 움직임을 수행했다는 완료 메시지를 받는다. 전자가 발생하여 모터에 이동 명령을 내리면 그 후 후자가 발생하기 때문에 이 두 메시지 사이에는 선후 관계가 존재한다. 따라서 이 두 메시지로부터 시작하는 시나리오 그룹은 동시에 수행되지 않기 때문에 하나의 태스크로 수행시킬 수 있다.

3.2 동기화 블로킹의 최소화

지금까지 설명한 방법들을 사용하면 가능한 적은 개수의 태스크를 사용하면서도 태스크 간 메시지 전달로 인한 블로킹을 제거할 수 있다. 그러나 아직 동기화에 의한 블로킹은 여전히 발생한다. 동기화에 의한 블로킹은 서로 다른 시나리오들이 공유 자원을 사용하기 위해 동기화 기법을 사용하면서 발생할 수 있는 블로킹을 뜻한다. 구체적으로 다음의 두 가지 경우에서 동기화에 의한 블로킹이 발생할 수 있다.

- 서로 다른 시나리오들이 한 능동 객체를 공유할 경우, 완전 수행 의미 체계를 지키기 위해 공유되는 능동 객체에 대한 접근을 동기화한다.
- 서로 다른 시나리오 그룹이 한 태스크로 수행될 경우, 하나의 태스크 문맥이 여러 시나리오 그룹에 의해 공유되기 때문에 이에 대한 접근을 동기화 한다.

이러한 동기화 블로킹을 예를 들어 설명하면 (그림 6)과 같다. 이 그림에는 세 개의 시나리오가 있으며 scenario_2와



(그림 6) 동기화 블로킹의 예를 보이기 위한 예. 범례는 (그림 1)과 동일하다.

scenario_3가 한 태스크에서, scenario_1이 다른 태스크에서 수행된다. 첫 번째 동기화 블로킹은 scenario_1과 scenario_2 사이에서 일어난다. 그림에서 볼 수 있듯이 이 두 시나리오는 한 능동 객체에 동시에 메시지를 보낼 수 있다. 한 능동 객체에서는 한 번에 하나의 액티비티만이 수행될 수 있기 때문에 이미 한 시나리오가 이 능동 객체에서 수행 중이라면 다른 시나리오는 더 이상 진행되지 못하고 블록 해야만 한다. 이 경우 공유되는 자원은 능동 객체가 된다. 두 번째 동기화 블로킹은 scenario_2와 scenario_3 사이에서 일어난다. 이 경우 두 시나리오 그룹 사이에서 공유되는 능동 객체는 없다. 그러나 태스크의 문맥은 하나이기 때문에 태스크가 이미 어떤 액티비티를 수행 중이라면 그 액티비티와는 다른 능동 객체에 속한 액티비티라고 할 지라도 이전 액티비티를 선점하고 태스크 문맥을 차지할 수 없다. 만약 이러한 조건이 지켜지지 않을 경우 이전 액티비티가 수행 중이던 문맥을 잃어버리는 문제가 발생한다. 따라서 태스크 문맥에 대한 접근은 동기화 되어야 하며 이 때문에 블로킹이 발생한다.

우리는 이러한 두 종류의 동기화 블로킹을 최소화 하기 위해 실시간 동기화 기법인 Immediate priority ceiling Inheritance Protocol(IIP)를 사용한다. IIP는 [5]에서 제안된 실시간 동기화 기법으로 스케줄링 되는 대상이 최초로 수행되기 전에 최대 한 번만 블록될 수 있으며, 일단 수행을 시작하면 더 이상 블록되지 않는 것이 보장되는 기법이다. 더군다나 최초에 발생하는 블로킹 시간은 자신보다 우선 순위가 낮은 스케줄링 대상들의 가장 긴 임계 영역 수행 시간으로 한정된다. IIP가 어떻게 이러한 조건들을 만족시키는지에 대한 구체적인 설명은 이 논문의 범위를 벗어나므로 다루지 않는다. SISA는 시나리오를 스케줄링의 대상으로 삼고, 각 능동 객체와 태스크의 문맥을 공유 자원으로 하여 IIP를 적용한다. 그 결과 최초의 단 한번의 블로킹을 제외한 모든 동기화 블로킹이 사라지며, 이 블로킹의 길이는 보다 낮은 우선순위의 시나리오들의 모든 액티비티들의 수행 시간 중 최대값으로 한정된다.

4. SISA의 개발 도구와 런타임 시스템 아키텍처

본 장에서는 SISA를 지원하는 개발 도구와 런타임 시스템 아키텍처를 설명한다. 설명을 간단히 하기 위해 우리는 가장 널리 알려진 개발 도구 중 하나인 RoseRT[16]를 확장하여

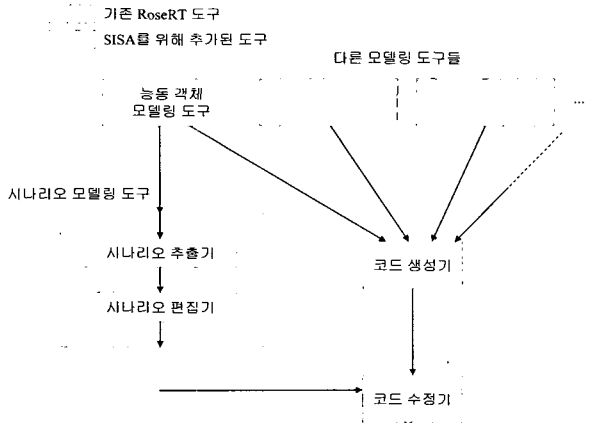
개발 도구를 만들고, RoseRT가 제공하는 런타임 시스템 아키텍처에 기반하여 SISA를 지원하는 아키텍처를 고안한다.

4.1 SISA를 위한 RoseRT 개발 도구의 확장

(그림 7)은 SISA를 지원하는 개발 도구를 보여주고 있다. 여기에서 능동 객체 모델링 도구와 코드 생성기들은 기존 RoseRT에서 제공하는 도구이며, 시나리오 모델링 도구와 코드 수정기는 SISA를 지원하기 위해 새롭게 추가된 도구이다. 첫째, 능동 객체 모델링 도구는 개발자가 시스템을 능동 객체의 네트워크로 모델링 할 수 있게 해 주는 도구이다. 둘째, 코드 생성기는 능동 객체 모델링 도구를 비롯한 여러 모델링 도구를 이용하여 설계된 모델들을 바탕으로 수행 가능한 코드를 생성하는 도구이다. 셋째, 시나리오 모델링 도구는 시나리오 추출기와 시나리오 추출기의 하부 도구들로 구성되며 이들은 각각 다음과 같은 작업을 한다. 전자는 능동 객체 모델링 도구로 설계된 모델을 분석하여 시나리오들을 추출한 다음 이를 기반으로 시스템을 시나리오들의 집합으로 모델링 한다. 이 과정은 매우 기계적으로 이루어지기 때문에 그 구체적인 알고리즘은 부록에 첨부한다. 후자는 개발자가 각 시나리오들의 우선순위를 설정할 수 있게 하고, 시나리오 그룹들을 묶어 하나의 태스크에서 수행되도록 할 수 있는 환경을 제공한다. 이와 더불어 이 도구는 개발자가 명시적으로 우선순위를 설정하지 않은 기타 시나리오들에 대해 적당한 우선순위들을 자동으로 할당하는 작업도 한다. 넷째, 코드 수정기는 코드 생성기가 생성한 코드가 SISA의 런타임 아키텍처에 맞게 동작하도록 수정하는 도구이다. 이 도구는 시나리오 모델링 도구가 제공하는, 시나리오와 시나리오 그룹에 대한 정보를 기반으로 동작한다.

새로 추가된 두 도구들은 XML 포맷의 파일로 시나리오와 시나리오 그룹에 대한 정보들을 교환한다. (그림 8)은 그러한 XML 파일의 예이다. 시나리오 추출기가 이 XML 파일을 생성하면 시나리오 편집기는 이 파일을 시각화하며 개발자가 추가로 제공하는 정보들을 XML 파일에 반영한다.

여기에서 각 scenarioGroup 엘리먼트는 한 시나리오 그룹을 나타내고 있다. 이들은 유일한 값을 가지는 id속성으로



(그림 7) SISA를 지원하는 개발 도구의 구성

```
1  -<scenarioBasedModel>
2    -<scenarioGroup id="1">
3      -<messageNode value="o1.p1.m1">
4        -<activityNode value="o1.a1">
5          <priority value = "2"/>
6          -<disjunctionNode>
7            +<messageNode value="o2.p1.m2">
8            -<messageNode value="o2.p1.m1">
9              +<activityNode value="o2.a1">
10               <priority value = "1"/>
11         </disjunctionNode>
12       </activityNode>
13     </messageNode>
14   </scenarioGroup>
15   +<scenarioGroup id="2">
16   +<scenarioGroup id="3">
     ......
17   -<MC>
18     <refScenarioGroup refID = "1"/>
19     <refScenarioGroup refID = "2"/>
20     <refScenarioGroup refID = "3"/>
21   </MC>
22   +<MC>
     ......
23 </scenarioBasedmodel>
```

(그림 8) 시나리오 모델링 도구가 사용하는 XML 파일의 예

구분된다. 개발자는 특정 액티비티 노드에 우선순위를 할당하여 루트 노드로부터 그 액티비티 노드까지의 패스에 해당하는 시나리오에 우선순위를 할당할 수 있다. 그러면 액티비티 노드를 나타내는 엘리먼트인 activityNode의 priority 속성 값이 개발자가 정한 값으로 기록된다. 또한 개발자는 시나리오 그룹들을 묶을 수 있다. 그러면 묶여진 시나리오 그룹들을 나타내는 MC 엘리먼트가 만들어 지고 연결된 scenario Group의 id값들이 MC의 자식 노드로 추가된다.

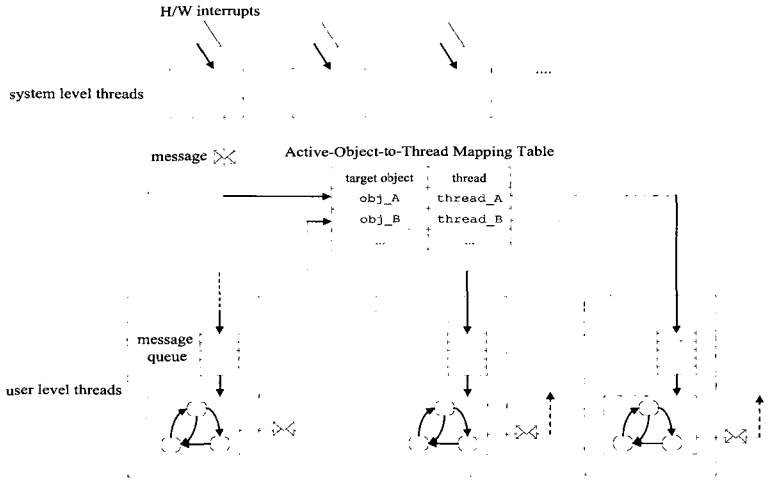
시나리오들의 우선순위는 선행 시나리오의 우선순위가 후행 시나리오의 우선순위보다 항상 크거나 같아야 한다는 제약 조건을 만족해야 한다. 즉 이들을 시나리오 트리로 표현했을 때 상위 노드의 우선순위가 하위 노드의 우선순위 이상이어야 한다. 이는 후행 시나리오를 높은 우선순위로 수행시키기 위해서는 선행 시나리오도 이와 같거나 더 높은 우선순위로 수행되어야 하기 때문이다. 시나리오 편집기는 만약 어떤 노드의 우선순위가 이 조건을 만족시키지 않도록 할당되어 있다면 그 노드의 모든 하위 노드들의 우선순위의 최대값으로 우선순위를 재조정한다. 우선순위가 할당되지 않은 그 외의 노드들에 대해서는 기본값으로 그 노드의 부모 혹은 자식 노드들과 동일한 우선순위를 할당하여 이 조건을 만족시킨다.

4.2 런타임 시스템 아키텍처

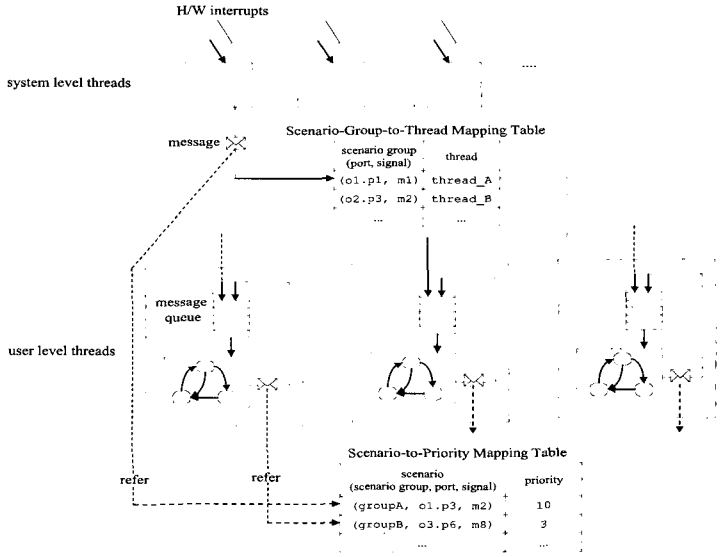
본 절에서는 RoseRT가 제공하는 런타임 시스템 아키텍처를 기반으로 하여 SISA를 지원하는 아키텍처를 구체적으로 다룬다. 우선 전자에 대해 간단히 설명한 다음 SISA를 지원하기 위해 여기에 취해진 추가 사항들을 설명한다.

4.2.1 RoseRT에서의 런타임 시스템 아키텍처

(그림 9)는 RoseRT가 제공하는 런타임 시스템 아키텍처를 보여준다. 한 시스템은 사용자 수준 쓰레드들과 시스템 수준 쓰레드들로 구성된다. 사용자 수준 쓰레드들은 능동



(그림 9) RoseRT가 제공하는 런타임 시스템의 아키텍처



(그림 10) SISA를 지원하기 위한 런타임 시스템의 아키텍처

객체들의 상태 기계들을 수행하는 쓰레드들이며, 시스템 수준 쓰레드들은 하드웨어 인터럽트에 의해 동작하여 능동 객체들의 SPP로 메시지를 전달하는 쓰레드들이다.

시스템 쓰레드가 특정 능동 객체로 메시지를 보내고자 할 때 그 능동 객체가 수행되고 있는 사용자 수준 쓰레드를 찾아야 한다. 이를 위해 *능동-객체-쓰레드* 매핑 테이블이 사용된다. 이 테이블은 각 능동 객체가 어떤 쓰레드로 수행되는지에 대한 매핑 정보가 기록되어 있는 테이블이다. 전달된 메시지는 각 사용자 수준 쓰레드 마다 하나씩 존재하는 메시지 큐에 저장된다. 메시지를 받은 사용자 수준 쓰레드는 메시지 큐에서 가장 높은 우선순위의 메시지를 하나씩 가져온다. 그런 다음 능동 객체의 상태 기계를 통해 그 메시지를 처리할 액티비티가 선택되어 수행된다. 액티비티가 수행중일 때 다른 능동 객체로 메시지를 전달하고자 할 경우 이전과 마찬가지로 능동-객체-쓰레드 매핑 테이블이 참조되어 대상 쓰레드의 메시지 큐에 메시지를 저장한다.

이 아키텍처는 다음과 같은 몇 가지 방법으로 최적화되어 사용될 수 있다. 우선 쓰레드 간 메시지 전달 시 메시지 큐 접근에 의해 발생할 수 있는 블로킹 시간을 줄이기 위해 쓰레드 간 메시지 전달에 사용되는 큐와 동일한 쓰레드 내에서의 메시지 전달을 위한 큐를 분리하여 사용하기도 한다. 또한 메시지 큐에서 우선순위가 가장 높은 메시지를 빠르게 찾을 수 있도록 우선순위 큐나 비트맵 큐를 활용할 수도 있다. 이러한 최적화 방안들에 대한 추가적인 설명은 본 논문의 범위를 벗어나므로 하지 않는다.

4.2.2 SISA를 위한 수정된 런타임 시스템 아키텍처

(그림 10)은 SISA를 지원하기 위해 고안된 런타임 시스템 아키텍처이다. 아키텍처는 RoseRT에서의 아키텍처와 비교하여 *능동-객체-쓰레드* 매핑 테이블이 사용되지 않으며, 대신 *시나리오-그룹-쓰레드*, *시나리오-우선순위* 매핑 테이블, 그리고 IIP가 이 사용된다는 점이 주요한 차이점이다. 구

```
1   SAP-SEND(signal, data)
2       ▷this corresponds to SAP.
3       destinationPort ← connectedTo[this]
4       ▷(port, signal) corresponds to a scenario group.
5       scenarioGroup ← (destinationPort, signal)
6       userThread ← GETTHREAD(scenarioGroup)
7       ▷(scenario group, port, signal) corresponds to a scenario.
8       priority ← GETMESSAGEPRIORITY(scenarioGroup, destinationPort, signal)
9       ▷Creates a new message.
10      message ← (destinationPort, signal, data, priority, scenarioGroup)
11      ENQUEUE(queue[userThread], message)
12      if userThread is idle
13         SETPRIORITY(userThread, priority)
14      else
15         SETPRIORITY(userThread, max(priority, priority[userThread])
```

(그림 11) 시스템 수준 쓰레드에서 사용자 수준 쓰레드로 메시지를 전달하는 작업을 수행하는 코드

체적으로 다음과 같은 사항들이 추가되거나 수정되었다.

- 시스템 쓰레드에서 발생된 메시지를 보낼 사용자 수준 쓰레드를 결정하기 위해 *시나리오-그룹-쓰레드* 매핑 테이블이 참조된다.
- 사용자 수준 쓰레드에서 발생된 메시지는 아무런 매핑 테이블의 참조 없이 항상 동일한 쓰레드로 전달된다.
- 메시지를 생성할 때 *시나리오-우선순위* 매핑 테이블이 참조되어 메시지의 우선순위가 결정된다. 각 쓰레드는 대기 중인 메시지와 현재 처리중인 메시지의 최대 우선순위로 동작한다.
- 사용자 수준 쓰레드는 액티비티를 수행하기 직전에 IIP를 사용하여 능동 객체와 태스크 문맥을 보호한다.

이러한 추가, 수정 사항들이 코드 레벨에서 어떻게 구현되는지 살펴보기로 한다. 우리는 시스템의 초기화 과정과 함께 시나리오가 수행되면서 일어나는 일련의 과정을 설명한다. 단순화를 위해 의사 코드를 사용하여 구현을 논한다.

첫째, 시스템이 초기화될 때 앞으로 사용될 사용자 수준 쓰레드들이 생성된다. 몇 개의 쓰레드들을 생성해야 하는지에 대한 정보는 시나리오 모델링 도구가 제공하는 XML 파일에서 얻을 수 있다. 먼저 다른 시나리오 그룹들과 묶여지지 않은 각 시나리오 그룹들에 대해 각각 한 개의 쓰레드를 생성한다. 그런 다음 묶여진 시나리오 그룹들에 대해서는 한 묶음 마다 한 쓰레드를 생성한다.

둘째, 시스템이 초기화될 때 모든 공유 자원들에 대해 각각 뮤택스를 생성하고 초기화한다. 여기에서 공유되는 자원은 시스템의 모든 능동 객체들과 모든 사용자 수준 쓰레드들이다. IIP의 규칙에 따라 각 뮤택스들의 실링 값은 자신을 접근하는 시나리오들 중 가장 우선순위가 높은 시나리오의 우선순위 값으로 초기화된다. 각 공유 자원들이 어떤 시나리오들에 의해 접근될 수 있는가 하는 정보는 XML 파일을 분석하여 기계적으로 알아낼 수 있다.

셋째, 시스템 수준 쓰레드가 메시지를 발생시킬 때에는 다음의 두 가지 작업을 해야 한다. 우선 생성된 메시지를 처리할 사용자 수준 쓰레드를 찾아야 하며, 메시지를 보낸 다음에는 그 쓰레드가 메시지를 실제로 처리할 수 있도록 우선순위를 조정하는 작업을 한다. 전자를 위해서 우리는 시나리오-그룹-쓰레드 매핑 테이블을 사용한다. 이 테이블은 각 시나리오 그룹과 그 시나리오 그룹을 수행할 쓰레드가 저장되어 있는 매핑 테이블이다. 시나리오 그룹은 특정 SPP로 도착하는 외부 메시지의 종류에 따라 구분되기 때문에 메시지를 받을 SPP의 이름과 메시지의 종류가 이 테이블의 키가 된다. (그림 11)의 5-6 번째 줄이 이러한 작업을 하는 코드이다. 메시지를 보낼 SPP의 이름(*destinationPort*)과 메시지의 종류(*signal*)를 키로 하여 이에 해당하는 시나리오 그룹(*scenarioGroup*)을 수행시킬 사용자 수준 쓰레드(*userThread*)를 찾는다.

후자는 쓰레드가 새로 시작할 시나리오의 우선순위로 동작할 수 있도록 하는 작업이다. (그림 11)의 여덟 번째, 12-15 번째 줄의 코드들이 이를 위한 것들이다. 새로 시작할 시나리오의 우선순위는 시나리오-우선순위 테이블을 참조하여 결정된다. 이 테이블은 각 시나리오들과 그 시나리오를 수행할 우선순위가 기록된 테이블이다. 시나리오는 특정 시나리오 그룹에서 메시지를 받는 포트와 메시지의 종류로 구분되기 때문에 이들이 이 테이블의 키가 된다. 매핑 테이블을 참조하여 우선순위(*priority*)가 정해지면 실제로 사용자 수준 쓰레드(*userThread*)의 우선순위를 설정하게 되는데 쓰레드가 이미 다른 시나리오를 수행 중인가 아닌가에 따라 크게 두 가지 경우로 나뉜다. 먼저 쓰레드가 아무런 시나리오도 수행중이지 않은 휴지 상태일 경우에는 쓰레드의 우선순위는 매핑 테이블에서 찾은 우선순위(*priority*)로 설정된다. 그 반대의 경우에는 현재 그 쓰레드가 수행중인 우선순위(*priority*[*userThread*])와 매핑 테이블에서 찾은 우선순위(*priority*) 중 큰 값으로 우선순위를 설정한다.

넷째, 메시지를 받은 사용자 수준 쓰레드는 메시지 큐에서 가장 우선순위가 높은 메시지를 가져온 다음 상태 기계에 따라 액티비티를 수행한다. SISA를 지원하기 위해서는 이 과정에서 액티비티를 수행하기 직전에 IIP를 사용하여 능동 객체와 쓰레드 문맥을 보호하며, 수행을 마친 다음에 쓰레드의 우선순위를 조정하는 작업이 필요하다. 전자는 (그림 12)의 여덟 번째 줄의 코드에 의해 수행된다. 이 코드는 쓰레드의 우선순위를 메시지를 받는 능동 객체(*destination Object*)와 쓰레드(*this*)의 뮤택스 실링 값 중 최대값으로 증가시킨다. IIP에서는 이렇게 쓰레드의 우선순위를 높이는 것으로 공유 자원들을 보호하는 것을 구현하게 된다. 우선순위가 높아지면 이 능동 객체와 쓰레드를 사용할 다른 시나리오들은 수행을 시작할 수 조차 없게 되어 자동적으로 동기화가 이루어진다.

후자는 (그림 12)의 11번째 줄의 코드에 의해 수행된다.

```
1   MainLoop
2      forever
3         ▷A message with highest priority is dequeued.
4         ▷When there is no message in the queue, thread is blocked.
5         ▷this corresponds to a thread currently running.
6         message ← Dequeue(queue[this])
7         destinationObject ← owner[destinationPort[message]]
8         SetPriority(this, max(ceiling[destinationObject], ceiling[this]))
9         ▷ State is changed, then an activity is selected and executed.
10        RunFSM(this, message)
11        SetPriority(this, (max(for all priorities of messages in queue[this])
```

(그림 12) 사용자 수준 쓰레드에서 메시지 처리를 하는 코드

```
1   Port-Send (signal, data)
2      ▷this corresponds to a port object that is used to send a message.
3      destinationPort ← connectedTo[this]
4      scenarioGroup ← scenarioGroup[currentMessage[owner[this]]]
5      priority ← GetMessagePriority(scenarioGroup, destinationPort, signal)
6      message ← (destinationPort, signal, data, priority, scenarioGroup)
7      ▷currentThread refers to a thread currently running.
8      Enqueue(queue[currentThread], message)
```

(그림 13) 사용자 수준 쓰레드에서의 메시지를 전달하는 코드

이 코드는 현재 메시지 큐(*queue[this]*)에 있는 메시지들 중 가장 높은 우선순위를 가진 메시지의 우선순위로 쓰레드의 우선순위를 설정하는 작업을 한다. 이렇게 하면 능동 객체와 쓰레드 문맥에 대한 보호가 해제되며, 쓰레드의 우선순위가 다음 번 수행될 시나리오의 우선순위로 조정되는 두 가지의 작업을 동시에 이루게 된다.

마지막으로 사용자 수준 쓰레드가 메시지를 전달 하는 코드는 (그림 13)에 나타나 있다. 다섯 번째 줄은 시스템 수준 쓰레드가 메시지를 전달할 때와 마찬가지로 시나리오-우선순위 매핑 테이블을 참조하여 메시지의 우선순위를 정하는 코드이다. 여덟 번째 줄은 생성한 메시지를 쓰레드의 메시지 큐에 넣는 코드이다. 여기서 주의할 점은 메시지 큐를 소유하고 있는 쓰레드가 현재 수행중인 사용자 수준 쓰레드(*currentThread*)라는 점이다. 즉, 메시지를 보내는 쓰레드와 받는 쓰레드가 동일하다.

5. 실험적 성능 평가

이 장에서는 SISA에 의해 생성된 코드의 실시간 성능을 평가한다. 성능 비교를 위해 기존의 접근법을 사용하는 대표적인 개발 도구인 RoseRT에서 생성한 코드를 대조군으로 사용한다. 우리는 이 코드를 생성할 때 기존의 알려진 최선의 방법들[8, 9, 12]을 사용하여 태스크 집합을 유도했다. 실험에 사용된 시스템은 산업용 PBX(private branch exchange, 사설 교환기) 시스템이다. 이 시스템은 여러 대의 셀 폰들과 연결되어 이들의 통화 요청을 서비스하는 시스템이다. 우선 이 실험의 환경 설정에 대해 설명한 다음, 측정한 성능 지표들을 제시하고 실험 결과를 소개하고 분석한다.

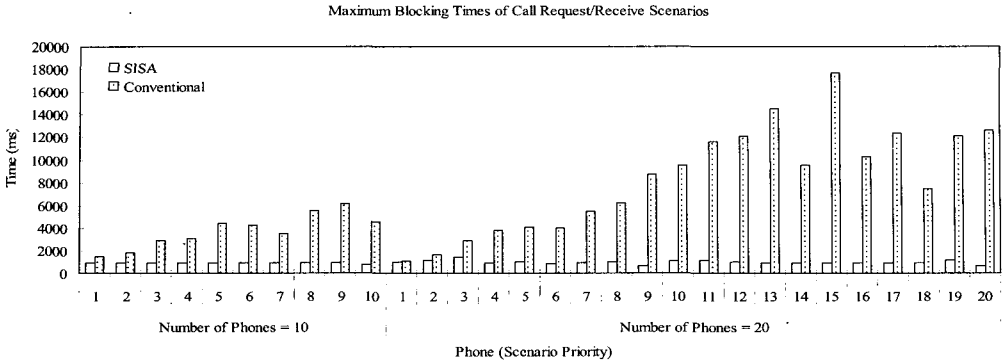
5.1 실험 환경 구축

실험에서 사용된 PBX 시스템은 그 구조와 행동이 동적으로 극심하게 변하는 복잡한 내장형 시스템이다. 우선 이 시스템은 서비스를 받는 폰과 콜 요청, 콜 연결 등에 따라 이에 해당하는 능동 객체들을 동적으로 생성시키고 소멸시키는 것을 반복시키기 때문에 그 구조가 동적으로 변한다. 또한 시스템 내 메시지들의 흐름도 이런 구조의 변화에 따라 동적으로 극심하게 변화한다.

PBX 시스템을 구성하는 능동 객체로는 셀 폰, 시스템 관리자 등을 추상화하는 능동 객체들, 각 콜 요청, 콜 연결, 연결 감시등을 처리하는 능동 객체들이 있다. 이러한 PBX 시스템을 사용하는 액터는 셀 폰, 타이머, 시스템 관리자 등이 있다. 시스템은 이들로부터 외부 입력 메시지를 받아 시나리오들을 시작하게 된다. 이에 따라 PBX 시스템의 시나리오들을 시발하는 외부 입력 메시지들은 (1) 각 폰의 버튼(파워, 숫자, 보내기, 종료) 메시지들, (2) 각 폰의 연결 상태를 감시하기 위하여 각 폰에게 주기적으로 PING 메시지를 보내기 위한 타임아웃 메시지들, (3) 이 PING 메시지에 대해 각 폰들이 응답하는 ACK 메시지들, (4) 시스템 관리자가 PBX 시스템의 상태를 모니터하거나 설정을 변경하기 위하여 보내는 메시지들이 있다. 우리는 이러한 외부 입력 메시지(시나리오)들의 우선순위를 다음과 같은 규칙으로 할당하였다.

- 한 셀 폰의 모든 버튼 메시지들은 같은 우선순위를 갖는다.
- 각 폰은 서로 다른 우선순위를 가지며, 각 버튼 메시지는 이를 보내는 폰의 우선순위를 갖는다.
- 시스템 관리자에 의한 메시지는 모든 폰의 버튼 메시지들 보다 낮은 우선 순위를 갖는다.
- PING 메시지를 발생시키기 위한 타임아웃 메시지와 ACK 메시지는 가장 낮은 우선순위를 갖는다.

실험에 사용된 환경은 다음과 같다. 대상 하드웨어는 Sun Microsystems의 Sun Blade 100 이며 사용된 운영체제는



(그림 14) 콜 요청/응답 시나리오의 최대 블로킹 시간

Sun Solaris 9 (SunOS 5.9)이다. 하드웨어에 대한 추상화 계층을 제공하는 라이브러리로는 RoseRT 런타임 시스템 라이브러리 2003.06.00을 사용하였으며 이는 gcc 3.0.1로 컴파일 되었다. 우리는 PBX 시스템과 연결된 셀 폰의 개수를 다섯 개에서 백 개까지 변화시키면서 성능을 측정하였다.

5.2 성능 지표

우리는 다음의 성능 지표를 사용하였다:

- 최대 시나리오 블로킹 시간
- 최대 시나리오 응답 시간
- 여러 시나리오 개수를 가진 시스템들에서 평균화된 시나리오 블로킹 시간들의 표준 편차
- 여러 시나리오 개수를 가진 시스템들에서 미션 크리티컬 시나리오의 응답 시간의 표준 편차
- 유도된 태스크의 개수

처음 두 성능 지표는 코드의 응답성을 결정한다. 시나리오 블로킹 시간은 자신보다 낮거나 같은 우선순위의 시나리오가 수행되는 동안 기다리는 시간이다. 시나리오 응답 시간은 시나리오를 시발시키는 외부 메시지가 인큐된 시점으로부터 액티비티 연속의 맨 마지막 액티비티가 수행될 때까지의 시간이다. 이는 (1) 쓰레드 간 메시지 전달 시간, (2) 쓰레드 간 문맥 전환 시간, (3) 시나리오를 구성하는 각 액티비티들의 수행 시간의 합, (4) 낮은 우선순위의 시나리오에 의한 블로킹 시간, (5) 보다 높은 우선순위의 시나리오에 의하여 선점되어 소비되는 시간으로 구성된다.

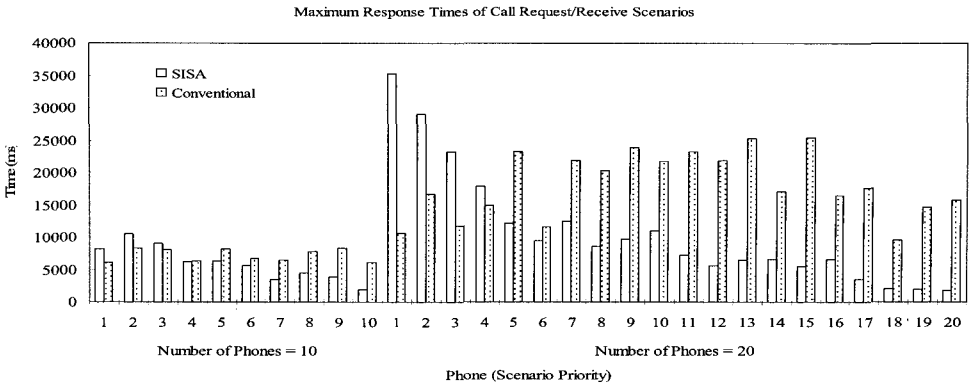
나머지 세 지표는 대규모의 시스템을 지원할 수 있는지를 가름한다. 코드가 대규모의 시스템에서도 올바르게 동작하기 위해서는 모든 시나리오의 블로킹 시간이 시나리오의 개수가 증가함에 따라 심하게 변하지 않아야 한다. 한편 시스템이 커짐에 따라 낮은 우선순위의 시나리오의 응답 시간은 높은 우선순위의 시나리오의 개수가 증가하면서 길어질 수 밖에 없게 된다. 그러나, 시스템이 미션 크리티컬 이벤트에 대해 제 시간에 응답하기 위해서는 미션 크리티컬 시나리오의 응답 시간이 시스템의 규모에 상관없이 거의 일정해야 한다. 이러한 시스템 특성을 정량화하기 위해, 우리는 시스템의 시나리오 개수를 증가시켜가면서 평균화된 블로킹 시간의 표준 편차를 계산한다. 또한 미션 크리티컬 시나리오의 응답 시간에 대해서도 이를 수행한다. 마지막으로, 문맥 전환 부하를 줄이고 메모리를 절약하기 위하여 대규모의 시스템일지라도 태스크의 숫자 역시 가능한 작아야 한다.

5.3 성능 결과

여기에서는 콜 요청/응답 (보내기 버튼) 시나리오들에 대한 결과들을 제시한다. 다른 시나리오들에서도 이와 비슷한 실험 결과를 보였으므로 논문에 모두 제시하지 않는다. 이 시나리오들은 셀 폰의 보내기 버튼이 눌러졌을 때 시발되는 시나리오들이다. 이 절 전체에 걸쳐, 시나리오의 우선순위가 높을수록 더 큰 번호를 매긴다.

1. 블로킹 시간: (그림 14)는 폰의 개수가 열 개와 스무 개인 경우에 대하여 각 폰 별 시나리오의 최대 블로킹 시간을 보여준다. 여기에서 더 큰 숫자가 더 높은 우선순위를 나타낸다. 이 결과는 모든 시나리오에 대하여 SISA에서가 기존 접근법에 비하여 최대 블로킹 시간이 크게 줄어든 것을 보여준다. 구체적으로 스무 개의 폰을 가진 시스템에 대하여, 모든 시나리오의 최대 블로킹 시간이 평균적으로 79.2% 줄었다. 또한 SISA에서는 각 시나리오의 최대 블로킹 시간이 우선순위에 상관없이 거의 일정하게 유지되는 반면, 기존 구현에서는 우선순위가 높을수록 더 커지는 것을 볼 수 있다. 그 이유는 기존 능동 객체 기반 다중 쓰레딩에서는 블로킹이 여러 번 발생할 수 있으며, 우선순위가 높은 시나리오일수록 낮은 우선순위의 시나리오가 더 많아 지기 때문에 블로킹 시간이 더 길어지기 때문이다. 그 결과 미션 크리티컬 시나리오(20번째 폰)는 블로킹 시간이 94.5%나 감소하였다.

2. 응답 시간: (그림 15)는 폰의 개수가 열 개와 스무 개인 경우에 대하여 각 시나리오의 최대 응답 시간을 보여준다. 이는 모든 시나리오의 SISA에서의 응답시간이 전반적으로 기존의 구현에서보다 더 작은 것을 보여준다. 구체적으로 스무 개의 폰을 가진 시스템에 대하여, 모든 시나리오가 평균적으로 응답 시간이 30.3% 정도 줄었다. 또한 시나리오의 우선순위가 높을 수록 최대 응답 시간이 SISA에서가 기



(그림 15) 콜 요청/응답 시나리오의 응답 시간

존에 비하여 작아지는 반면, 낮은 우선순위에 대하여서는 반대의 현상이 나타나는 것을 볼 수 있다. 스무 개의 폰을 가진 시스템에서 중대-임부 시나리오(스무 번째 폰)의 응답 시간은 88.6% 줄어든 반면, 가장 낮은 우선순위의 시나리오(첫 번째 폰)에 대해서는 두 배 이상(230%) 늘어났다. 이 이유는 SISA에서 높은 우선순위의 시나리오가 블로킹을 덜 당하여 더 많이 수행되기 때문에 선점 시간이 그만큼 길어졌기 때문이다. 즉, SISA에서 낮은 우선순위의 시나리오는 높은 우선순위의 시나리오의 응답 시간이 짧아지는 만큼 그 응답 시간이 길어질 수 밖에 없다.

3. 유도된 태스크의 숫자와 시나리오 개수 증가에 따른 블로킹/응답 시간 변화: (그림 14)는 모든 시나리오의 블로킹 시간이 SISA의 코드에서는 거의 일정한 반면 기존 구현에서는 스무 개의 폰을 가진 시스템에서 열 개의 폰을 가진 시스템보다 확연히 큰 것을 보여준다. 또한 (그림 15)에서 미션 크리티컬 시나리오(열 번째 폰과 스무 번째 폰)의 응답 시간을 두 시스템에서 비교하면, SISA에서는 거의 일정한 반면 기존 접근법에서는 스무 개의 폰을 가진 시스템에서 크게 증가하였음을 볼 수 있다. 이러한 현상을 정량화하기 위하여, 다섯 개에서 백 개까지 폰(시나리오)의 개수를 변화시키면서 (1) 시나리오들의 블로킹 시간의 평균값과 (2) 미션 크리티컬 시나리오의 응답 시간의 표준 편차를 <표 1>과 같이 산출하였다. <표 1>의 결과는 SISA에서 쓰레드 개수에 따른 시나리오들의 블로킹 시간의 편차는 97%나 줄었으며, 미션 크리티컬 시나리오의 응답 시간의 편차는 98%나 줄은 것을 보여준다.

〈표 1〉 시나리오 개수 증가에 따른 블로킹/응답 시간의 표준 편차

	SISA	기존 접근법
시나리오의 평균 블로킹 시간	99.78 ms	3324.93 ms
미션 크리티컬 시나리오의 응답 시간	199.18 ms	6084.52 ms

한편, 유도된 태스크의 숫자는 n 개의 폰이 있을 때 다음과 같다.

- 기존의 접근법: $n + 0.5n + 10$ 개의 태스크들. 첫째 n 개의 태스크들은 폰을 추상화하는 능동 객체들을 위한 것이고, 둘째 $0.5n$ 개의 태스크들은 각 콜 연결을 처리하기 위한 능동 객체들이다. 마지막 10 개의 태스크들은 시스템 관리자를 추상화하는 능동 객체, 연결 감시를 위한 능동 객체, 몇몇 컨트롤러나 데이터베이스 관리 객체들을 위한 것들이다.
- SISA: $n + 2$ 개의 태스크들. 첫째 n 개의 태스크들은 폰 요청을 관리하는 시나리오를 위한 것들이고, 둘째 2 개의 태스크는 시스템 관리자의 요청을 처리하는 시나리오와 연결 상태를 감시하기 위한 시나리오를 위한 것들이다.

이와 같이, 기존의 접근법에서 유도된 태스크의 개수는 SISA에서보다 훨씬 컸다. 이러한 결과들은 SISA가 기존 접근법에 비해 규모의 시스템을 더 잘 지원한다는 것을 명백히 보여준다. 게다가, 이 결과들은 또한 시스템의 규모가 커질수록 SISA와 기존 접근법간의 성능 차이가 더 심해지는 것을 보여준다.

6. 관련 연구

능동 객체 기반 태스크 매핑을 사용하여 자동으로 생성된 코드의 성능이 종종 개발자가 예측하기 힘들게 나쁜 현상이 나타나자, 이를 해결하기 위한 노력들이 많이 있어왔다. 우선 능동 객체 기반 태스킹 매핑의 틀 안에서 코드의 성능을 좋게 하기 위하여 능동 객체들을 그룹하기 위한 안내지침들을 제시하는 연구가 있었다[8, 9, 12, 31]. 그러나 1.1절에서 설명하였듯이 이러한 안내지침들은 개발자들이 적용하기 어렵거나, 적용된다고 하더라도 실시간 성능 문제를 해결하는 근본적인 방법이 아니라는 한계점이 있다. 그 밖에, 개발자로 하여금 객체 모델의 스케줄링 가능성을 분석할 수 있도록 도와주어 코드의 성능을 예측할 수 있도록 도와주기 위한 노력들이 있었다. [34]에서는 각 메시지의 처리에 대해 응답 시간을 분석하는 기법을 제시하였다. [11]에서는 이용률 분석을 통해, [31], [33]에서는 응답 시간 분석을 통해 스케줄링 가능성을 분석하는 기법을 제시하였다. 한편 [32], [33]에서는 선점 임계 스케줄링 기법[39]을 적용하여 각 외부 이벤트에 대해 시스템이 응답하기까지의 블로킹을 한번

으로 한정시켜 줄 수 있는 아이디어를 제시하고 있다. 그러나 이 기법은 모든 메시지의 우선순위가 고정되어야 한다는 가정을 기반으로 하고 있다. 즉 메시지를 시발하는 외부 메시지에 따라 동적으로 우선 순위를 바꿀 수 없기 때문에 시나리오 자체가 우선순위를 가질 수 없는 한계점이 있다. 게다가 이들 연구에서는 메시지가 태스크로 어떻게 매핑될 지에 대해서 열린 문제로 남겨두었으며, 연구 결과의 구현과 실험적 검증에 까지 이르지 못하였다.

SISA는 시나리오 기반 설계와 요구 공학으로부터 크게 영향을 받았다. [6]에서 시나리오 기반 설계에 관한 개관을 제시하고 [40]에서 시스템 개발에서 시나리오를 사용하는 현재의 관행에 관한 개관을 제시하였듯이, 객체 지향 설계를 보완하기 위하여 시나리오를 도입한 수많은 연구가 있었다. 대부분의 연구는 [6], [18], [27], [28]에서와 같이 요구 사항과 목표를 모델링하기 위해 시나리오를 사용하거나, [20], [21], [38], [41]에서와 같이 시나리오로부터 행동 모델을 합성하는 것에 초점을 두고 있다. SISA는 각 시나리오가 구현 수준의 태스크로 매핑되어 다중 태스킹 구현을 자동 생성하는 것에 초점을 두고 있다. 이와 같이 이들 연구와 SISA에서 시나리오를 사용하는 목적은 다르지만, 시나리오가 최종단 사용자가 인지할 수 있는 모델링 실체이자 시간 제약과 같은 사용자의 요구 사항이 직접적으로 연관된 것이라는 점에서 출발한다는 데에서 그 동기가 같다.

SISA는 또한 둘 이상의 시나리오들을 관리하는 기법을 제공한 여러 연구로부터 영향을 받았다. 실시간 UML 프로파일[25]과 고수준 MSC(hMSCs) [1], [29]에서는 여러 다른 시나리오들을 연관시켜 하나의 시나리오를 구성하는 방법을 제공하였다. [7]과 [37]는 또한 선조건이나 시발조건을 사용하여 시나리오를 결합하는 기법을 제시하였다. 여기에서는 시나리오들을 서로 연관시키는 대신 각 시나리오가 언제 발생할 수 있는지에 대한 정보가 제공된다. SISA에서는 이러한 연구들과 유사하게 시나리오들을 그룹 지워 태스크로 매핑시킬 수 있는 방법을 제공한다.

아울러 우리의 연구는 다양한 모델 변환 기법에 의하여 영향을 받았다. 많은 연구들이 성능 예측을 위해 모델 변환 기법을 사용한다[4]. 한편 [10], [20], [24], [35]는 보다 나은 맞춤형 출력을 생성하기 위하여 중간 모델을 사용한다. 우리는 시나리오 기반 모델을 같은 목적에서 사용한다. [10], [13], [20], [23]은 또한 메타 모델 변환에 관한 규칙을 명세함으로써 모델 변환기를 자동으로 생성하는 방법을 제안하였다. 이 기술들은 중간 시나리오 기반 모델을 유도하는 데 있어서 우리의 접근법에 통합되어 적용될 수 있다.

7. 결 론

우리는 객체 지향 내장형 시스템을 위해 최적의 응답 시간을 가지는 태스크들의 집합을 유도하는 방법, 이를 지원하는 개발 도구, 런타임 아키텍처를 총칭하는 SISA를 제시하였다. 본 논문이 기여하고 있는 바는 다음의 네 가지로 요약할 수 있다: 첫째, 시나리오 기반 태스크 집합 유도 방법을 제안하였다. 이 방식은 각 시나리오에 대해 최소한의 응답 시간을 가지면서 태스크의 숫자가 최소화된 태스크 집합을 유도한다. 이 방식을 사용하면 태스크간 메시지 전달에 의한 블로킹이 사라지며, 각 시나리오의 블로킹 시간이 보다 낮은 우선순위의 시나리오의 가장 긴 액티비티의 수행 시간보다 작게 된다. 둘째, 우리는 시스템의 객체 지향 모델을 수정하지 않고서 개발자가 시스템의 시간적 특성을 모델링할 수 있게 해주는 개발 도구를 제시하였다. 이는 기존의 개발 도구들에서는 불가능한 것이었다. 셋째, 제안된 방법을 지원하는 런타임 시스템 아키텍처를 제안하였다. 이 아키텍처는 기존의 아키텍처를 수정하거나 몇몇 특성들을 추가함으로써 우리의 방법이 어떻게 지원될 수 있는지를 명백하게 보여준다. 마지막으로, 우리는 SISA를 기존의 산업용 PBX 시스템에 적용하고, 성능 평가 결과를 제시하였다. 이 결과는 SISA가 기존에 알려진 최선의 방법에 비해 미션 크리티컬 시나리오의 블로킹 시간을 94.5%까지 줄임으로써, 결과 시스템의 성능을 크게 향상시키는 것을 명백하게 보여준다. 실험 결과는 또한 SISA가 기존의 접근법에 비해 대규모의 시스템을 보다 더 잘 지원하는 것을 보여준다.

이 연구는 다양한 향후 연구를 필요로 하며, 우리는 이를 후속 연구로 진행 중이다. 첫째, 능동 객체들의 네트워크와 시나리오 집합의 두 모델링 관점의 일관성을 효율적으로 유지시키기 위한 방법을 고려하고 있다. 구체적으로, 원래 능동 객체들로 설계된 시스템이 변경되었을 때, 이 변화를 시나리오 트리에 반영하기 위하여 필요한 추가적인 작업을 최소화하는 방법이 제공되어야 한다. 둘째, 고정 우선순위 스케줄링을 확장하여 고급 실시간 스케줄링 기법을 채용하여 실시간 스케줄링 가능성을 보장하는 코드를 자동으로 생성하는 연구를 진행하고 있다. 이것이 가능해 지면 개발자는 시나리오의 우선순위가 아닌 종료 시한만을 모델링하고, 이를 만족시키는 코드를 자동 생성시킬 수 있다. 셋째, 분산 시스템을 지원하는 연구가 필요하다. 우선 교체 가능한 능동 객체를 재 컴파일 없이 다른 바이너리를 통해 사용할 수 있도록 지원하는 연구를 진행하고 있다. 또한 다중 프로세스로 구성되는 분산 시스템에서 SISA를 효율적으로 지원하는 것에 대한 연구도 진행 중이다. 마지막으로, 자동차 시스템 등 다양한 다른 실시간 응용에 SISA를 적용하여 새로운 문제점을 찾고 이를 해결하는 연구도 진행 중이다.

참 고 문 헌

[1] R. Alur, G.J. Holzmann, and D. Peled, "An Analyser for Message Sequence Charts," Proceedings of Second Int'l Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS '96), pp.35-48, 1996.

[2] ARTiSAN Software Tools Incorporation. Real-Time Studio, http://www.artisansw.com

[3] M. Awad, J. Kuusela, and J. Ziegler, "Object-Oriented Technology for Real-Time Systems : A Practical Approach Using OMT and Fusion," Prentice Hall, 1996.

[4] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni, "Model-Based Performance Prediction in Software Development : A Survey," IEEE Transactions on Software Engineering, pp.295-310, Vol.30, No.5, 2004.
[5] A. Burns and A. Wellings, "Real-Time Systems: Specification, Verification, and Analysis," chapter Advanced Fixed Priority Scheduling, pp.32-65, Prentice Hall, 1996.
[6] J. M. Carroll, R. L. Mack, S. P. Robertson, and M. B. Rosson, "Binding Objects to Scenarios of Use," International Journal of Human-Computers. Stud., Vol.41, No.1/2, pp.243-276, 1994.
[7] J. M. Carroll, Ed., "Scenario-Based Design: Envisioning Work and Technology in System Development," John Wiley and Sons, 1995.
[8] B. P. Douglass, "Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks and Patterns," Addison-Wesley, 1999.
[9] B. P. Douglass, "Real-Time UML: Developing Efficient Objects for Embedded Systems," Addison-Wesley, 1999.
[10] D. Galran, L. Cai, and R. L. Nord, "A Transformational Approach to Generating Application Specific Environments", Proceedings of ACM SIGSOFT Symposium of Software Development Environment, pp.68-77, 1992.
[11] D. Gaudrean and P. Freedman, "Temporal Analysis and Object-Oriented Real-Time Software Development: A Case Study with ROOM/Objectime," Proceedings of IEEE Real-Time Systems Symposium, pp.110-119, 1996.
[12] H. Gomaa, "Designing Concurrent, Distributed, and Real- Time Applications with UML," Addison-Wesley Longman, 2000.
[13] G. Gullekson and B. Selic, Design Patterns for Real-Time Software, Proceedings of Embedded Systems Conference West, 1996.
[14] W. Ho, J. Jézéquel, A. Guennec, and F. Pennaneac'h, "UMLAUT : An Extendible UML Transformation Framework," Proceedings of Automated Software Engineering (ASE'99), pp.275-278, 1999.
[15] IAR Systems Incorporation, visualSTATE, www.iar.com
[16] IBM Rational Software Corporation, "Rational Rose Real Time User Guide: Revision 2001.03.00," 2000.
[17] I-Logix Incorporation. Rhapsody tools. http://www.ilogix.com
[18] Institute for Electrical and Electronic Engineers and The Open Group, Base Specifications Issue 6, IEEE Std. 1003.1 (POSIX), 2004 Edition, System Interfaces volume, 2004.
[19] H. Kaindl, "A Design Process Based on a Model Combining Scenarios with Goals and Functions," IEEE Transactions on Systems, Man, and Cybernetics-Part A : Systems and Humans, Vol.30, No.5, pp.537-551, 2000.
[20] G. Karsai, J. Sztipanovits, and H. Franke, "Towards Specification of Program Synthesis in Model-Integrated Computing," Proceedings of IEEE ECBS Conference, pp. 226-233, 1998.
[21] I. Kru¨ ger, R. Grosu, P. Scholz, and M. Broy, "From MSCs to Statecharts," Distributed and Parallel Embedded Systems, F.J. Rammig, ed., Kluwer Academic Publishers, pp.61-71, 1999.
[22] A. Lamsweerde and L. Willemet, "Inferring Declarative Requirements Specifications from Operational Scenarios," IEEE Transactions on Software Engineering, Vol.24 , No.4, pp.1089-1114, 1998.
[23] D. Milicev, "Automatic Model Transformations Using Extended UML Object Diagrams in Modeling Environments," IEEE Transaction on Software Engineering, Vol.28, No.4, pp.413-431, 2002.
[24] J. Mukerji and J. Miller, "Model Driven Architecture (MDA) Guide Version 1.0.1" OMG Document Number : omg/2003-06-01, 2003.
[25] Object Management Group, "UML Profile for Schedulability, Performance, and Time," OMG Document ptc/2003-09-01, http://www.omg.org/cgi-bin/doc?ptc/2002-09-03, 2003.
[26] Object Management Group. Unified Modeling Language (UML), version 2.0 (under finalization), OMG Documentation, http://www.omg.org/technology/documents/modeling_spec_catalog.htm#uml, 2003.
[27] C. Potts, "Using Schematic Scenarios to Understand User Needs," Proceedings of Symposium on Designing Interactive Systems: Processes, Practices, Methods, and Techniques (DIS '95). MI : ACM, pp.247-256, August, 1995.
[28] C. Rolland, C. Souveyet, and C. Ben Achour, "Guiding Goal Modeling Using Scenarios," IEEE Transactions on Software Engineering, Vol.24, pp.1055-1071, 1998.
[29] E. Rudolph, P. Graubmann, and J. Grabowski, "Tutorial on Message Sequence Charts '96," Proceedings of IFIP TC6 WG6.1 Int'l Conf. Formal Description Techniques (FORTE '96), pp.1629-1641, 1996.
[30] B. Selic, G. Gullekson, and P. T. Ward, "Real-Time Object-Oriented Modeling," John Wesley and Sons, 1994.
[31] M. Saksena, P. Freeman, and P. Rodziewicz, "Guidelines for Automated Implementation of Executable Object Oriented Models for Real-Time Embedded Control Systems," Proceedings of IEEE Real-Time Systems Symposium, pp.240-251, 1997.
[32] M. Saksena, P. Karvelas, and Y. Wang. "Automatic Synthesis of Multi-tasking Implementations from Real-Time Object-Oriented Models," Proceedings of IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, pp.360-367, 2000.
[33] M. Saksena and P. Karvelas. "Designing for Schedulability: Integrating Schedulability Analysis with Object-Oriented Design," Proceedings of Euromicro Conference on Real-Time Systems, pp.101-108, 2000.
[34] M. Saksena, A. Ptak, P. Freedman, and P. Rodziewicz. "Schedulability Analysis for Automated Implementations of Real-Time Object-Oriented Models," Proceedings of IEEE Real-Time Systems Symposium, pp.92-102, 1998.
[35] C. Simonyi, "Intentional Programming-Innovation in the Legacy Age," International Federation for Information Processing Work Group 2.1, http://research.microsoft.com/ip, June, 1996.
[36] Telelogic Corporation. TAU, http://www.telelogic.com
[37] P.P. Texel and C.B. Williams, "Use Cases Combined with Booch, OMT, and UML" Prentice-Hall, 1997.
[38] S. Uchitel, J. Kramer, and J. Magee, "Synthesis of Behavioral Models from Scenarios," IEEE Transactions on Software Engineering, Vol.29, No.2, pp.99-115, 2003.
[39] Y. Wang and M. Saksena, "Scheduling Fixed Priority Tasks with Preemption Threshold," Proceedings of IEEE Real-Time Computing Systems and Applications Symposium, pp. 328-335, 1999.
[40] K.Weidenhaupt, K. Pohl, M. Jarke, and P. Haumer, "Scenarios in System Development : Current Practice," IEEE Software, pp.33-45, 1998.
[41] J. Whittle and J. Schumann, "Generating Statechart Designs

from Scenarios," Proceedings of International Conference on Software Engineering (ICSE '00), pp.314-323, 2000.
[42] J. Zalewski, Real-Time Software Architectures and Design Patterns: Fundamental Concepts and Their Consequences - 키note Address, Proceedings of 24th IFAC/IFIP Workshop on Real-Time Programming, 1999.

부록. 시나리오 트리 생성 알고리즘

시나리오 트리를 생성하는 알고리즘은 (그림 16)과 같다. 이 알고리즘은 (1) SPP로 도착하는 외부 메시지로부터 각 시나리오 트리를 생성하고 (Total-Trees-Build()), (2) 모든 가능한 메시지 생성들을 고려하면서 메시지 흐름의 경로를 추적하는 것으로 이루어진다 (Tree-Build-From-Message/Activity/Junction-Node()). 여기에서 가능한 메시지 흐름의 경로를 추적하는 것은 Tree-Build-From-Activity-Node()와 Tree-Build-From-Junction-Node()에서 한 액티비티에서 보내지는 메시지들의 정규 표현을 유도함으로써 이루어진다.

김 세 화

e-mail : ksaehwa@redwood.snu.ac.kr
1997년 서울대학교 전기공학부(학사)
1997년~1998년 서울대학교 자동화시스템 공동연구소(연구원)
2000년 서울대학교 전기컴퓨터공학부 (공학석사)
2002년 서울대학교 전기컴퓨터공학부 박사과정 수료
2000년~현재 서울대학교 전기컴퓨터공학부 박사과정
관심분야 : 내장형 시스템 소프트웨어, 실시간 운영체제, 내장형 미들웨어, 객체지향 설계방법론

박 지 용

e-mail : parkjy@redwood.snu.ac.kr
2002년 서울대학교 전기공학부(학사)
2005년 서울대학교 전기컴퓨터공학부 석박사통합과정 수료
2005년~현재 서울대학교 전기컴퓨터공학부 박사과정
관심분야 : 실시간 운영체제, 내장형 시스템, Aspect 지향 프로그래밍, 재구성가능 운영체제

홍 성 수

e-mail : sshong@redwood.snu.ac.kr
1986년 서울대학교 컴퓨터공학과(학사)
1988년 서울대학교 컴퓨터공학과(공학석사)
1994년 University of Maryland. Department of Computer Science (공학박사)
1988년~1989년 한국전자통신 연구소 (연구원)
1994년~1995년 University of Maryland. Department of Computer Science(Faculty Research Associate)
1995년~1995년 Silicon Graphics Inc.(Member of Technical Staff)
1995년~1997년 서울대학교 전기공학부 전임강사
1997년~2001년 서울대학교 전기공학부 조교수
2001년~현재 서울대학교 전기컴퓨터공학부 부교수
관심분야 : 실시간 운영체제, 내장형 시스템 개발 방법론, 내장형 미들웨어

```
Total-Trees-Build(M)                ▷ M is the input model.
▷ Trees is the set of all scenario trees
1  Trees ← {}
2  for each p ∈ SPPs[M]
3     do for each m ∈ IncomingMessages[p]
4        do create tree T
7           create message node n with message m
8           root[T] ← n
9           Tree-Build-From-Message-Node(n)
10          Trees ← Trees ∪ {T}

Tree-Build-From-Message-Node(n)
1  A ← MatchesActivities[n]
2  if |A| = 1
3     then create activity node n' with activity a ∈ A
4        child[n] ← n'
5        Tree-Build-From-Activity-Node(n')
6  else create disjunction node n'
7     child[n] ← n'
8     for each a ∈ A
9        do create activity node n'' with activity a
10          child[n'] ← n''
11          Tree-Build-From-Activity-Node(n'')

Tree-Build-From-Activity-Node(n) .
1  P ← Regular-Expression-For-Messages-Sent-By(activity[n])
2  P' ← Components-Without-Outermost-Junctions(P)
3  if|P'| = 1              ▷ only one message is sent out unconditionally.
4     then create message node n' with message m ∈ P'
5        child[n] ← n'
6        Tree-Build-From-Message-Node(n')
7  elseif |P'| ≠ 0 ▷ one or more messages are sent out conditionally.
8     then create junction node n' of type Outermost-Junction-Type(P)
9        child[n] ← n'
10       for each p ∈ P'
11          doTree-Build-From-Junction-Node(n', p)

▷ P is a regular expression for message sequence.
Tree-Build-From-Junction-Node(n, P)
1  P' ← Components-Without-Outermost-Junctions(P)
2  if|P'| = 0              ▷ no message is sent out.
3     then create flow final node n'
4        child[n] ← n'
5  elseif |P'| = 1 ▷ only one message is sent out unconditionally.
6     then create message node n' with message m ∈ P'
7        child[n] ← n'
8        Tree-Build-From-Message-Node(n')
9  else    ▷ One or more messages are sent out conditionally.
10    then create junction node n' of type Outermost-Junction-Type(P)
11    child[n] ← n'
12       for each p ∈ P'
13          doTree-Build-From-Junction-Node(n', p)
```

(그림 16) 시나리오 트리 생성 알고리즘