변경가능성과 상호운영성을 고려한 소프트웨어 기반 시뮬레이터 아키텍처 패턴의 정의

(Defining an Architectural Pattern for the Software Based Simulators in Consideration of Modifiability and Interoperability)

국 승 학 † 김 현 수 †† 이 상 욱 †††
(Seung Hak Kuk) (Hyeon Soo Kim) (Sanguk Lee)

요 약 시뮬레이션은 컴퓨터를 이용하여 실제 사물이나 작업의 상태, 혹은 프로세스를 모방하여 그 특징을 찾아내는 작업을 지칭하며, 시뮬레이터는 이러한 시뮬레이션 작업을 수행하는 하드웨어/소프트웨어 도구를 말한다. 다양한 시뮬레이터의 개발에 있어 공통적으로 요구되는 비기능적 속성은 변경가능성, 상호운영성, 확장성이다. 그러나 기존의 시뮬레이터 개발에 관한 연구는 관심 시뮬레이션 모델에 대한 개발에 관한 것이며, 이러한 비기능적 요구사항에 대한 관심이 적다. 이에 본 논문에서는 소프트웨어 기반 시뮬레이터 개발에 있어 요구되는 비기능적 요구사항 중 변경가능성, 상호운영성, 확장성을 고려한 시뮬레이터 아키텍처 패턴을 제시한다. 본 논문에서는 아키텍처 패턴을 정의하기 위해 시뮬레이터의 필수 요소를 파악하고 그들 간의 관계를 정의하였으며, 비기능적 요구사항을 반영할 수 있는 구조로 설계하였다. 제시된 패턴은 다양한 시뮬레이션 모델을 구축할 수 있도록 시뮬레이션 모델 컴포넌트를 중심으로 이들의 조합을 통해 문제를 해결할 수 있다. 이는 시뮬레이션 모델의 재구축을 통해 유연하게 시스템의 변경가능성을 보장하며, 시뮬레이션 모델에 다양한 인터페이스를 추가할 수 있고, 시뮬레이션 모델 컴포넌트의 인터페이스를 통일시켜 상호운영성 및 확장성을 보장한다. 이 논문의 아키텍처 패턴은 향후 개발될 다양한 소프트웨어 기반 시뮬레이터의 참조 모델로 활용될 수 있다.

키워드 : 소프트웨어 기반 시뮬레이터, 소프트웨어 아키텍처, 아키텍처 패턴

Abstract Simulation is the imitation of some real thing, state of affairs, or process. The act of simulating something generally entails identifying certain key characteristics or behaviors of a selected physical or abstract system. And a simulator is the software or hardware tool that performs simulation tasks. When developing a simulator, the non-functional requirements such as modifiability, interoperability, and extendability should be required. However, existing studies about the simulator development focus not on such non-functional requirements but on the methodologies to build the simulation model. In this paper, we suggest the new architectural pattern for the software based simulator in consideration of such non-functional requirements. In order to define the architectural pattern, we identify the essential elements of the simulators, define relationships between them, and design the architectural structure with the elements to accommodate such non-functional requirements. According to the proposed pattern we can solve the simulation problems to combine the various simulation model components. The pattern guarantees modifiability by reconstructing the simulation model, also

· 본 연구는 지식경제부 및 산업기술연구회의 협동연구과제의 일환으로 수행하였음{08AR2310, GPS/Galileo환경에서의 위성항법신호생성/수신처리 및 측위성능향상 기초연구}

† 정 회 원 : 충남대학교 컴퓨터공학과
triple888@cnu.ac.kr
†† 종신회원 : 충남대학교 전기정보통신공학부 교수
hskim401@cnu.ac.kr
(Corresponding author임)
††† 정 회 원 : 한국전자통신연구원 위성관제항법 연구팀 책임연구원
slee@etri.re.kr

논문접수 : 2009년 4월 28일
심사완료 : 2009년 6월 10일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제15권 제8호(2009.8)

guarantees interoperability and extendability by adding various interfaces to the simulation model and by keeping the consistent interfacing mechanism between the simulation model components. The suggested architectural pattern can be used as the reference architecture of the simulator systems that will be developed in future.

Key words : Software based simulator, Software Architecture, Architectural pattern

1. 서 론

시뮬레이션이란 복잡한 문제를 해석하기 위하여 모델에 의한 실험, 또는 사회현상 등을 해결하는 데에 있어 실제와 비슷한 상태를 수식 등의 모델로 만들어 모의적으로 연산을 되풀이하여 그 특성을 파악하는 작업으로 정의할 수 있다[1]. 즉 실제 또는 가상의 동적 시스템 모형 또는 모델을 컴퓨터를 이용하여 연구하는 것을 말하며 모의실험 또는 모사라고 한다. 이러한 시뮬레이션은 실제 시스템의 동작에 대한 정보를 획득하거나 시스템의 성능 향상을 위한 운영, 자원 관리 방법의 개발, 새로운 개념 및 시스템의 구현 전 테스트, 분산 시스템의 구축 전 동작에 대한 정보 획득을 위해 널리 사용된다[1]. 이러한 시뮬레이션을 이용하면 실제 시스템에 대한 요구사항을 분석하는데 도움을 주며, 제시한 모델을 쉽게 검증할 수 있다. 무엇보다도 시뮬레이션의 가장 큰 장점은 실제 시스템을 구축하는 것 보다 시간과 비용을 절약할 수 있다는 것이다[1]. 예를 들어, 제품의 생산라인 시스템 구축 작업을 생각해보면, 생산라인에 필요한 다양한 컴포넌트의 구입 및 조립, 생산 프로세스의 정의, 다양한 스케줄링 방법에 대한 정의가 필요하다[2]. 이러한 시스템을 구축할 경우 한 번에 모든 요구사항을 만족할 수 있는 시스템을 구축하는 것은 불가능하며, 그 기간 또한 최소 몇 달에서 몇 년의 기간이 소요된다. 또한 구축 후 시스템의 병목 현상과 같은 문제가 발생할 경우 원인을 분석하고 해결방안을 찾는 시간 또한 엄청날 것이다. 그러나 시뮬레이션 작업을 통해 초기에 요구사항 및 고려사항을 파악하고 문제점을 분석한 후 시스템을 구축한다면 보다 강건한 시스템을 구축할 수 있을 것이다.

시뮬레이션 기능을 수행하는 하드웨어/소프트웨어 시스템을 시뮬레이터라고 하며, 동작 방법, 사용 목적에 따라 다양한 시뮬레이터로 분류할 수 있다[3]. 시뮬레이터는 크게 범용 시뮬레이션 프레임워크 기반의 시뮬레이터와 특정 도메인을 위해 독립적으로 개발된 시뮬레이터로 분류된다. 범용 시뮬레이션 시스템은 일종의 시뮬레이션 프레임워크라고 볼 수 있다. AnyLogic[4], ARENA[5], BuildSim[6], Extendsim[7], Flexsim[8], SimApp[9]등과 같은 프레임워크들이 있다. 이러한 도구들은 시뮬레이션 모델링 기능 및 시뮬레이션 모델의 동작 환경을 제공해주기 때문에 다양한 분야의 시뮬레이션을 수행할 수 있다. 특정 도메인을 위한 시뮬레이터는 해당 도메인의 특성 및 그 사용 목적에 따라 개발된 시뮬레이터로 비행 시뮬레이터, 교통 흐름 시뮬레이터, 네트워크 공격 및 탐지 시뮬레이터, 위성 항법 시스템 시뮬레이터 등이 있다.

특정 도메인을 위한 시뮬레이터는 범용 시뮬레이션 시스템 상에 구현될 수 있으며, 이러한 경우 범용 시뮬레이션 시스템 상에서 동작하는 시뮬레이션 모델 혹은 알고리즘으로 표현된다. 그러나 범용 시뮬레이션 시스템은 개발하고자 하는 시뮬레이터에서 요구하는 모든 기능을 제공할 수는 없다. 예를 들어 개발하고자 하는 시뮬레이터가 입력을 데이터 파일, 외부 시스템과의 상호작용, 웹 등으로부터 모두 제공받을 수 있어야 한다면, 이는 결국 시뮬레이션 모델에서 처리해야 한다. 그렇지만 범용 시뮬레이션 시스템을 이용하지 않고 시뮬레이터를 구축할 경우 시뮬레이션 모델 및 알고리즘의 구현과 범용 시뮬레이션 시스템에서 제공하는 일반적인 기능도 구현해야한다. 즉 개발될 시뮬레이터가 일반적인 시뮬레이션 시스템에서 요구하는 기능, 비기능적 요구사항을 달성할 수 있는 구조를 갖고 있어야 함을 의미한다. 그러나 대부분의 시뮬레이터 개발에 대한 연구는 범용 시뮬레이션 시스템을 이용한 개발 방법에 관한 것이며, 범용 시뮬레이션 시스템을 이용하지 않는 경우를 위한 고려사항이나 필수 기능 및 비기능적 요구사항에 대한 연구는 거의 없으며, 참조할만한 아키텍처 패턴도 존재하지 않는다. 이에 본 논문에서는 범용 시뮬레이션 시스템을 이용하지 않고 시뮬레이터를 개발해야하는 경우 활용할 수 있는 아키텍처 패턴을 정의한다. 시뮬레이터를 위한 참조 아키텍처를 정의하기 위해 우리는 상향식 접근 방법을 채택한다. 즉 기존의 여러 유형의 시뮬레이터들을 분석하고, 이 과정에서 시뮬레이터들이 반드시 가져야 할 공통적인 기능 요소들을 파악한다. 또한 시뮬레이터 시스템에서 요구되는 비기능적 특성들도 파악한다. 이렇게 파악된 기능 요소들과 비기능적 특성들을 바탕으로 시뮬레이터를 위한 아키텍처 패턴을 정의한다.

시뮬레이터 시스템은 시뮬레이션 기능을 구현한 시뮬레이션 모델과 이들이 동작하기 위한 시뮬레이션 환경에 대한 기능적인 요구사항이 존재한다. 또한 개발 과정에서 비기능적 요구사항으로 변경가능성, 상호운영성,

확장성, 그리고 성능에 대한 요구사항이 존재한다. 시뮬레이터는 다양한 시뮬레이션 모델의 구축 및 검증 작업에 있어 시뮬레이션 모델 자체의 변경, 특정 알고리즘의 교체, 시뮬레이션 과정에서의 입출력 데이터의 변경과 같은 수정에 대한 요구사항이 빈번하게 발생한다. 또한 개발된 시뮬레이터 시스템은 외부의 소프트웨어 혹은 하드웨어 시스템과 연동을 고려하여 개발한다. 그리고 실제 시스템과 유사한 성능 및 실제 시스템의 규모를 고려한 확장성 또한 요구되는 주요 비기능적 요구사항이다. 우리는 이러한 변경가능성, 상호운영성 및 확장성과 관련된 비기능적 요구사항을 중심으로 소프트웨어 기반 시뮬레이터의 아키텍처 패턴을 정의하고, 이를 현재 개발 중인 한국전자통신연구원의 소프트웨어 기반 위성항법시뮬레이터[10]에 적용한다.

이 논문은 다음과 같이 전개된다. 2장에서 여러 유형의 시뮬레이터를 분석하여 기능 요소를 도출하는 부분과 비기능적 특성을 파악하기 위한 내용이 기술된다. 3장에서 시뮬레이터를 위한 아키텍처 패턴을 정의하고, 정의된 아키텍처 패턴의 구성 요소에 대하여 설명하고, 제안된 아키텍처 패턴이 시뮬레이터에서 요구되는 비기능적 특성을 어떻게 만족하는가에 대하여 기술한다. 4장에서 적용사례로서, 제안된 아키텍처 패턴을 바탕으로 소프트웨어 기반 위성항법 시뮬레이터의 아키텍처를 설계한다. 아키텍처 패턴의 구성요소가 실제 시스템의 아키텍처에 적용될 때 어떤 형상으로 나타나는지를 볼 수 있다. 계속해서 아키텍처 패턴의 적용 효과를 살펴보기 위해 기존 위성항법 시뮬레이터의 아키텍처와 아키텍처 패턴을 적용한 시뮬레이터의 아키텍처를 비교하면서 기존 아키텍처에서 달성하기 어려웠던 품질 특성이 새로운 아키텍처에서 달성될 수 있음을 밝힌다. 5장에서 기존 여러 유형의 아키텍처 패턴 중 이 논문에서 제시하는 패턴과 유사한 패턴을 대상으로 시뮬레이터를 위한 아키텍처 패턴으로 활용 가능성 측면에서 비교하여 제안된 아키텍처 패턴이 시뮬레이터를 위한 패턴으로서 타당함을 보인다.

2. 소프트웨어 기반 시뮬레이터 아키텍처 요구사항 파악

2.1 소프트웨어 기반 시뮬레이터의 기능적 요구사항

본 논문에서는 소프트웨어 기반 시뮬레이터의 기능적 요구사항을 도출하기 위해 다양한 시뮬레이터로부터 요구되는 공통의 기능을 추출하였다. 일반적인 소프트웨어 기반 시뮬레이터는 시뮬레이션 기능을 표현하는 시뮬레이션 모델과 시뮬레이션 모델의 수행을 위한 시뮬레이션 환경에 대한 요구사항으로 구분할 수 있다. 이렇게 추출된 공통적인 기능은 본 논문에서 제시하는 시뮬레이션 아키텍처 패턴에서 일반화된 형태로 정의되고, 향후 개발될 다양한 시뮬레이터들의 참조 아키텍처(Reference Architecture)로 활용될 수 있다.

2.1.1 시뮬레이션 모델(Simulation Model)

시뮬레이션 모델은 시뮬레이션 대상이 되는 실제 개념 또는 사물의 정적/동적 특징을 모사하는 수식, 알고리즘 혹은 프로그램이다[1]. 간단한 시뮬레이터의 경우 단일 모델을 이용해 원하는 시뮬레이터를 구축할 수 있으나 일반적으로 다양한 시뮬레이션 알고리즘 및 로직의 조합/연동을 통해 그 기능을 구축한다. 예를 들어 교통흐름 시뮬레이터는 도로 환경에 관한 로직을 구현하는 모듈, 자동차의 행위를 구현한 모듈, 신호 체계를 구현한 모듈 등 다양한 시뮬레이션 모듈들의 조합을 통해 전체 시뮬레이션 모델을 완성한다[11]. 이러한 시뮬레이션 모델은 시뮬레이터의 의도에 따라 그 구조가 결정된다. 시뮬레이션 모델을 정의하기 위한 기능적 요구사항은 다음과 같다.

- 시뮬레이션 모델: 시뮬레이션 모델은 시뮬레이터의 가장 핵심적인 부분으로 시뮬레이션 목적을 달성하기위한 핵심적인 알고리즘을 포함한다. 예를 들어, 네트워크 공격 및 탐지 시뮬레이터[12]에서는 공격 생성 알고리즘 및 탐지 알고리즘, 네트워크 상태 모델 등과 같은 다양한 시뮬레이션 알고리즘의 조합을 통해 시뮬레이터의 기능이 완성된다. 이처럼 시뮬레이터는 다양한 시뮬레이션 알고리즘 및 로직의 조합을 통해 원하는 모델을 구축한다.
- 시뮬레이션 모델 정의 및 관리: 시뮬레이션 모델은 새로운 알고리즘의 추가, 기존 알고리즘의 변경, 모델을 구성하는 컴포넌트의 교체 등에 따라 다른 내용으로 변경되며, 이와 같이 시뮬레이션 모델을 정의하거나 관리할 수 있는 기능이 요구된다. 예를 들어, 네트워크 공격 및 탐지 시뮬레이터[12]에서는 공격 생성 알고리즘, 탐지 알고리즘, 네트워크 상태 모델 등과 같은 다양한 시뮬레이션 요소들을 정의하고 그들의 조합을 통해 시뮬레이션 모델을 정의한다. 마찬가지로 항공기 시뮬레이터[13]은 항공기의 엔진 및 프로펠러의 동적 모델 알고리즘, 공력 데이터 계산 알고리즘, 대기 모델과 같은 다양한 시뮬레이션 알고리즘을 정의하고 그들의 조합을 통해 시뮬레이션 모델을 구축한다. 제품 생산 라인 시뮬레이터[2]의 경우에도 시뮬레이션 프로세스를 정의하는 작업이 요구된다. 이는 각 단위 작업에서의 입출력 정보를 기반으로 데이터의 흐름을 정의하는 작업을 의미한다. 이때 개별 컴포넌트(시뮬레이션 알고리즘 모듈)를 어떤 식으로 조합하느냐에 따라서 각 컴포넌트의 입출력 데이터, 연동 방식 등이 변경된다. 따라서 시뮬레이터 시스템에서는

이러한 시뮬레이션 알고리즘의 상호작용에 대한 관리가 쉽게 수행될 수 있어야 한다. 시뮬레이션 작업에 참여하는 시뮬레이션 알고리즘 모듈들 사이의 상호작용에 대한 메커니즘에 대한 정의가 필요하며, 시뮬레이션 모델의 정의 및 관리에는 이러한 시뮬레이션 알고리즘 모듈들 간의 상호 작용에 대한 정의 및 관리를 포함한다.

- 시뮬레이션 알고리즘 모듈 관리: 대부분의 복잡한 시뮬레이터는 단일 시뮬레이션 알고리즘으로 구현되지 않고, 다양한 시뮬레이션 알고리즘의 조합/연동을 통해 구축된다. 따라서 시뮬레이션 작업에 참여하는 다양한 시뮬레이션 모듈들을 관리하는 기능이 요구된다. [13]의 항공기 시뮬레이션 모델을 구성하는 다양한 시뮬레이션 알고리즘은 수시로 추가/제거 될 수 있고, 이들의 조합을 변경하여 새로운 시뮬레이션 모델을 구축하기도 한다. 특히 동적인 시뮬레이션 모델의 변경을 요구하는 시뮬레이터 시스템일수록 이러한 시뮬레이션 알고리즘 모듈 관리 기능이 필요하다.
- 시뮬레이션 작업 수행 및 통제: 시뮬레이션 작업은 시뮬레이터의 시뮬레이션 모델에 의해 수행된다. 그러나 시뮬레이터에는 시뮬레이션 모델의 작업 수행을 제어하고 통제하는 기능이 포함되어야 한다. 이는 시뮬레이션 모델의 실행, 중단, 재개, 종료와 같은 기본적인 기능뿐만 아니라 사용자와의 상호작용 기능까지 포함한다. 또한 시뮬레이션 모델은 스케줄러와 같은 다양한 서비스와의 연동을 통해 그 작업이 수행되는데 이를 제어하고 관리하여 원활한 작업의 수행을 제어 통제하는 기능 또한 필요하다.

2.1.2 시뮬레이션 환경(Simulation Environment)

시뮬레이션 환경은 시뮬레이션 모델을 수행하기 위한 기본적인 기능을 제공하는 환경이다. 시뮬레이션 환경에서는 다양한 시뮬레이션 작업과 관련된 서비스를 제공함으로써 시뮬레이션 모델이 동작할 수 있는 환경을 제공한다. 시뮬레이션 환경과 관련된 기능 요구사항은 다음과 같다.

- 시뮬레이션 동작 환경 제공: 시뮬레이션 모델이 동작할 수 있는 기본적인 환경을 의미한다. 일반적으로 범용 시뮬레이션 시스템은 이러한 시뮬레이션 동작 환경을 제공한다.
- 시뮬레이션 기록: 시뮬레이션 동작을 기록하기 위한 로깅 기능이 요구된다.
- 시뮬레이션 스케줄링: 시뮬레이션 모델의 동작을 제어하기 위한 시뮬레이션 스케줄링 기능이 요구된다.
- 시뮬레이션 타이밍 제어: 스케줄링과 관련된 시간을 제어하기위한 타이밍 기능이 요구된다.
- 시뮬레이션 이벤트 관리: 전체 시뮬레이션 동작을 제어하기 위한 이벤트 관리 기능이 요구된다.

2.2 소프트웨어 기반 시뮬레이터의 비기능적 요구사항

소프트웨어 기반 시뮬레이터 개발 시 가장 빈번하게 발생하는 요구사항은 변경가능성, 상호운영성, 확장성, 그리고 성능이다. 본 논문에서는 이러한 요구사항 중 변경가능성, 상호운영성, 확장성을 고려하여 아키텍처 패턴을 정의한다. 성능 요구사항의 경우 일반적으로 시뮬레이션 모델이나 알고리즘의 개선 및 자원 관리 방법에 의해 해결되므로 즉, 시뮬레이션 모델에서 그것을 수용할 수 있기 때문에, 이 논문에서는 성능을 제외한 세 가지 요구사항에 초점을 맞춰 아키텍처 패턴을 정의한다.

2.2.1 변경가능성(modifiability)

변경가능성은 시스템 혹은 소프트웨어에 어떠한 수정을 가해야할 때 얼마나 쉽게 이를 반영할 수 있는지에 대한 요구사항이다. 소프트웨어 기반 시뮬레이터의 경우 시뮬레이션 모델, 외부 시스템과의 인터페이싱, 입력 데이터의 변경에 의한 내부 시스템의 변화가 빈번하게 발생하며, 이를 쉽게 수용할 수 있어야 함을 의미한다. 예를 들어 프로세스 중심의 시뮬레이터인 생산라인 시뮬레이터[2]의 경우 생산 공정의 다양성에 따른 프로세스의 변경이 빈번하게 발생한다. 제품 개발에 있어서의 특정 작업의 추가/삭제, 작업을 수행하는 컴포넌트의 변경 등은 시뮬레이션 모델 자체의 변경에 대한 요구사항이다.

인터페이스 변경의 경우 상호운영성과 연계하여 고려해야할 요소로, 외부의 다양한 시스템 혹은 내부의 시뮬레이션 알고리즘의 변경이 쉽게 이루어져야 함을 의미한다. 예를 들어 생산라인 시뮬레이터[2]의 경우 외부의 교통 흐름 시뮬레이터[11]와의 연동을 통해 물류 이동 및 재고 관리에 대한 시뮬레이션 기능까지 확장한다고 가정하면, 재료 수급 및 물류의 유통 과정의 요인에 교통흐름 정보가 반영되어야 한다. 이 경우 외부의 교통흐름 시뮬레이터와의 연동 문제로 인한 인터페이스의 추가, 변경에 대한 요구사항이 발생한다. 또한 내부의 생산 프로세스에 제품 검증 기능을 추가하는 것을 가정하면, 제품 생산 종료 후 검증을 통해 제품의 폐기 혹은 수리 과정이 추가되면서, 내부 컴포넌트들 사이에 인터페이스의 변경을 요구한다.

입력 데이터의 변경에 대한 요구사항은 시뮬레이션 환경 변수, 시뮬레이션 데이터, 시뮬레이션 모델 및 내부 알고리즘 변경에 따른 입출력 데이터의 변경 등이 있다. 이 중 시뮬레이션 데이터나 입출력 데이터의 경우 인터페이스의 변경에 영향을 준다. 앞서 설명한 것과 같이 시뮬레이션 데이터의 획득 경로, 모델 변경 및 알고리즘 교체에 따른 입출력 데이터 변경은 항상 인터페이스의 변경을 동반하기 때문에 이러한 변경 요구사항은 인터페이스의 변경에 포함될 수 있다. 그러나 시뮬레이

션의 환경 변수는 시뮬레이터 자체의 동작에 영향을 준다. 예를 들어 생산라인 시뮬레이터의 경우 환경 변수로 참여하는 직원의 수, 생산 라인을 가동하는 날의 수 등이 전체 시뮬레이션의 동작에 영향을 미치며, 이러한 환경 변수에 의해 동적인 변경이 가능해야 한다.

2.2.2 상호운영성(interoperability)

상호운영성 요구사항의 경우 앞서 설명한 것과 같이 해당 시뮬레이터와 외부의 시스템과의 상호 연동, 내부 시뮬레이션 알고리즘(모듈)들 사이의 연동에 대한 두 가지로 나뉠 수 있다.

외부 시스템과의 연동의 경우 입력으로 주어지는 데이터를 처리하기 위한 인터페이스의 변경과 외부 시스템으로의 시뮬레이션 결과의 출력과 관련된 인터페이스의 변경에 대한 요구사항이 존재한다. 이러한 외부 시스템과의 상호운영성을 보장하기 위해서는 각 시뮬레이터 시스템 혹은 외부 시스템 사이에의 데이터 교환 방법을 통일하는 방법을 이용한다. 예를 들어 최근 시뮬레이터 시스템간의 통합 및 상호운영성을 위한 HLA(High Level Architecture)-RTI (RunTime Infrastructure)[15]는 서로 다른 조직에서 개발한 다양한 시뮬레이터 시스템을 하나의 시뮬레이터 시스템으로 통합하는 방법을 제공한다. HLA 기반 시뮬레이터 시스템에서는 개별 시뮬레이터 시스템을 페더레이트(Federate)로 모델링하고, 이들을 통합해 페더레이션(Federation)을 구축한다. 즉 페더레이션은 통합 시뮬레이터 시스템이다. 여기서 각 페더레이트간 상호운영성과 관련된 기능은 RTI에 구현된다. 이러한 RTI는 개별 페더레이트를 관리하고, 페더레이트간 데이터 교환 메커니즘을 제공한다[15]. 그러나 HLA 기반 시뮬레이터 시스템의 경우 모든 시뮬레이터 시스템이 페더레이트로 구현되어야 하며, RTI가 구축된 환경에서만 상호운영성을 보장할 수 있다는 한계가 있다.

내부 시뮬레이션 모듈의 상호운영성 보장은 개별 시뮬레이션 알고리즘 모듈이 교체, 변경되더라도 이를 쉽게 수용할 수 있어야함을 의미한다. 수시로 변경 및 교체 요구사항이 존재하는 시뮬레이터 시스템의 경우 내부의 알고리즘 모듈을 쉽게 교체/변경할 수 있는 구조로 설계되어야 한다.

2.2.3 확장성(extendability)

확장성은 시스템 혹은 소프트웨어의 기능의 확장이나 환경의 변화에 대해 하드웨어/소프트웨어가 그것을 얼마나 쉽게 수용할 수 있는지를 나타내는 척도이다. 소프트웨어 기반 시뮬레이터의 확장성이란 외부의 시뮬레이터와의 연동, 내부적인 기능의 확장이 이루어질 수 있음을 의미한다. 예를 들어 본 논문의 사례연구인 위성항법 시뮬레이터[10]의 경우 현재 적게는 4개의 위성에 대한 신호를 생성하고 수신 처리하는 기능을 포함하고 있으나, 향후 27개의 위성, 더 나아가 54개의 위성에 대한 신호를 생성하도록 확장할 수 있어야 한다. 이는 내부적으로 시뮬레이션 모델의 확장이 필요하고 그것을 내부적으로 수용할 수 있어야 함을 의미한다. 또한 외부 위성항법 응용 시스템들과의 연동을 위한 시스템의 확장도 요구된다. 현재 시뮬레이터는 신호생성 시뮬레이터와 신호수신시뮬레이터의 상호작용에만 중점을 두고 있지만, 향후 항공이나 해양 등과 같이 다양한 위성항법 응용 시스템들과의 연동을 위해 웹을 기반으로 한 서비스, 네트워크를 통한 서비스 등의 확장성도 고려해야 한다.

3. 소프트웨어 기반 시뮬레이터의 아키텍처 패턴

3.1 제안 아키텍처 패턴

본 논문에서는 앞서 설명한 것과 같이 소프트웨어 기반 시뮬레이터의 일반화된 기능적 요구사항을 기반으로 비기능적 요구사항을 만족하기 위한 아키텍처 패턴을 정의한다. 기능적인 요구사항은 2장에서 기술한 여러 유형의 시뮬레이터에서 요구되는 기능의 일반화된 형태이며, 변경가능성, 상호운영성, 확장성은 가장 중요하게 요구되는 비기능적 요구사항이다. 본 논문에서는 이를 만족하기 위해 다음 그림 1과 같은 아키텍처 패턴을 정의한다.

본 논문에서 제안하는 소프트웨어 기반 시뮬레이터의 아키텍처 패턴은 크게 5가지 요소로 구성된다. 이는 다양한 소프트웨어 기반 시뮬레이터에서 요구되는 공통적

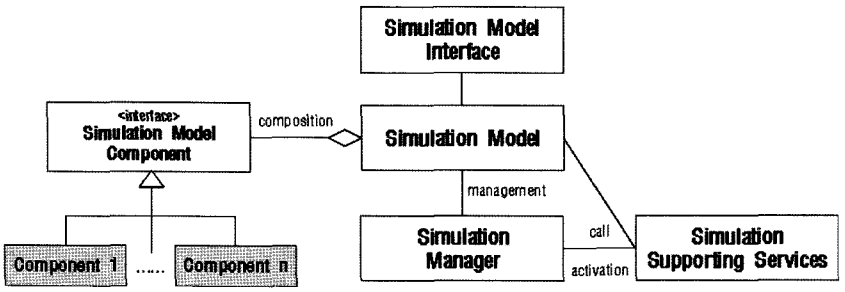

그림 1 소프트웨어 기반 시뮬레이터 아키텍처 패턴 개념도

Class Simulation Model Component	Collaborator • Other simulation model components • Simulation Model
Responsibility • 개별 시뮬레이션 알고리즘의 기능 및 서비스를 제공	

그림 2 시뮬레이션 모델 컴포넌트 CRC

Class Simulation Model Interface	Collaborator • External systems • Simulation Model
Responsibility • 시뮬레이션 모델과 외부의 시스템과의 인터페이스 역할 • 외부 시스템과의 연동을 위한 프로그램 인터페이스를 제공	

그림 3 시뮬레이션 모델 인터페이스 CRC

Class Simulation Manager	Collaborator • Simulation Model • Simulation Supporting Services
Responsibility • 시뮬레이션 모델 및 시뮬레이션 지원 서비스 초기화 기능 • 시뮬레이션 모델의 동작 및 환경을 관리 • 시뮬레이션 작업의 스케줄링 기능 제공	

그림 4 시뮬레이션 관리자 CRC

Class Simulation Supporting Services	Collaborator • Simulation Model • Simulation Manager
Responsibility • 시뮬레이션 모델 및 시뮬레이션 모델 컴포넌트에서 공통적으로 사용하는 기능을 제공	

그림 5 시뮬레이션 지원 서비스 CRC

Class Simulation Model	Collaborator • Simulation Model Component • Simulation Manager • Simulation Model Interface • Simulation Supporting Services
Responsibility • 시뮬레이션 모델 컴포넌트의 조합 • 시뮬레이터 시스템의 최 상위 기능을 제공 • 조합에 참여하는 개별 시뮬레이션 컴포넌트의 관리 기능	

그림 6 시뮬레이션 모델 CRC

인 요소를 일반화 하여 추출한 것들이다. 우선 소프트웨어 기반 시뮬레이터가 제공하는 핵심 기능은 '시뮬레이션 모델'로 구현된다. 이러한 시뮬레이션 모델은 내부적으로 개별 시뮬레이션 알고리즘을 구현한 '시뮬레이션 모델 컴포넌트'의 조합으로 구축된다. 시뮬레이션 작업의 수행을 제어하는 '시뮬레이션 관리자'가 초기의 시뮬레이션 모델을 초기화하고 작업 제어를 수행하며, '시뮬레이션 지원 서비스'는 시뮬레이션 모델 혹은 시뮬레이션 모델 컴포넌트가 동작하기 위한 공통 기능을 제공한다.

다음은 각 아키텍처 구성 요소에 대한 자세한 설명이다.

- 시뮬레이션 모델 컴포넌트(Simulation Model Components)(그림 2): 시뮬레이션 모델의 세부 로직 또는 알고리즘을 구현한 모듈의 추상화된 형태이다. 시뮬레이션 모델 컴포넌트의 변경가능성, 다른 컴포넌트들과의 상호운영성 및 확장성을 고려하여 그림 7과 같이 내부 구조를 정의하였다. 시뮬레이션 모델 컴포넌트는 앞서 설명한 것과 알고리즘의 개선, 시뮬레이션 모델의 변경과 같은 요구사항으로 인해 수시로 교체되거나 변경될 수 있다. 이 경우 시뮬레이션 모델 컴포넌트의 입/출력 데이터 종속성에 의해 그 변경의 범위가 결정된다. 본 논문에서는 이러한 시뮬레이션 모델 컴포넌트의 입/출력 데이터를 메시지로 추상화하여 변경 및 교체로 인해 발생하는 변경의 범위를 최소화한다. 또한 모든 시뮬레이션 모델 컴포넌트의 외부 인터페이스가 동일한 구조를 갖기 때문에 일관된 구조를 유지할 수 있다.
- 시뮬레이션 모델(Simulation Model)(그림 6): 시뮬레이터 시스템의 핵심 기능을 표현한 것으로 하위의 개별 시뮬레이션 모델 컴포넌트의 조합을 통해 실현된다. 예를 들어 교통흐름 시뮬레이터[11]는 도로 환경 시뮬레이션 로직, 자동차의 행위 표현 로직, 신호 체계 로직 등 다양한 시뮬레이션 로직의 조합을 통해 시뮬레이터

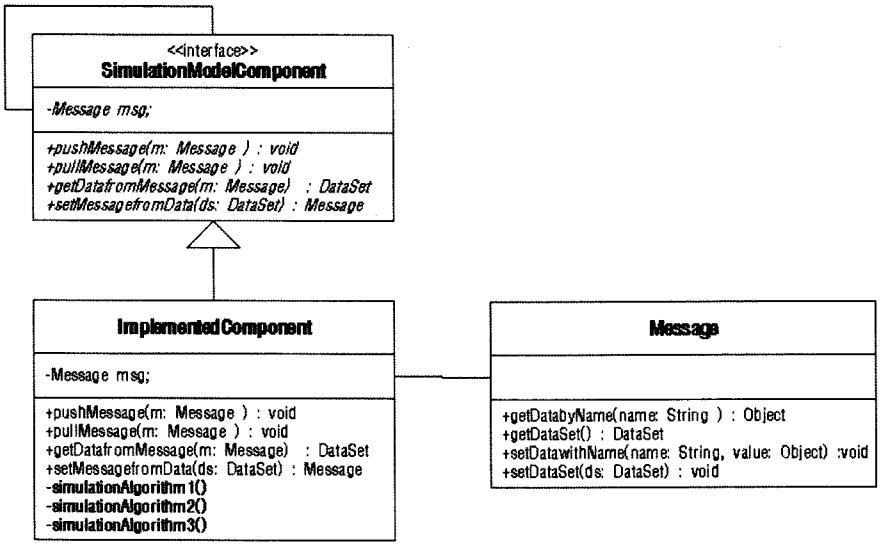

그림 7 시뮬레이션 모델 컴포넌트 클래스다이어그램

의 기능을 제공한다. 본 논문에서는 이러한 시뮬레이션 로직을 개별 시뮬레이션 모델 컴포넌트로 설계하고, 이들 간의 조합을 통해 시뮬레이션 모델을 정의한다.

- 시뮬레이션 모델 인터페이스(Simulation Model Interface)(그림 3): 시뮬레이션 모델 인터페이스는 외부 시스템과의 연동을 위한 인터페이스이다. 이 아키텍처 패턴에서는 인터페이스의 다원화를 통해 다양한 외부 시스템과의 상호운영성을 보장한다. 인터페이스 요소를 통해 다양한 시스템과의 입/출력 관계를 정의할 수 있다.
- 시뮬레이션 관리자(Simulation Manager)(그림 4): 시뮬레이션 모델을 수행하기 위하여 관련된 속성 및 환경에 대한 설정을 기반으로 시뮬레이션 기능이 원활하게 동작할 수 있도록 제어하는 기능을 담당한다. 예를 들어 해당 시뮬레이션 작업에 스케줄링 기능이 포함되어야 할 경우 시뮬레이션 지원 서비스 중 하나인 스케줄러 기능을 연결하여 시뮬레이션 작업을 제어한다.
- 시뮬레이션 지원 서비스(Simulation Supporting Services)(그림 5): 시뮬레이션 과정에 공통으로 요구되는 기능을 추상화한 것으로 능동적 서비스와 수동적 서비스로 구분된다. 능동적 서비스는 실제 시뮬레이션 작업을 활성화 시키는 서비스이다. 시뮬레이션 작업의 흐름에 따른 제어 기능을 수행하는 스케줄러, 이와 관련된 시뮬레이션 시간을 제어하는 타이머, 그리고 발생하는 이벤트를 처리하기 위한 이벤트 관리자 등이 있다. 수동적 서비스는 시뮬레이션 모델의 수행 과정에서 이용되는 서비스로 로깅 서비스, 데이터 관리 서비스 등이 있다.

3.2 제안 아키텍처 패턴의 동작

이 아키텍처 패턴의 동작은 두 가지 시나리오로 설명된다. 첫 번째는 시뮬레이션 시스템의 초기화 과정에 대한 내용이고, 두 번째는 시뮬레이션 시스템의 동작에 관한 설명이다.

시뮬레이터 시스템의 초기화 과정은 그림 8의 순서로 동작한다. 그림 8에서 시뮬레이터 시스템은 하나의 시뮬레이션 모델로 구축되며, 시뮬레이션 모델은 두 개의 시뮬레이션 모델 컴포넌트로 구성된다.

1) 사용자의 시뮬레이션 시스템 초기화에 대한 입력은 시뮬레이션 관리자가 받는다.
2) 시뮬레이션 관리자는 시뮬레이션 모델을 초기화한다.
3) 시뮬레이션 모델의 초기화 과정은 개별 시뮬레이션 모델 컴포넌트를 초기화하는 과정을 포함한다. 그림 8에서는 우선 시뮬레이션 모델 컴포넌트1을 초기화 한다.
4) 초기화된 시뮬레이션 모델 컴포넌트1은 시뮬레이션 모델에 자신의 컴포넌트를 등록한다.
5) 시뮬레이션 모델은 시뮬레이션 모델 컴포넌트2를 초기화한다.
6) 초기화된 시뮬레이션 모델 컴포넌트2는 시뮬레이션 모델에 자신의 컴포넌트를 등록한다.
7) 초기화가 완료된 시뮬레이션 모델은 시뮬레이션 관리자에 자신을 등록한다.
8) 시뮬레이션 관리자는 시뮬레이션 모델 인터페이스를 초기화한다.
9) 초기화된 시뮬레이션 모델 인터페이스는 시뮬레이션 관리자에 자신을 등록한다.

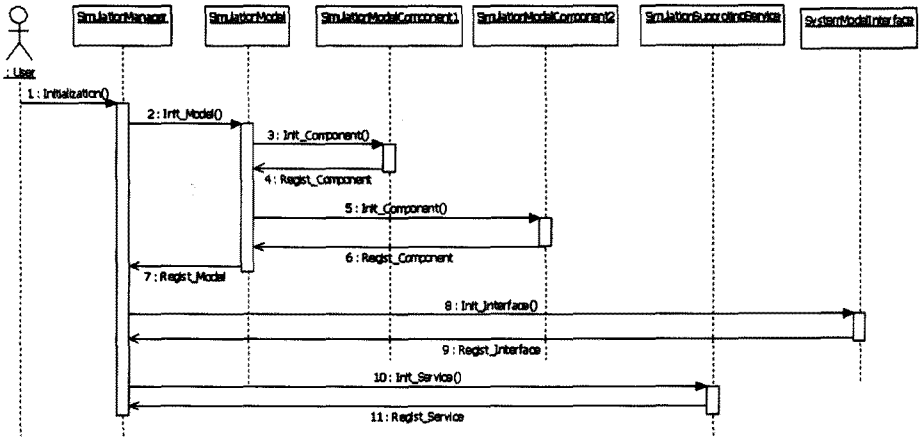

그림 8 시뮬레이션 시스템 초기화 과정

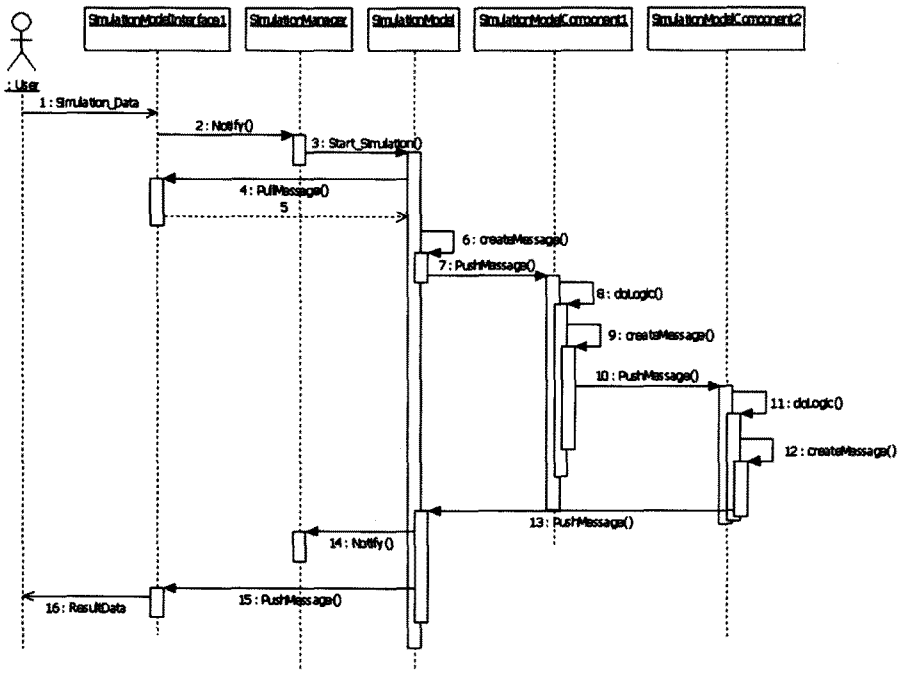

그림 9 시뮬레이션 시스템 동작 과정

10) 시뮬레이션 관리자는 시뮬레이션 지원 서비스를 초기화한다.

11) 초기화된 시뮬레이션 지원 서비스는 자신을 시뮬레이션 관리자에 등록한다.

그림 9는 본 논문의 아키텍처 패턴을 적용한 시뮬레이션 시스템의 동작 과정을 나타낸다. 동작 과정은 초기화된 시뮬레이션 시스템에서 입력 받은 데이터를 순차적으로 처리한 다음사용자에게 전달하는 과정을 보여준다.

1) 사용자로부터 시뮬레이션 데이터가 전달된다.

2) 시뮬레이션 모델 인터페이스1은 데이터를 입력받아 시뮬레이션 관리자에 통보한다.

3) 시뮬레이션 관리자는 시뮬레이션 모델에 시뮬레이션 작업을 수행시킨다.

4) 시뮬레이션 모델은 시뮬레이션 모델 인터페이스1에

필요한 데이터를 요청한다.

5) 시뮬레이션 모델 인터페이스1은 데이터를 반환한다.
6) 시뮬레이션 모델은 획득한 데이터를 메시지 형태로 변환한다.
7) 시뮬레이션 모델은 메시지형태의 데이터를 시뮬레이션 로직인 시뮬레이션 모델 컴포넌트1에 전달한다.
8) 시뮬레이션 모델 컴포넌트1은 전달 받은 데이터를 이용해 로직을 수행한다.
9) 시뮬레이션 모델 컴포넌트1은 로직의 수행 결과를 메시지 형태로 변환한다.
10) 시뮬레이션 모델 컴포넌트1은 생성한 메시지를 다음 로직인 시뮬레이션 모델 컴포넌트2에 전달한다.
11) 시뮬레이션 모델 컴포넌트2는 전달받은 메시지를 이용해 내부 로직을 수행한다.
12) 시뮬레이션 모델 컴포넌트2는 로직의 수행 결과를 메시지 형태로 변환한다.
13) 시뮬레이션 모델 컴포넌트2는 시뮬레이션 모델에 메시지를 전달한다.
14) 시뮬레이션 모델은 작업의 종료를 시뮬레이션 관리자에게 통보한다.
15) 시뮬레이션 모델은 시뮬레이션 모델 인터페이스1에 결과 메시지를 전달한다.
16) 시뮬레이션 모델 인터페이스1은 최종 결과를 사용자에게 반환한다.

3.3 제안 아키텍처 패턴의 구현 과정

다음은 본 논문에서 제시하는 아키텍처 패턴을 이용해 소프트웨어 기반 시뮬레이터 시스템을 구현하기 위한 가이드라인이다. 위에서 설명한 각 구성 요소의 구현 과정 및 고려사항은 다음과 같다.

Step0. 시뮬레이션 기능 정의: 시뮬레이션의 기능을 정의하는 것은 시뮬레이션 모델에서 제공하는 기능을 정의하는 것을 의미한다. 이러한 기능 정의는 시뮬레이션 시스템의 개발 목적과 일치하기 때문에 이미 결정되어 있다고 가정한다.

Step1. 개별 시뮬레이션 모델 컴포넌트 정의: 시뮬레이션 모델이 어떠한 알고리즘 혹은 로직들로 구현될지 결정하는 것이다. 또한 각 시뮬레이션 알고리즘 혹은 로직의 컴포넌트 단위를 결정하는 작업이 포함된다. 하나의 시뮬레이션 알고리즘은 다수개의 시뮬레이션 모델 컴포넌트로 분할되어 구현될 수 있으므로 각 시뮬레이션 모델 컴포넌트의 기능 및 인터페이스를 정의하여야 한다. 또한 시뮬레이션 모델 컴포넌트의 입/출력 데이터도 정의되어야 한다. 시뮬레이션 모델 컴포넌트의 단위는 시뮬레이터의 로직을 개발하는 개발자가 결정할 문제이지만 일반적으로 다음과 같은 원칙을 따른다.

- 시뮬레이션 모델의 기능 단위를 시뮬레이션 모델 컴포넌트로 정의
- 시뮬레이션 모델 컴포넌트들 간에 공통적인 요소를 새로운 컴포넌트로 분할
- 시뮬레이션 모델 컴포넌트의 입력과 출력을 담당하는 기능을 인터페이스로 표현

Step2. 시뮬레이션 지원 서비스 정의: 시뮬레이션 지원 서비스는 시뮬레이션 모델에서 필요로 하는 공통 기능이다. 시뮬레이션 지원 서비스를 정의하기 위해서는 다음과 같은 작업을 수행한다.

- 시뮬레이션 모델 컴포넌트에서 요구하는 지원 기능 파악
- 시뮬레이션 모델 컴포넌트를 활성화하기 위해 요구되는 지원 기능 파악
- 지원 기능의 서비스 제공 방법 결정
- 시뮬레이션 모델 컴포넌트의 연결 정보 명세 방법 결정

Step3. 시뮬레이션 모델 정의: 시뮬레이션 모델 정의 과정은 시뮬레이션 모델 컴포넌트들과 이들 간의 상호작용을 명세 하는 과정이다. 시뮬레이션 모델 명세에는 시뮬레이션 모델을 구성하는 각 시뮬레이션 모델 컴포넌트의 목록, 이들 간의 입출력 데이터 맵핑, 그리고 시뮬레이션 지원 서비스와의 연동 방법 등이 명시된다. 시뮬레이션 모델은 다음의 순서에 따라 정의된다.

- 시뮬레이션 모델 컴포넌트의 조합 방법 결정
- 시뮬레이션 모델 컴포넌트의 조합 명세 방법 결정
- 시뮬레이션 모델 컴포넌트의 연결 정보 명세 방법 결정
- 시뮬레이션 모델 명세 작성

Step4. 시뮬레이션 모델 컴포넌트 구현: 개별 시뮬레이션 모델 컴포넌트를 구현한다.

Step5. 시뮬레이션 모델 인터페이스 정의: 시뮬레이션 모델 인터페이스는 다음의 순서에 따라 개발한다.

- 시뮬레이션 모델의 입/출력에 해당하는 추상인터페이스를 구현
- 외부의 시스템과의 연동을 위한 인터페이스를 추상 인터페이스를 상속받아 구현, 예를 들어 파일 입출력 인터페이스의 경우 파일 읽기/쓰기 기능을 통해 해당 인터페이스를 구현
- 새로운 시스템과의 연동이 필요할 경우 새로운 인터페이스를 구현

Step6. 시뮬레이션 관리자 기능 정의 및 구현: 시뮬레이션 관리자는 각각의 시뮬레이터의 환경에 맞추어 개발하며, 다음과 같이 개발한다.

- 시뮬레이션 환경 및 속성에 대한 정보를 추출하여 관리하는 모듈 구현
- 시뮬레이션 모델의 동작과 관련된 동적인 정보를

관리하는 모듈 구현

- 시뮬레이션 모델의 동작에 시뮬레이션 지원 서비스가 필요한 경우 이를 연결

3.4 제안 아키텍처 패턴의 비기능적 요구사항 달성 여부

이 절에서는 소프트웨어 기반 시뮬레이터 시스템에 요구되는 변경가능성, 상호운영성, 확장성과 같은 비기능적 특성이 제안된 아키텍처 패턴에서 어떻게 달성되고 있는가를 살펴본다.

- 변경가능성의 달성 여부: 일반적으로 소프트웨어의 변경가능성을 달성하기 위한 전략으로 변경의 범위를 최소화하고, 변경의 파급효과를 방지하고, 변경 요인이 발휘되는 바인딩 시간을 늦추는 방법을 사용한다. 제안된 아키텍처 패턴에서는 시뮬레이션 모델 컴포넌트를 외부의 입/출력 기능을 담당하는 부분(메시지)과 내부 기능을 담당하는 컴포넌트로 분리하여 입/출력 데이터의 변경에 대한 변경의 범위를 최소화 하고, 그 파급효과를 최소화하도록 정의하였다. 시뮬레이션 명세를 통해 시뮬레이션 모델 컴포넌트를 시뮬레이션 수행 시에 바인딩 되도록 함으로써 바인딩 시간을 늦출 수 있게 되었다. 또한 시뮬레이션 시스템의 기능 변경은 시뮬레이션 명세를 통해 시뮬레이션 모델 컴포넌트의 동적인 재조합 방식으로 쉽게 달성할 수 있게 하였으며, 외부의 인터페이스 변경은 인터페이스 다원화를 통해 달성할 수 있게 하였다. 예를 들어 외부 입력으로 파일로부터 데이터를 로딩하여 사용하던 시스템이 웹을 통해 데이터를 전달받고자 할 경우에 입력 인터페이스만 변경하여 데이터를 읽어 들이게 함으로써 내부 로직에 아무런 변경이 없게 되고, 궁극적으로 변경으로 인한 파급효과를 최소화할 수 있었다.
- 상호운영성의 달성 여부: 시뮬레이션 모델과 시뮬레이션 모델 컴포넌트의 인터페이스 방식을 일관성 있게 유지하고, 메시지 기반으로 데이터를 전달하게 함으로써 상호운영성을 달성할 수 있게 되었다. 또한 외부 시스템과의 연동 시 다원화된 인터페이스의 확장을 통해 상호운영성을 쉽게 달성할 수 있도록 설계되었다.
- 확장성의 달성 여부: 시뮬레이션 모델 컴포넌트와 시뮬레이션 모델 인터페이스를 쉽게 추가할 수 있도록 아키텍처 패턴을 개방형 구조로 설계함으로써 확장성을 달성할 수 있게 되었다.

4. 소프트웨어 기반 시뮬레이터 아키텍처 적용 사례

4.1 위성항법 시뮬레이터 아키텍처 설계

위성항법 분야에서의 다각적인 발전에 대비해 한국전자통신연구원에서는 다양한 응용 개발을 위한 시뮬레이션 도구를 개발 중에 있다. 이 도구는 여러 위성의 상황에 대한 시뮬레이션을 수행하기 위해 신호생성부와 신호수신처리부로 구성되며, 앞으로 개발될 위성항법 시스템의 다양한 기능을 시뮬레이션 할 수 있게 함으로써 위성항법 응용 분야, 연구 분야, 교육 분야에 활용될 계획이다.

이 위성항법 시뮬레이터는 기존의 위성항법 시스템뿐만 아니라 향후 개발될 갈릴레오 위성, GPS(Global Positioning System) 위성의 현대화 등에 대해서도 그 기능을 시뮬레이션 해야 한다. 또한 시뮬레이션 대상 위성, 지상관제소, 수신기 등 참여하는 시스템의 수가 언제든지 증가할 수 있는 상황에 대해서도 이를 수용할 수 있게 확장성이 있어야 하며, 향후 개발될 다양한 알고리즘 및 시뮬레이션 모델의 검증을 위해 다양한 컴포넌트가 교체 가능하도록 개방형 아키텍처를 갖고 있어야 한다. 이런 요구사항을 반영하기 위해 본 논문에서 제시한 시뮬레이터 아키텍처 패턴을 적용하여 그림 10과 같이 위성항법 시뮬레이터의 아키텍처를 설계하였다. 위성항법 시뮬레이터의 아키텍처는 제안된 시뮬레이터 아키텍처 패턴을 확장하여 정의되었는데 시뮬레이션 모델 컴포넌트를 관리하기 위하여 시뮬레이션 모델 컴포넌트 관리자를 추가하였고, 시뮬레이션 지원 서비스를 능동적 서비스와 수동적 서비스로 세분하여 적용하였다. 능동적 서비스는 시뮬레이션 작업의 수행에 직접적으로 영향을 미치는 서비스로 개별 시뮬레이션 모델 컴포넌트를 활성화 시키거나, 작업 순서를 제어하는 기능을 수행한다. 예를 들어 타이머나 스케쥴링 서비스와 같은 것들이다. 수동적 서비스는 로깅 서비스나 데이터 변환 서비스와 같이 개별 시뮬레이션 모델 컴포넌트들이 요구하는 기능을 제공하는 서비스들이다. 그림 10에서 시뮬레이션 지원 서비스와 시뮬레이션 관리자와의 관계에서 화살표는 서비스 제공의 방향성을 나타낸다.

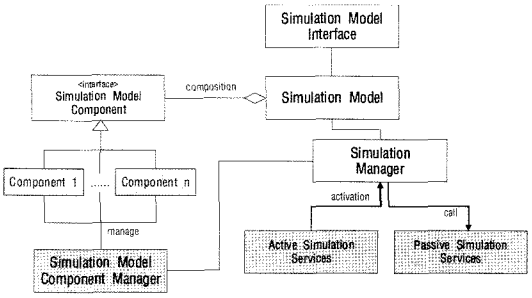

그림 10 위성항법 시뮬레이터-아키텍처 패턴 적용

위성항법 시뮬레이터 중 위성항법 신호생성 시뮬레이터와 위성항법 신호수신 시뮬레이터에 제안된 아키텍처 패턴을 적용하였으며, 적용 후의 위성항법 시뮬레이터의 아키텍처는 그림 11과 같다.

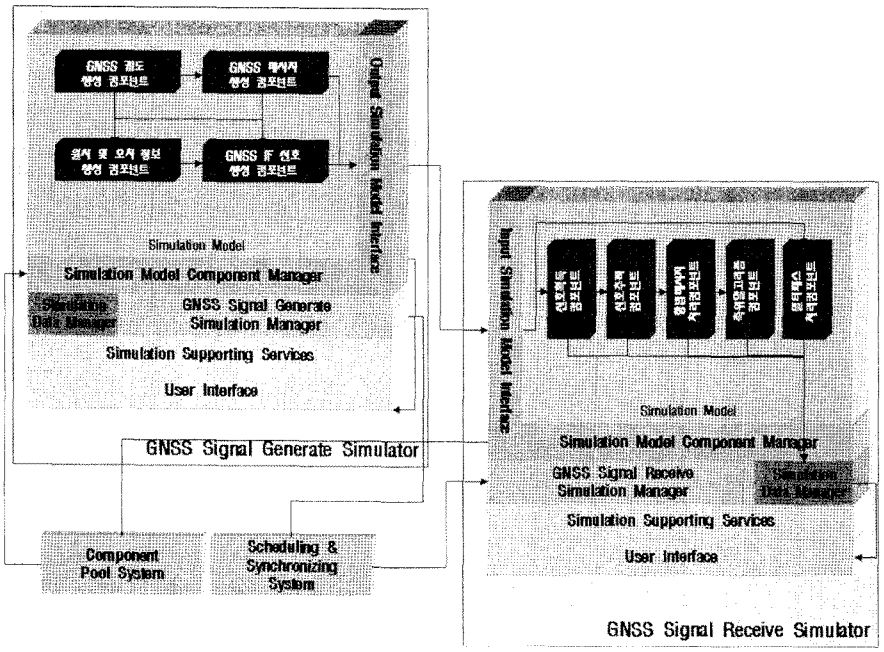

그림 11 위성항법 시뮬레이터 아키텍처

4.1.1 시뮬레이션 모델 컴포넌트

시뮬레이션 모델 컴포넌트는 기능 단위로 컴포넌트를 분할하였다. 다음은 위성항법 신호생성 시뮬레이터와 위성항법 신호수신 시뮬레이터에서 정의된 시뮬레이션 모델 컴포넌트에 대한 설명이다.

위성항법 신호생성 시뮬레이터 기능 요구사항

위성항법 신호생성 시뮬레이터는 여러 가지 목적의 항법수신기의 성능을 검증할 수 있도록 갈릴레오 및 GPS 위성의 신호를 생성하는 기능을 수행한다. 이때 실제 위성과 유사한 신호를 생성하기 위해 이온층, 대기권 및 도플러 효과 등의 요인으로 인해 발생하는 오차를 고려하여 IF(Intermediate Frequence) 레벨로 신호를 생성한다. 본 논문에서는 이러한 위성항법 신호 생성 시뮬레이터의 기능 요구사항으로부터 시뮬레이션 모델 컴포넌트에 대응되는 4개의 주요 기능들을 추출하였다.

- 항법위성궤도생성: GPS 24개 및 갈릴레오 30개의 위성궤도 정보를 생성한다. 비대칭 지구중력장, 달과 태양 및 다른 행성에 의한 중력장, 태양 복사압에 의한 섭동력을 포함하고 ECEF(Earth-Centered, Earth-Fixed) 지구 중심 고정좌표계를 사용하여 궤도시뮬레이션 알고리즘을 통해 그 기능을 구현한다.
- 위성항법 메시지생성: 궤도생성 데이터와 이온층 오차 파라미터, 위성시계오차 파라미터 등을 이용하여 GPS 및 갈릴레오 규격에 따라 GPS 12개 위성의 L1, L2C 메시지를 생성하고 갈릴레오 15개 위성의 E1, E5A메시지 생성 기능을 제공한다.
- 원시 및 오차정보 생성: 신호지연 오차(이온층, 대기권, 멀티패스) 및 시계정보를 이용하여 갈릴레오 15개 및 GPS 12개 위성과 수신기 간의 오차지연시간을 생성하며, 위성궤도정보와 수신기 동특성 정보를 이용하여 위성과 수신기 간의 전파지연시간 및 도플러를 생성하는 기능을 제공한다.
- IF 신호 생성: 오차정보 및 전파지연시간, 도플러 효과를 입력 받아서 GPS 항법 메시지와 GPS C/A(Coarse/Acquisition) 코드 및 IF 캐리어를 이용하여 IF 신호를 생성하는 기능, 실제 항법신호와 유사하도록 채널 노이즈를 추가하여 수신기의 RF(Radio Frequence) 입력 단에서 사용하는 대역폭 필터링 기능, 자동이득 조절기를 거쳐 최종적으로 양자화를 수행하는 기능을 포함한다.

위와 같은 기능 요구사항의 분석을 통해 4개의 기능을 그림 12와 같이 시뮬레이션 모델 컴포넌트로 설계하였다. 이때 모든 시뮬레이션 모델 컴포넌트들은 상호운영성 및 변경가능성을 위해 동일한 컴포넌트 인터페이스를 갖는다.

위성항법 신호생성 시뮬레이터의 경우 외부의 입/출력과 관련된 기능이 궤도생성 컴포넌트와 IF 신호생성 컴포넌트에 포함되어 있기 때문에 입/출력과 관련된 컴포넌트를 따로 나타내지 않았다. 또한 입/출력과 관련된 기능의 확장은 시뮬레이션 모델 컴포넌트의 setMessagefromData(), getDatafromMessage() 메소드를 확장해

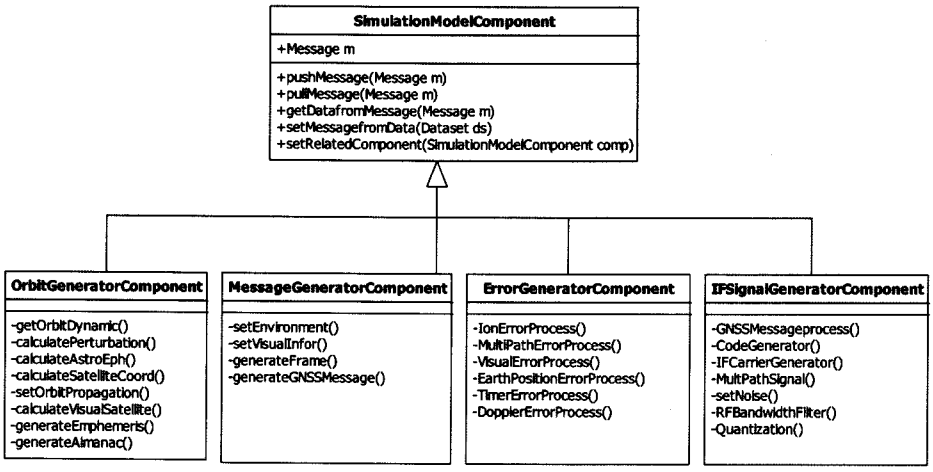

그림 12 시뮬레이션 모델 컴포넌트 설계

구현할 수 있도록 설계하였다.

위성항법 신호생성 시뮬레이터의 각 시뮬레이션 모델 컴포넌트들 사이에 교환될 데이터는 Message 클래스를 통해 일반화하였다. 이름-값 쌍(name-value pair)을 통해 어떠한 데이터가 전달되더라도 모든 컴포넌트에서 필요한 데이터를 추출할 수 있도록 정의하였다. 신호 생성 시뮬레이션 모델 컴포넌트에 대한 입력 메시지 구조는 표 1과 같으며, 모든 시뮬레이션 모델 컴포넌트에 동일한 방법으로 적용하였다. 또한 Message클래스를 확장해 궤도 생성 시뮬레이션 모델 컴포넌트를 이용하는 경우 그 내용을 이해할 수 있도록 설명을 달아주었다.

위성항법 신호수신 시뮬레이터의 기능 요구사항

위성항법 신호수신 시뮬레이터는 위성항법 신호생성 시뮬레이터에서 생성된 위성항법 신호를 수신하여 신호 획득, 신호 추적, 항법 메시지 처리, 측위 알고리즘 기능을 수행하여 사용자 애플리케이션에서 활용될 수 있는 항법메시지와 측정값을 추출하는 기능을 수행한다.

- 신호획득: 위성항법 신호생성 시뮬레이터로부터 디지털 위성신호를 수신하여, GPS/갈릴레오 신호의 PRN(Pseudo Random Number), 상관값, 코드 시작점, 추정 도플러 정보를 획득하는 기능을 수행한다.
- 신호추적: 신호획득 결과를 추적 초기 값으로 하여 위성과 사용자간 동적인 위치변화로 인한 코드 오프셋과 캐리어 도플러 변화를 추적하고 의사거리 계산을 위한 코드 시작점과 항법 데이터 비트를 얻기 위한 상관 적분 값을 제공하는 기능을 수행한다.
- 항법메시지처리: 위성항법 신호에 포함된 항법 메시지를 처리하여 시작점 및 위성의 데이터를 출력으로 제공하고 의사거리 및 반송파 정보를 계산하여 제공하는 기능을 수행한다.
- 측위 알고리즘: 위성 궤도 정보, 의사거리 및 반송파 정보, 위성항법 보정정보로부터 사용자의 위치, 속도, 시각 정보를 제공하는 기능을 수행한다.
- 멀티패스 처리: 단일 신호와 함께 들어오는 멀티패스를 완화 또는 제거하는 기능을 제공한다.

위의 위성항법 신호수신 시뮬레이터의 기능에 대해서도 시뮬레이션 모델 컴포넌트로 설계하였으며, 그림 11의 시뮬레이션 모델에 포함된 컴포넌트의 형태를 갖는다.

4.1.2 시뮬레이션 모델 명세 및 시뮬레이션 컴포넌트 관리자

시뮬레이션 모델은 시뮬레이션 모델 컴포넌트의 조합을 통해 생성된다. 위성항법 신호생성 및 신호수신 시뮬레이터의 경우 제시한 위의 컴포넌트의 조합을 이용해 시뮬레이션 모델을 생성한다. 그러나 지금 정의한 시뮬레이션 모델 컴포넌트는 현재의 요구사항을 반영하여 설계된 시뮬레이션 모델 컴포넌트이므로 향후 새로운

표 1 궤도생성 시뮬레이션 모델 컴포넌트 입력 메시지 형태

Name	Value	Description
Satellite State Vector	int SatelliteStateVector[6];	위성의 상태벡터
Satellite Orbit Element	Vector SatelliteOrbit;	위성궤도6요소
Simulation Start Time	Data start;	시뮬레이션 시작시간
Integral Scale	int scale;	적분 주기
Source of Perturbation	int sourceofPertu;	섭동력원-내부 구현 알고리즘을 선택

요구사항이 나타나면 컴포넌트의 추가와 교체를 통해 새로운 요구사항을 반영하게 될 것이다. 따라서 다양한 조합의 컴포넌트 모델이 시뮬레이터 상에서 동작할 수 있도록 시뮬레이션 모델에 대한 명세와 이를 통해 시뮬레이션 모델을 구축하는 기능을 수행할 시뮬레이션 컴포넌트 관리자를 설계하였다.

시뮬레이션 모델 명세는 시뮬레이션을 표현하는 시뮬레이션 모델 컴포넌트들의 조합에 관하여 기술한 것으로 참여하는 시뮬레이션 모델 컴포넌트의 목록과 각 시뮬레이션 모델 컴포넌트들 사이에 전달되어야 할 데이터에 관하여 기술하고 있다. 그림 11의 위성항법 신호생성 시뮬레이터의 모델은 다음과 같은 XML 기반의 시뮬레이션 모델 명세를 통해 정의된다.

```
<SimulationModelDescription>
  <ComponentList>
    <Component ID=1 name="OrbitGeneratrorComponent">
      <inputData name="Satellite State Vector" from="InputContainerInterface"/>
      <inputData name="Satellite Orbit Element" from="InputContainerInterface"/>
      <inputData name="Simulation Start Time" from="InputContainerInterface"/>
      ……
      <OutputData name="Satellite State Vector"/>
      <OutputData name="Satellite State Vector"/>
      ……
    </Component>
    <Component ID=2 name="MessageGeneratorComponent">
      <inputData name="Satellite Orbit Info" from="OrbitGeneratorComponent"/>
      <inputData name="Error Info" from="ErrorGeneratorComponent"/>
      ……
    </Component>
    ……
  </ComponentList>
<SimulationModelDescription>
```

그림 13 시뮬레이션 모델 명세

그림 13의 시뮬레이션 모델 명세는 <Component List> 엘리먼트 하위에 시뮬레이션 모델을 구성하는 시뮬레이션 모델 컴포넌트들의 목록을 갖는다. 각 시뮬레이션 모델 컴포넌트에는 입/출력 데이터와 연관되는 시뮬레이션 모델 컴포넌트의 정보를 포함한다. 위와 같은 시뮬레이션 모델 명세를 기반으로 시뮬레이션 관리자는 시뮬레이션 컴포넌트 관리자를 통해 시뮬레이션 모델을 완성한다. 그림 14는 시뮬레이션 모델을 완성하는 과정을 도식적으로 보여주고 있다. 시뮬레이션 관리자를 통해 시뮬레이션 모델 명세(①)가 시뮬레이션 모델 컴포넌트 관리자로 전달이 되면(②), 시뮬레이션 모델 명세에 기술된 순서에 따라 시뮬레이션 모델 컴포넌트를 생성하여 (③,④,⑤,⑥) 시뮬레이션 모델을 완성한다.

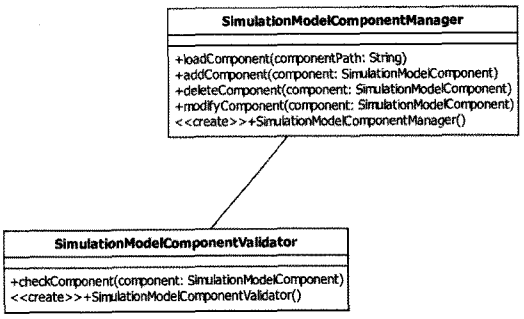

그림 15 시뮬레이션 모델 컴포넌트 관리자

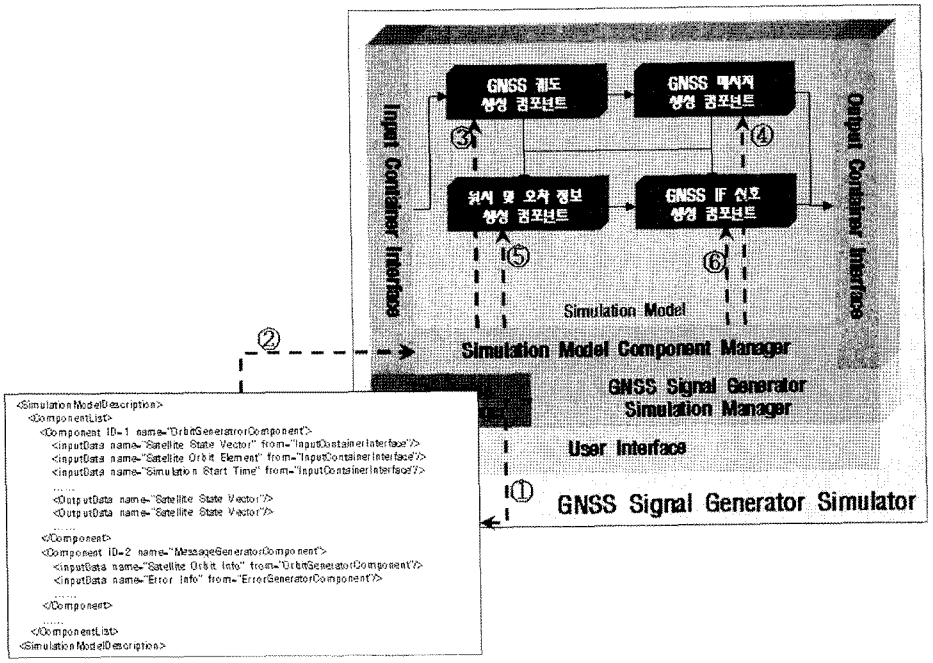

그림 14 시뮬레이션 모델 초기화

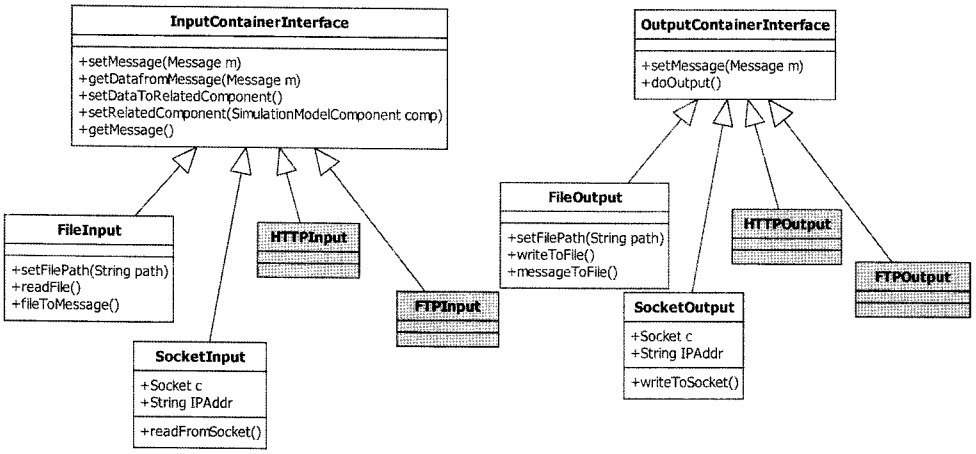

그림 16 시뮬레이션 모델 인터페이스 설계

이와 같은 시뮬레이션 모델 명세와 시뮬레이션 모델 컴포넌트 관리자를 통해 향후 새로운 시뮬레이션 모델 컴포넌트가 추가되거나 변경될 때에도 유연하게 대처할 수 있다. 그림 15의 시뮬레이션 모델 컴포넌트 관리자는 시뮬레이션 모델 컴포넌트를 등록하거나 수정할 때 시뮬레이션 모델 컴포넌트가 적합한지, 상호운영성에 문제가 없는지를 확인하기 위하여 컴포넌트 검증기를 통해 검증한다.

4.1.3 시뮬레이션 모델 인터페이스

시뮬레이션 모델 인터페이스는 외부 시스템과의 연동을 위한 인터페이스이다. 위성항법 신호생성 시뮬레이터 및 신호수신 시뮬레이터의 설계에 있어 각각 입출력을 담당하는 시뮬레이션 모델 인터페이스를 그림 16과 같이 설계하였다. 이 인터페이스는 현재 파일을 통한 입출력과 소켓을 이용한 직접 통신을 지원한다. 그러나 향후 웹, FTP 등과 같은 다양한 통신 방법을 지원하게 될 것이다. 이를 위해 시뮬레이션 모델 인터페이스를 추상화하여 향후 추가될 인터페이스가 이를 상속받아 구현될 수 있도록 설계하였다.

4.1.4 시뮬레이션 관리자

시뮬레이션 관리자는 전체 시뮬레이션 작업을 관리하고 통제하는 기능을 수행한다. 지금은 신호생성 및 신호수신 시뮬레이터를 대상으로 설계하였으나 향후 다양한 시뮬레이터들이 동일한 아키텍처로 개발될 것을 예상해 시뮬레이션 작업 관리 기능을 추상화하여 추상 클래스 SimulationManager로 표현하고 향후 개발될 시뮬레이터가 이를 구현할 수 있도록 설계하였다. 그림 17은 시뮬레이션 관리자의 클래스 다이어그램을 보여준다.

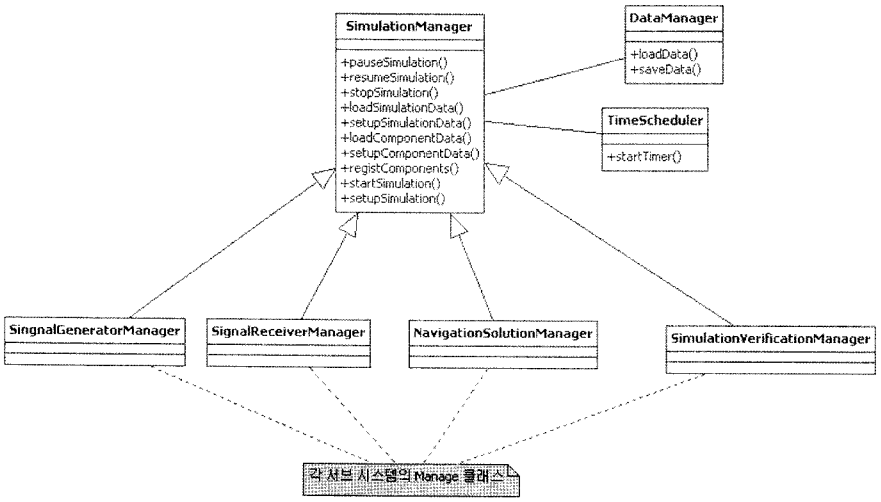

그림 17 시뮬레이션 관리자 설계

4.1.5 시뮬레이션 지원 서비스(그림 18)

시뮬레이션 지원 서비스는 능동적인 서비스와 수동적인 서비스로 나눠진다. 위성항법 시뮬레이터의 경우 시뮬레이션 모델의 동작과 관련된 능동 서비스로 타이머 서비스, 스케줄러 서비스가 존재한다. 위성항법 신호생성 시뮬레이터는 실제 위성이 주기적으로 전송하는 신호를 생성하기 위해 타이머 서비스가 그 주기를 결정한다. 또한 각 시뮬레이션 모델 컴포넌트의 작업 순서를 제어하는 것은 스케줄러 서비스가 담당한다. 위성항법 시뮬레이터의 수동 서비스로는 시뮬레이션 작업 상황을 기록하기 위한 로깅 서비스가 있다.

4.2 아키텍처 패턴의 적용 효과

이 절에서는 시뮬레이터 아키텍처 패턴의 적용 효과를 살펴보기 위해 기존 위성항법 시뮬레이터의 아키텍처(그림 19)와 아키텍처 패턴을 적용한 시뮬레이터의 아키텍처(그림 11)를 비교하면서 기존 아키텍처에서 달성하기 어려웠던 품질 특성이 새로운 아키텍처에서 달성될 수 있음을 보이고자 한다.

논문 [10]에서는 위성항법 시뮬레이터의 초기 설계를 제시하고 있다. 초기 설계에서는 위성항법 시뮬레이터를 기능 요소 중심으로 설계하였으며, 본 논문에서 제시하는 소프트웨어 기반 시뮬레이터의 필수 기능 요소들을 모두 포함하고 있지 않았다. 또한 비기능적 요구사항을 고려하지 않았기 때문에 시뮬레이터의 변경 및 확장에 대한 요구사항을 반영하기 어려웠다. 그림 19는 기존 위성항법 시뮬레이터의 신호생성 부분의 설계 내용이다.

기존 위성항법 시뮬레이터는 향후 개발될 시뮬레이션 알고리즘의 변경에 대한 요구사항을 반영하기 어렵다. 그림 19에서 보는 바와 같이 특정 컴포넌트가 교체되면 전체 시뮬레이터를 새롭게 빌드(build)하여야 한다. 따라서 다양한 알고리즘의 변화를 시뮬레이션 할 때 요구되는 즉시성을 달성하기 어렵다. 그러나 본 논문의 시뮬레이터는 시뮬레이션 아키텍처 패턴을 적용함으로써 변경가능성을 쉽게 달성할 수 있다. 시뮬레이터의 내부 알고리즘을 시뮬레이션 모델 컴포넌트로 구현하고, 컴포넌트들을 공통된 인터페이스를 갖는 컨테이너에 포함하여 관리함으로써 컴포넌트가 교체된다고 하더라도 시뮬레이터 전체를 다시 빌드할 필요가 없어서 컴포넌트의 교체가 용이하다. 또한 전체 시뮬레이션 모델의 구성을 그림 13과 같은 모델 명세파일로 관리함으로써 실행 시점에 그 구성을 변경할 수 있다. 아울러 표 1과 같이 개별 시뮬레이션 모델 컴포넌트의 입출력 데이터를 메시지화함으로써 데이터의 변경에 의한 영향 범위를 최소화하였다. 따라서 향후 개발될 다양한 시뮬레이션 알고리즘은 시뮬레이션 모델 컴포넌트 인터페이스를 구현하는 것으로 쉽게 변경/교체가 가능하다. 또한 시뮬레이터 시스템의 내부 코드의 변경이 아닌 모델 명세파일의 수정을 통해 시뮬레이션 모델의 구성을 쉽게 변경할 수 있기 때문에 시뮬레이터의 구현 사항을 자세히 모르더라도 쉽게 그 구성을 변경해가면서 시뮬레이션 작업을 수행할 수 있다.

또한 기존 위성항법 시뮬레이터는 외부 시스템과의 상호운영성 측면에서 단순한 방법의 연동만을 고려하였다. 즉 직접적인 연동보다는 각 시스템이 수행될 때 생성되는 모든 데이터를 파일로 저장하고, 필요한 시점에 파일을 전송하는 방식으로 연동하였다. 그러나 시뮬레이션 아키텍처 패턴을 적용하면서 다양한 형태의 상호운영성을 달성할 수 있었다. 앞서 기술한 바와 같이 상호

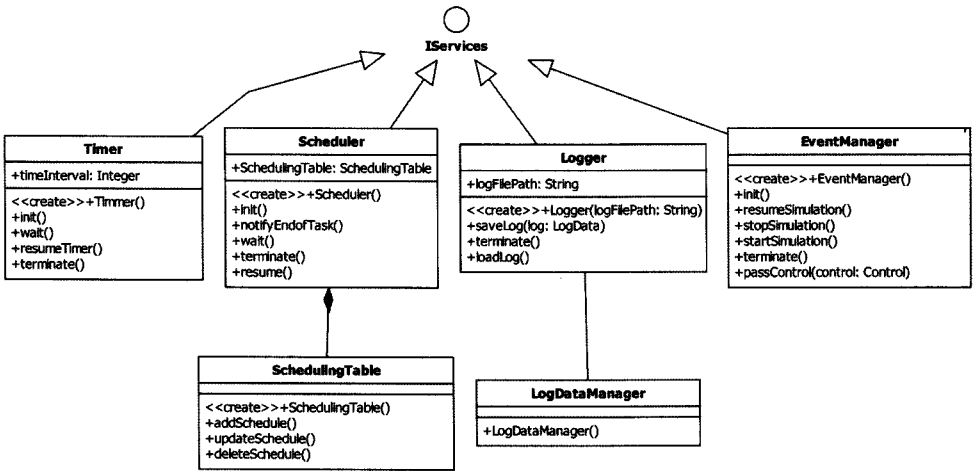

그림 18 시뮬레이션 지원 서비스

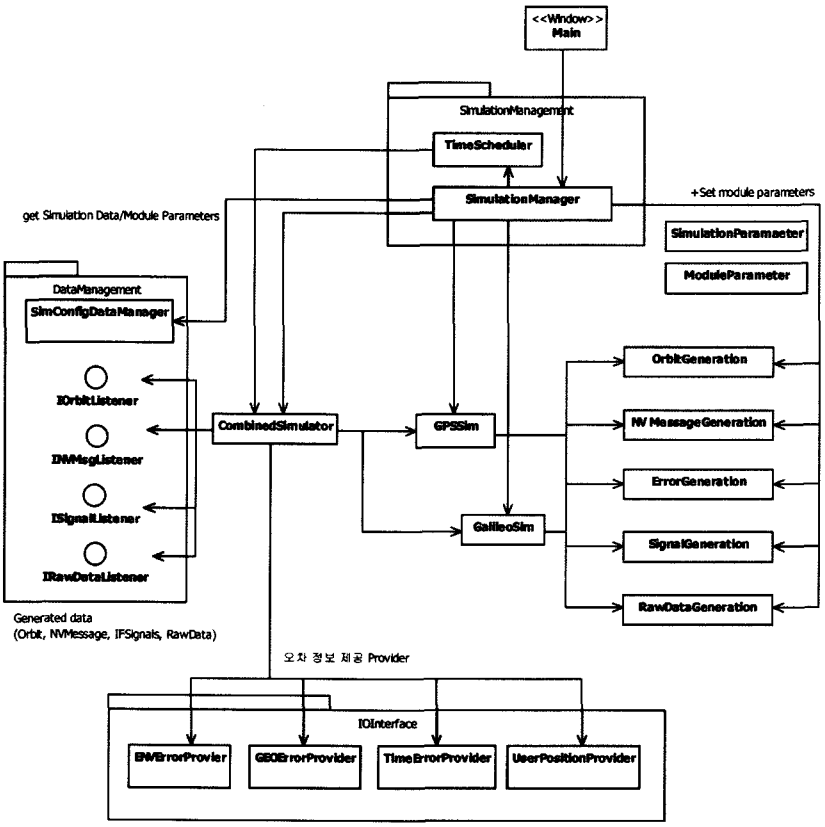

그림 19 기존 위성항법 시뮬레이터 구조

운영성은 내부 시뮬레이션 모델 컴포넌트들 간의 연동과 외부 시스템과의 연동을 고려하였다. 인터페이스의 통일, 입출력 데이터의 메시지화를 통해 내부 시뮬레이션 모델 컴포넌트들 사이의 상호운영성을 달성하였으며, 향후 다양한 시스템과의 연동을 고려하여 그림 16과 같이 외부 시스템과의 인터페이스 방식을 다원화함으로써 외부 시스템과의 상호운영성을 달성하였다.

마지막으로, 확장성은 변경가능성과 상호운영성을 달성하기 위해 사용되었던 요소들을 통해 향상되었다. 인터페이스의 통일, 설정파일 기반의 동적인 시뮬레이션 모델 구성 기능을 통해 내부 시스템의 확장성을 쉽게 달성할 수 있고, 외부 시스템과의 연동을 통한 확장성 역시 인터페이스 방식의 다원화를 통해 쉽게 달성할 수 있다.

5. 기존 아키텍처 패턴들과의 비교

이 논문에서 우리는 소프트웨어 기반 시뮬레이터의 아키텍처 패턴을 제안하였다. 기존에 다양한 연구를 통해 여러 유형의 아키텍처 패턴 및 설계 패턴이 제시되었다. 그것들은 각각 적용 대상 아키텍처의 핵심적인 요소를 추상적으로 모델링하기 위해 아키텍처 패턴이 적용될 시스템들의 기능적 및 비기능적 요구사항을 파악하고 이런 요구사항을 바탕으로 시스템 구성 요소들을 모델링하고 있다. 이 절에서는 이 논문에서 제안하는 시뮬레이터 아키텍처 패턴과 유사성을 갖는 기존 패턴들을 비교하여 이 논문에서 제안하는 아키텍처 패턴이 소프트웨어 기반 시뮬레이터 시스템의 참조 아키텍처로 타당함을 보이고자 한다. 이 장에서는 홀-파트(Whole-Part)[16], 컴포지트(Composite)[17], 컴포넌트-컨피규레이터(Component-Configurator)[18] 패턴을 비교 대상 아키텍처 패턴으로 선정하였다.

그림 20은 홀-파트 패턴을 보여준다. 이 패턴은 시스템을 구성하는 다양한 기능 요소를 분할하여 파트 컴포넌트로 구성하고 이를 조합하여 전체 기능을 구현한 홀 컴포넌트로 구축하는 설계 패턴이다. 이러한 홀-파트 패턴은 시스템의 변경 요구사항을 파트 컴포넌트들을 통해 쉽게 반영할 수 있으며, 또한 파트 컴포넌트를 추가함으로써 기능의 확장을 용이하게 이룰 수 있다는 장점을 갖는다.

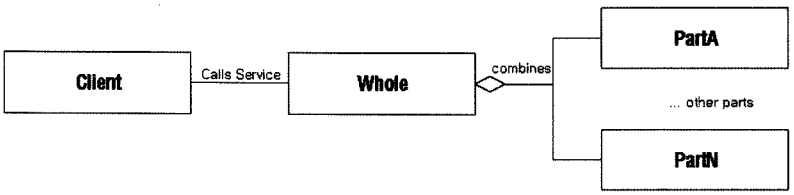

그림 20 홀-파트(Whole-Part) 패턴

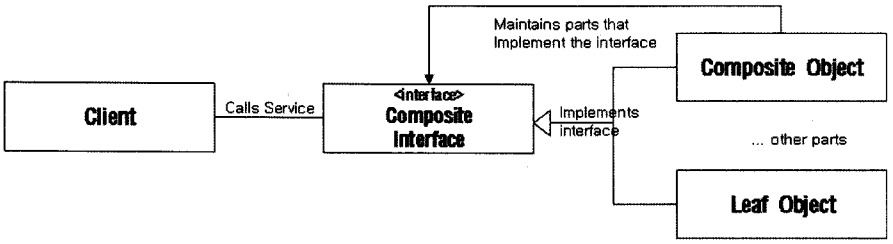

그림 21 컴포지트(Composite) 패턴

앞서 설명한 바와 같이 시뮬레이션 모델은 여러 시뮬레이션 모델 컴포넌트의 조합을 통해 구현되며 수시로 시뮬레이션 모델 컴포넌트의 변경을 요구한다. 따라서 시뮬레이션 모델 컴포넌트의 변경 및 교체가 쉽게 이루어져야 한다. 이러한 관점에서 홀-파트 패턴은 시뮬레이션 모델이나 시뮬레이션 모델 컴포넌트를 구현하는데 적합하다고 할 수 있다. 그러나 대부분의 시뮬레이션 시스템은 이러한 시뮬레이션 모델 컴포넌트의 교체가 동적으로 이루어지는 것을 요구하고 있다. 따라서 시뮬레이션 모델을 홀(Whole)로 구현할 경우 이를 동적으로 구성하기 위한 요소가 추가되어야 한다. 또한 시뮬레이터 시스템은 시뮬레이션 모델을 수행하고, 제어하는 기능이 포함되어야 하며, 다양한 시스템과의 연동을 고려하여 구현되어야한다. 따라서 홀-파트 패턴은 시뮬레이션 모델을 구축하는 데 활용될 수 있으나 시뮬레이터 시스템의 아키텍처 패턴으로 활용되기 위해서는 시뮬레이션 관리자의 기능을 담당하게 될 컴포넌트와 스케줄링 등과 같은 지원 서비스 기능을 수행할 컴포넌트 등이 추가되어야만 한다.

그림 21의 컴포지트 패턴은 시스템의 기능 모듈들을 조합하여 상위 컴포넌트를 구축한다는 점에서 홀-파트 패턴과 유사한 설계 패턴이다. 그러나 홀-파트 패턴이 홀 컴포넌트의 내부 구현을 위해 파트 컴포넌트들을 조합하였다면, 컴포지트 패턴은 인터페이스를 구현하는 하위 컴포넌트들을 조합하여 상위 컴포지트 컴포넌트를 구현한다. 이 패턴 역시 하위 컴포넌트의 교체나 변경이 쉽고, 새로운 컴포넌트의 추가도 인터페이스를 통해 쉽게 이루어질 수 있다. 그러나 앞서 설명한 홀-파트 패턴과 마찬가지로 컴포지트 패턴은 시뮬레이션 모델을 구현하기에는 적합할 수 있으나, 시뮬레이션 모델을 동적으로 조합할 수 있도록 지원하는 요소가 추가되어야 한다. 또한 앞서 설명한 것과 같이 소프트웨어 기반 시뮬레이터는 단순히 시뮬레이션 모델만으로 완성되는 것이 아니다. 시뮬레이션 작업의 실행 및 통제가 필요하며, 시뮬레이션 모델과 시뮬레이션 지원 서비스가 분리되어야 한다.

그림 22의 컴포넌트-컨피규레이터 패턴은 이 논문에서 제시하는 시뮬레이터 아키텍처 패턴과 좀 더 유사하다. 이 패턴에서는 시스템을 구성하는 다양한 기능을 컴포넌트로 구현하고, 그 컴포넌트들을 컨피규레이터를 통해 동적으로 조합하여 시스템을 구축한다. 이 패턴에서 컨피규레이터는 컴포넌트 조합 명세를 통해 실행 시간에 컴포넌트들을 조합한다.

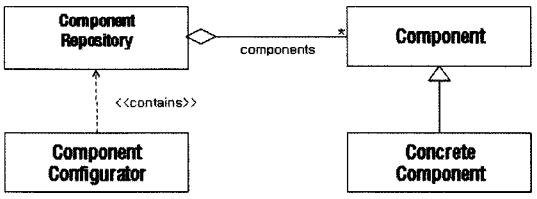

그림 22 컴포넌트-컨피규레이터(Component-Configurator) 패턴

컴포넌트-컨피규레이터 패턴을 시뮬레이터 아키텍처에 적용할 경우 시뮬레이터 시스템의 개발에서 요구되는 몇 가지 요구사항을 만족할 수 있다. 시뮬레이션 모델의 정의 및 관리는 컨피규레이터를 통해 구현될 수 있으며, 세부 기능에 해당하는 시뮬레이션 모델 컴포넌트는 구체적인 컴포넌트(concrete component)로 구현될

수 있다. 그러나 이 패턴은 전체 기능을 실행하고 통제하는 역할을 수행할 요소를 포함하고 있지 않다. 또한 이 패턴에서는 시뮬레이션 모델을 구성하는 기능 컴포넌트들과 지원 서비스 컴포넌트가 동일한 컴포넌트의 개념 아래에서 구현된다. 그러나 시뮬레이터 시스템에서는 시뮬레이션에 참여하는 기능 컴포넌트와 시뮬레이션을 지원하는 지원 서비스 컴포넌트와의 구분이 필요하다. 따라서 이 패턴이 시뮬레이터 시스템의 개발에 적용되기 위해서는 시뮬레이션 관리자, 시뮬레이션 지원 서비스의 역할과 기능이 컨피규레이터와 컴포넌트로부터 각각 분리되어 적용되어야 한다. 이 논문에서 제시하는 아키텍처 패턴은 이러한 시뮬레이터의 기능적, 비기능적 요구사항을 반영하여 아키텍처 요소를 분할하여 모델링하였기 때문에 향후 개발될 다양한 시뮬레이터의 참조 아키텍처로 활용 가능하다.

6. 결론 및 향후 연구 방향

시뮬레이션은 컴퓨터를 이용하여 실제 시스템들을 모방하여 그 특징을 찾아내는 작업을 지칭하며, 시뮬레이션 작업을 수행하는 소프트웨어/하드웨어적 장치를 시뮬레이터라 한다. 이러한 시뮬레이션은 실제 시스템의 동작에 대한 정보를 획득하거나 시스템의 성능 향상을 위해 운영 및 자원 관리 방법의 개발, 새로운 개념 및 시스템의 구현 전 테스트, 분산 시스템의 구축 전 동작에 대한 정보 획득을 위해 널리 사용된다. 그러나 시뮬레이션은 해당 시스템의 정적, 동적인 특징을 반영한 모델을 기반으로 작업이 수행된다. 시뮬레이션 모델은 수시로 변경되며, 이를 검증하기 위한 작업이 반복적으로 수행된다. 시뮬레이션 모델의 변경에 따른 시뮬레이터의 내부 구조의 변경, 외부의 시스템과의 연동을 통한 새로운 해석의 도출, 시뮬레이션 모델 확장에 따른 기능의 확장은 대부분의 시뮬레이터 개발에서 발생하는 요구사항들이다.

이 논문에서는 소프트웨어 기반 시뮬레이터 시스템의 개발에 있어 보다 더 특화된 아키텍처 패턴을 제시하는 것이 목적이며, 이를 위해 여러 유형의 시뮬레이션 시스템을 분석하여 기능적 요소를 파악하였고, 시뮬레이션 시스템이 요구하는 비기능적 요구사항을 중심으로 이를 달성하기 위한 아키텍처 패턴을 제시하였다. 이 패턴의 설계 과정에서는 시뮬레이션 수행 시 많이 나타나는 변경가능성, 상호운영성, 확장성과 같은 비기능적 요구사항을 반영하였다. 시뮬레이션 모델 컴포넌트를 조합하여 시뮬레이션 모델을 정의하는 방법, 시뮬레이션 매니저를 통한 시뮬레이션 모델의 수행 방법 등을 통하여 다양한 시뮬레이션 시스템을 구축하는데 이 패턴을 활용할 수 있다. 이 패턴은 시뮬레이션 모델 명세를 변경하는 방식으로 유연하게 시스템의 재구축을 가능하게 함으로써 변경가능성과 확장성을 보장한다. 또한 시뮬레이션 모델 컴포넌트의 인터페이스를 단순화하고 일관성을 유지함으로써 상호운영성을 보장하고, 시뮬레이션 모델에 다양한 인터페이스를 추가할 수 있게 함으로써 확장성을 보장한다. 본 논문에서 제시한 시뮬레이터 아키텍처 패턴은 한국전자통신연구원에서 개발 중인 위성항법 시뮬레이터 적용되었으며, 시뮬레이션 모델 컴포넌트의 교체를 통해 다양한 시뮬레이션 작업을 수행 중에 있다.

본 논문의 아키텍처 패턴은 향후 다양한 시뮬레이터 개발에서 고려해야할 비기능적 요소를 반영할 수 있도록 가이드라인을 제시하는 것에 그 의미를 두며, 향후 또 다른 비기능적 요소를 반영할 수 있도록 아키텍처 패턴의 변형(variants)에 대해서 연구할 예정이다.

참 고 문 헌

[1] C. A. Chung, *SIMULATION MODELING HANDBOOK: A Practical Approach*, CRC Press, 2004.

[2] S. M. Cha, Y. H. Chang, "Development of a Virtual Simulator for Agile Manufacturing System," *26th Annual Conference of the IEEE*, vol.3, pp.1949-1954, 2000.

[3] A.Sulistio, C.S.Yeo, R.Buyya, "Taxonomy of computer-based simulation and its mapping to parallel and distributed system simulation tools," *Software Practice and Experience*, pp.653-673, 2004.

[4] AnyLogic, http://www.xjtek.com/

[5] ARENA, http://www.arenasimulation.com/

[6] BuildSim, http://www.tritera.com/products/web_buildsim/bs_page1.shtml

[7] Extendsim, http://www.extendsim.com/prods_overview.html

[8] Flexsim, http://www.flexsim.com/

[9] SimApp, http://www.simapp.com/

[10] J. E. Lee, I. W. Joo, S. Lee, J. H. Kim, "Preliminary Design of Software GNSS Signal Simulator," *The Proceeding of Korean Society for Aeronautical and Space Sciences Conference*, pp.395-398, 2008. (in Korean)

[11] H. G. Lee, S. H. Chung, K. R. Ryu, "A Parallel Microscopic Simulator for Urban Traffic Modeling," *Journal of KISS*, vol.25, no.12, pp.1359-1367, 1998. (in Korean)

[12] J. H. Shim, H. K. Jung, W. C. Lee, K. H. Choi, S. K. Park, G. H. Jung, "Implementation of a Network Simulator for Cyber Attacks and Detections based on SSFNet," *Journal of KISS:Computing Practices*, vol.8, no.4, 2002. (in Korean)

[13] J. C. Kim, C. S. Yu, "PC를 이용한 경항공기 비행운동 시뮬레이터", The *Proc. of Korean Society for Aeronautical and Space Sciences Conference*, pp. 392-397, 1994. (in Korean)

[14] VEGA, SMP2.0 Handbook, European Space Agency,

2005.
[15] S. xiaoxia, Z. Qiuhai, "AN INTRODUCTION TO THE HIGH LEVEL ARCHITECTURE (HLA) RUNTIME INFRASTRUCTURE (RTI)," *SICE 2003 Annual Conference*, vol.1, pp.1136-1139, 2003.
[16] F. Buschmann, R. Meunier, H.Rohnert, P. Sommerlad, M.Stal, Pattern Oriented Software Architecture-A System of Patterns, vol.1, Wiley, 1996, pp.225-242.
[17] F. Buschmann, K. Henny, D. C. Schmidt, Pattern Oriented Software Architecture-A Pattern Language for Distributed Computing, vol.4, Wiley, 2007, pp.319-321.
[18] F. Buschmann, H. Rohnert, D. C. Schmidt, M. Stal, Pattern Oriented Software Architecture-Patterns for Concurrent and Networked Object, vol.2, Wiley, 2000, pp.225-242.

국 승 학
정보과학회논문지 : 소프트웨어 및 응용
제 36 권 제 1 호 참조

김 현 수
정보과학회논문지 : 소프트웨어 및 응용
제 36 권 제 1 호 참조

이 상 욱
1988년 2월 연세대학교 천문기상학(이학사). 1991년 3월 미국 Auburn대학교 항공우주공학(석사). 1994년 3월 미국 Auburn대학교 항공우주공학(박사). 1993년 3월 ~현재 한국전자통신연구원, 책임연구원. 관심분야는 위성제어 및 관제, 위성항법 및 항법 응용