한국시뮬레이션학회 논문지
Vol. 19, No. 1, pp. 153-160 (2010. 3)

모델 행동 양식의 시뮬레이션을 이용한 초기 디자인 검증 방법

신승훈[1] · 박승규[2†] · 최경희[2]

A Strategy for Validation in Preliminary Design Stage using The Simulation of Model Behavior

Seung Hun Shin · Seung Kyu Park · Kyung Hee Choi

ABSTRACT

Most part of errors in software development process are included during the stage of requirements definition and design. And correction or elimination of errors from those stages requires much more efforts and costs than those from the later part of software development process. However, despite of the importance of the validation of requirement definition and design stages, several kinds of problem have made it hard to be done successfully. Therefore, in this paper, we introduce a novel validation process for the preliminary design stage. The validation process is based on simulations of model and it can be used to validate requirements and model simultaneously. Models in the validation process will take only the behavior of software and be built on Ptolemy framework. The usability of our validation process is confirmed with a case study over DNS system environment. And the result of simulation shows well-known errors or vulnerabilities can be found with simulations of model which has the behavior of software. This means our validation process can be used as a process to validate requirements and models during the early stage of software development process.

Key words : Model Validation, Behavior Test, Domain Name System, Ptolemy

요 약

소프트웨어 개발 프로세스에서 발생되는 오류의 대부분은 요구사항 정의와 디자인 단계에서 비롯된다. 또한 이들 단계에서 포함된 오류의 수정 및 제거는 이후 단계에서 포함된 오류의 수정에 비해 훨씬 많은 노력과 비용을 요구한다. 하지만 요구사항과 디자인된 모델의 검증은 매우 중요한 작업임에도 불구하고 다양한 문제점들로 인해 성공적으로 수행되고 있지 못하다. 따라서 본 논문에서는 초기 디자인 단계에서 요구사항 및 모델의 검증 작업을 동시에 수행할 수 있는 모델의 시뮬레이션을 통한 검증 절차를 제안한다. 이 때 작성되는 모델은 Ptolemy 상에 소프트웨어의 행동 양식을 바탕으로 작성된다. 제안된 방법을 DNS 시스템 환경에 적용하여 사용 가능성을 확인하였으며, 시뮬레이션 결과에 따르면 행동 양식을 반영한 모델의 시뮬레이션을 통해 기존에 알려진 시스템의 내재된 오류 혹은 취약성을 발견 가능함이 확인되었다. 이는 제안된 방법이 소프트웨어 개발 프로세스 초기에서 요구사항과 모델을 검증하는 절차로 이용될 수 있음을 의미한다.

주요어 : 모델 검증, 행동 양식 테스트, 도메인 네임 시스템, 볼레미

1. 서 론

IEEE의 표준 문서들(IEEE, 1990, 1998, 2004)은 검증(Validation)을 "시스템이나 컴포넌트가 정의된 요구조건을 만족하는지 여부를 확인하기 위해 개발 프로세스의 중간 혹은 마지막에 수행되는 시스템 혹은 컴포넌트에 대한 평가 프로세스"라고 정의하고 있으며, 널리 알려진 것과 같이 이는 소프트웨어의 품질과 신뢰도 보장에 지대한 영향을 미치므로 소프트웨어 개발 프로세스의 모든 단계에서 수행되어야 한다. 하지만 소프트웨어의 개발과정에서 발생하는 전체 버그의 83%가 요구사항과 디자인 단계에 존재한다(Mogyorodi, 2001)는 것은 해당 단계에서 수행

2009년 12월 21일 접수, 2010년 3월 8일 채택
[1] 아주대학교 정보통신공학과 박사과정
[2] 아주대학교 정보 및 컴퓨터공학부 교수
주 저 자 : 신승훈
교신저자 : 박승규
E-mail; sparky@ajou.ac.kr

되는 검증 작업이 정상적으로 이루어지지 못하고 있음을 의미한다. 이와 같은 문제점은 디자인 이후 단계에서 버그를 찾아내 수정하는데 더욱 많은 노력과 비용이 소모되도록 하는 원인이 되므로, 요구사항 정의 및 초기 디자인 단계에서 수행되는 검증 작업의 중요성은 그 이후 단계에서 수행되는 검증 작업에 비해 훨씬 더 크다고 할 수 있다. 이와 같이 소프트웨어 개발 초기 단계에서 요구사항의 검증과 요구사항을 토대로 작성되는 초기 모델의 검증을 통한 오류 제거는 매우 중요한 작업의 하나이지만, 다양한 어려움으로 인해 실제적인 검증 작업은 모델 작성 완료 이후 단계에서야 활발히 수행되는 현상을 보이는데, 디자인 단계의 검증 작업을 어렵게 하는 요소 가운데 본 논문에서 주목하는 것들은 다음과 같은 사항들이다.

- 요구사항이 자연어로 기술되는 경우 기술자와 개발자 사이에서 발생 가능한 문맥 전달 과정상의 오류 배제가 어려움
- 요구사항의 명확한 의미전달을 위해 UML, XML 기반의 정형화된 기술들이 제안 및 사용되고 있으나, 이들의 처리 기술이 성숙되지 않아 정의된 모델에 내재된 설계상의 오류 판단은 어려움
- 개발 초기 단계에서는 프로그램 코드가 존재하지 않기 때문에 테스팅이나 시뮬레이션을 통한 검증이 어려움
- 네트워크를 통해 연동되거나 다양한 벤더에 의해 개발된 다양한 버전의 소프트웨어가 존재하는 복잡한 사용 환경을 갖는 소프트웨어의 경우 운용 환경에서 받는 영향에 대한 예측이 어려울 뿐만 아니라 테스트를 수행한다고 해도 복잡한 환경 구성이 어려움

이러한 소프트웨어 개발 초기 단계에서의 검증 작업이 갖는 어려움을 고려하여, 본 논문에서는 초기 디자인 단계(preliminary design stage)에 적용 가능한 검증 방법을 제안한다. 본 논문에서는 초기 디자인 단계에서의 검증 작업을 위한 모델링과 시뮬레이션에 Ptolemy(Brooks 등, 2008)를 사용하며, Ptolemy 상에 검증 대상 모델을 소프트웨어 요구 사항을 바탕으로 작성하고 이의 시뮬레이션을 통해 모델을 검증하도록 한다. 이와 같은 접근 방법의 실용성 확인을 위한 실험에는 DNS 시스템의 행동 양식(behavior) 정보가 반영된 모델이 이용되며, 이 모델의 시뮬레이션을 통해 정의된 모델 내의 불완전 요소 혹은 오류의 검출을 시도한다.

2. 관련 연구

소프트웨어의 검증을 위한 연구는 다양한 분야에서 다양한 기술을 바탕으로 활발하게 진행되고 있다. Tsai (2005) 등은 배포된 웹 서비스의 신뢰성 검증을 위한 방법으로 기술서(specification)만을 사용하는 V&V(verification and validation) 방법을 제안했다. 제안된 방법은 웹 서비스의 특성상 소스 코드 공개가 이루어지지 않는다는 사실에 바탕을 두고 있으며, 또한 전통적인 소프트웨어 검증 방식인 IV&V(Independent V&V) 대신 소프트웨어 개발, 배포, 사용자를 포괄하는 CV&V(Collaborative V & V)를 개념을 가진다. 한편 Cruickshank 등은 안전성이 강조되는 시스템에 대한 검증을 위해 안전성 관련 요구사항을 검증하는 프록시를 가지는 프레임워크를 제안하였다. 이 프레임워크는 사용자가 요구하는 '안전성'에 대한 정의를 명확히 하여, 위험 요소 정의, 분석, 요구사항 추적을 통해 검증의 수준을 높이는 방법을 이용해 소프트웨어의 안전성을 확보를 시도하였으며, 제안된 내용을 군용 프로그램에 반영하여 실험하였다(Cruickshank 등, 2009). 그러나 Tsai의 연구에서는 이미 개발이 완료되어 배포된 소프트웨어(웹 서비스)와 새로운 서비스의 통합과정에서 고려될 수 있는 검증 정책을 제안하고 있고, Cruickshank의 연구에서는 평가 항목 설정을 통해 시뮬레이션이 아닌 검증팀이 직접 평가를 수행한다는 점에서 본 논문에서 초점을 두는 디자인 단계에서의 시뮬레이션 기반 검증 방법과 차이를 가지고 있다.

또한 Seybold 등(2005)은 반정규(semi-formal) 요구사항 모델의 시뮬레이션을 바탕으로 한 소프트웨어 개발 프로세스의 초기 단계에서 수행 가능한 검증 및 오류지점 탐지(defect localization) 방법을 제안하였는데, 이 연구에서는 소프트웨어의 요구사항을 UML 및 상태 차트(state chart) 등을 이용해 모델링하고, 이를 자체 연구를 통해 개발한 모델링 언어이자 모델링 및 시뮬레이션 도구인 ADORA(Glinz 등, 2002)를 이용하여 시뮬레이션을 수행하며, 모델의 반복적인 시뮬레이션을 통해 요구사항의 검증뿐만 아니라 테스터와의 상호 작용 등을 통해 오류지역의 탐지까지 시도한다. 이 연구는 본 논문과 지향하는 목표점은 동일하지만, 본 논문에서 사용하는 도구에 비해 모델의 표현 방식이 제한적이며, 다양한 도메인 및 시스템을 대상으로 수행된 검증이 상대적으로 빈약하다는 차별성을 가진다.

Ptolemy는 UC Berkeley에서 Ptolemy 프로젝트를 통

해 개발한 자바(Java) 기반의 소프트웨어 프레임워크로, 아날로그와 디지털 기술 및 하드웨어와 소프트웨어 등 복합 기술이 적용되는 네트워크 및 신호 처리 분야와 같은 다양한 환경에 존재하는 복잡한 시스템의 모델링, 시뮬레이션 및 병행(concurrent) 시스템의 디자인 지원을 목적으로 한다(Brooks, 2008). Ptolemy에서 정의하는 각 액터(actor)는 통상적인 객체와 유사하게 인터페이스를 통해 액터의 내부 상태를 추상화하고, 동작을 기술하며, 이를 통해 외부 액터와 통신을 수행하도록 하여 계층 구조를 가지는 모델을 쉽게 도출할 수 있도록 한다. 또한 본 논문에서는 현재 Ptolemy가 제공하는 연산 모델(model of computation, domain) 가운데 네트워크 기반 컴퓨터 시스템 모델 작성에 가장 적합한 것으로 파악되는 PN(Process Network) 도메인을 이용하여 모델을 작성하고 시뮬레이션을 수행한다.

3. 모델 행동 양식 검증

초기 디자인 단계에서 수행되는 검증 작업의 난점은 앞서 기술한 바와 같이 자연어로 정의된 요구사항의 처리 및 시뮬레이션이 어려울 뿐만 아니라 요구사항을 모델로 기술하는 과정에서 문맥의 오인으로 인한 오류가 내재될 가능성이 있다는 것이다. 따라서 요구사항 기술 방법이 자연어가 아닌 정형화된 규정을 따르고, 이 규정이 컴퓨터에 의해 처리되고 시뮬레이션 될 수 있다면 가장 좋은 방법이겠으나, 현실은 그렇지 않기 때문에 가능한 소프트웨어 개발 초기 단계에 요구사항을 정형화된 모델로 표현하고 이를 시뮬레이션 하는 것이 바람직할 것이다.

따라서 본 논문에서는 소프트웨어 개발 프로세스 중 요구사항 정의 바로 다음 단계인 초기 디자인 단계에서 모델 및 시뮬레이션을 지원하는 도구를 이용하여, 정의된 요구사항을 정형화된 형식을 갖는 모델로 표현하고, 작성된 모델을 이용해 바로 시뮬레이션을 수행하도록 하여 검증 작업을 수행한다.

본 논문에서 제안하는 초기 디자인 단계의 모델 행동 양식 검증 방법은 그림 1에 나타난 것과 같은 절차를 가진다. 검증 절차는 크게 '모델 디자인'과 '시뮬레이션'의 두 단계로 구성되며, 반복된 시뮬레이션을 통해 요구사항에 대한 디자인 단계에서의 검증과 초기 디자인 모델의 검증을 동시에 수행한다. 우선 모델 디자인 단계 내의 '행동 양식 정의' 단계에서는 요구사항에 기술된 개발 대상 시스템의 동작 방식 및 타 시스템과의 상호 작용 등에 대한 정의가 이루어진다. 이 단계에서는 소프트웨어의 행동 양식만을 기술하기 때문에 소프트웨어에 요구되는 고유의 자료구조나 데이터베이스 사용과 같은 데이터 처리 부분은 표현 가능한 구조로 간략화 되어 정의될 수 있으며, 이 때 주로 다루어지는 소프트웨어의 행동 양식은 다음과 같은 사항들이다.

- 소프트웨어의 상태 전환(status transition) 처리 루틴
- 소프트웨어의 상태 정보(status information)
- 입력 및 출력 이벤트 처리 루틴
- 간략화 하여 표현된 주요 자료 구료
- 실시간 처리가 요구되는 경우 마감 시간(deadline)

요구되는 행동 양식의 정의가 완료되면, 작성된 행동 양식을 바탕으로 Ptolemy가 제공하는 액터(actor)를 이용하여 모델을 작성하게 되는데, 이는 요구사항이 가질 수 있는 모호함을 정형화를 통해 배제하는 효과를 가지도록 하지만, 이 과정에서 발생 가능한 요구사항의 불완전한 해석에 의한 모델 내 오류 포함에 유의하여야 한다.

이와 같은 과정을 통해 작성된 행동 양식 기반 모델은 Ptolemy 상에서 바로 시뮬레이션이 수행될 수 있으나, 현재까지의 모델은 소프트웨어의 정상적인 운용 상태를 가정하고 작성된 모델이므로 여기에 다양한 입력이 부여될 수 있도록 추가 요소를 정의하여 시뮬레이션 셋(simulation set)을 생성한다. 시뮬레이션 셋에는 소프트웨어의 정상적인 운용을 유도하는 데이터 이외에도 고려 가능한 모든 예외적인 상황을 유발하는 구성 요소들이 포함될 수 있다. 예외 상황을 유발하는 데이터의 추가는 요구사항 정의 과정에서 미처 고려되지 못한 예외 상황에서의 소프트웨어의 반응과 상태를 판단하는데 도움을 주게 되며, 모든 시뮬레이션 셋이 정의되면 이를 이용해 모델에 대한 시뮬레이션을 수행한다.

시뮬레이션 과정에서 나타나는 결과의 오류 포함 여부는 사전에 정의된 요구사항 기반 오라클을 기초로 하여, 시뮬레이션 과정에서 생성한 로그 분석이나 오류 발생 시 해당 시뮬레이션을 중단하도록 하는 이벤트 처리 루틴을 이용해 판단할 수 있도록 한다.

수행된 모든 시뮬레이션에서 오류 보고가 발생되지 않으면, 작성된 모델이 세부 디자인 단계(detailed design)에서 사용되도록 한다. 하지만, 시뮬레이션 과정에서 오류가 발견되는 경우에는 해당 오류가 요구사항의 오류인지 아니면 행동 양식 모델의 오류인지를 판단하는 과정을 거친다. 이 때 만약 오류가 요구사항으로부터 기인한 것이라

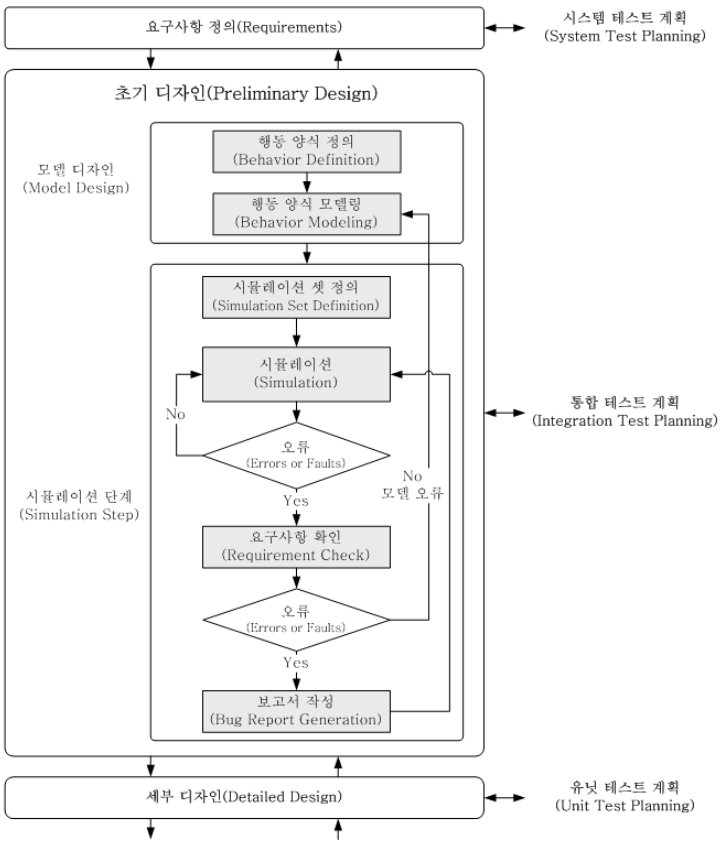

그림 1. 초기 디자인 단계의 검증 절차

판단되면 발견된 각 오류에 대한 오류 보고서를 작성하고 이를 취합하여, 모든 시뮬레이션이 종료된 후 요구사항의 수정 작업에 수집된 오류 보고서가 이용될 수 있도록 한다. 한편 모델 작성 과정상의 오류가 문제를 야기한 경우에는 행동 양식 모델이 수정될 수 있도록 하여, 요구사항과 모델의 검증을 시뮬레이션 과정을 통해 동시에 수행한다.

이와 같이 소프트웨어 개발 프로세스의 초기에 이루어지는 신중한 검증 작업은 비록 기존 방법에 비해 검증에 소요되는 비용 및 시간을 증가시킬 수 있으나, 이후 단계 혹은 개발 완료 후 개발 후 발견되는 오류의 수정에 소요되는 노력 및 비용을 고려할 때 충분한 가치를 가질 것으로 예상된다.

4. DNS 서버의 행동 양식 모델 검증

본 논문에서 제안된 초기 디자인 단계에서의 검증 프로세스의 가용성 확인은 DNS(Domain Name System) 서버와 이의 운용환경을 이용한다. DNS 서버가 사용되는 네트워크 환경은 그림 2에 표현된 것과 같이 도메인 네임 검색이 이루어지는 간략하고 정형화된 형태의 구조를 갖도록 하였다. 구성된 네트워크는 라우터를 중심으로 내부(local) 및 외부(remote) 네트워크로 구성되고, 각 네트워크에는 DNS 서버와 네임 검색 대상인 일련의 서버가 존재한다. 기준이 된 DNS 서버의 모델은 BIND 8의 행동양식이고, DNS 서버가 가지는 데이터베이스에는 도메인 네임 리소스 보관을 위한 리소스 레코드 테이블과 타 DNS 서버와 메시지 교환 시 메시지의 진위 파악을 위해 사용되는 데이터를 보관하는 트랜잭션 테이블을 포함한다. 모델의 단순화를 위해 DNS 클라이언트는 내부 네트워크에만 배치하였으며, 모든 메시지는 라우터를 경유해 교환된다. 이 때 메시지의 분실은 고려되지 않으며, 각 메시지가 타 시스템에 전달되는데 소요되는 지연 시간은 0~약 300ms이라고 가정하였다.

모델 내에서 사용하는 DNS 질의/응답 메시지는 그림 3에 나타난 것과 같은 형식을 갖는다. 메시지 구조는 IP 헤더와 UDP 헤더 중 DNS 질의 및 응답에 필요한 부분만을 취하여 L3(Layer 3, Network Layer) 헤더를 구성하고, 나머지 부분은 실제 DNS 메시지 형식을 그대로 적용하여 모델에서 사용하는 메시지 오브젝트를 구성한다. 메시지에 포함된 모든 필드들의 인식 및 처리에는 실제 DNS 서버가 해당 필드를 처리하는 방식을 적용한다.

표 1은 모델 내의 DNS 서버의 행동 양식에 대한 의사 코드를 나타내고 있다. 세부 행동 양식은 중복된 클라이언트의 질의 메시지 각각에 대해 개별적으로 타 DNS로 질의를 생성해 전달하는 BIND 8을 기준으로 작성하였고 (Householder 등, 2002; SAI, 2003; US-CERT, 2008), DNS 서버의 데이터베이스에 존재하는 두 개의 도메인 네임 검색 관련 테이블은 간략화 하여 서버 내에 정의된 자료 구조로 대치하였으며, 여기에는 질의 및 응답에 반드시 필요한 내용을 저장하도록 구성하였다. 즉, 리소스 레코드를 저장하기 위한 테이블에는 도메인 네임과 해당 도메인의 IP 주소 및 주소를 보관해야 하는 시간이 기록되는 TTL(Time To Live) 값 등 주요 서비스를 제공하는 데 필요한 주요 데이터가 저장되며, 내부 도메인 네임과 외부 도메인 네임(cache)이 모두 하나의 테이블에 저장되도록 간략화 하여 구성하였다. 이 때, TTL 값을 처리하는 과정에서 해당 DNS 서버가 속한 내부 호스트의 도메인 네임은 처리 대상에서 제외하여 해당 호스트의 권한 DNS(authoritative DNS)가 지속적으로 보관될 수 있도록 하였다. 트랜잭션 관리를 위한 테이블에는 질의를 보내온 클라이언트의 IP 주소와 포트 번호 및 트랜잭션 ID 그리고 타 DNS에 질의 전달 시 사용한 DNS 서버의 포트 번호를 저장하도록 한다. 이 트랜잭션 테이블에 저장된 각 정보는 타 DNS 서버에서 보내온 응답의 진위 여부 파악과 응답 메시지의 클라이언트에 대한 전달에 이용된다.

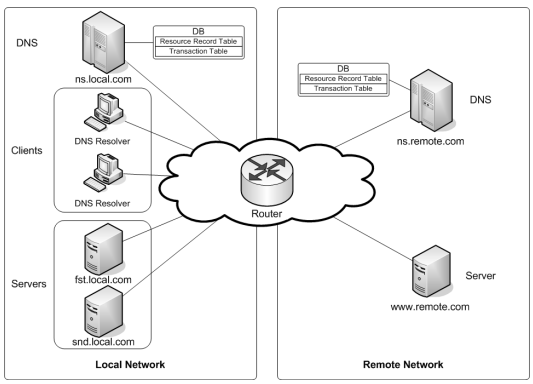

그림 2. DNS 시뮬레이션 네트워크 환경

L3 Header	Sndr Addr	Rcvr Addr	Sndr Port	Rcvr Port
DNS Header	ID	...		ARCOUNT
Question	QNAME	QTYPE		QCLASS
Answer	ANNAME	...		ANRDATA
Authority	AUNAME	...		AURDATA
Additional	ADNAME	...		ADRDATA

그림 3. 모델에서 사용하는 DNS 메시지 구조

표 1. DNS 서버의 의사 코드(pseudo code)

```
RR : A resource record in DNS resource record table
tID : Transaction ID in transaction table
portNo : UDP port number in transaction table

... each scheduled time for DNS actor ...

if no message on input interface
  return
else {
  for each RR in the cache {
    update TTL
    if TTL is expired
      eliminate the RR from the cache
  }
  take message from input interface
  if message is a query {     // QR == 0
    if message from the local network {
      if DNS has the RR
        response with the IP & return
      else
        forward the query to remote DNS & return
    }
    else {
      if DNS has the RR {
        response with IP & return
      else
        response with negative answer & return
    }
  }
  else {              // response : QR == 1
    if message has correct tID and portNo {
      if DNS has not the RR
        make the RR from message
      forward message to client
      return
    }
    else
      return
  }
}
```

그림 4는 그림 2에 제시된 네트워크 환경이 Ptolemy 상에 표현된 것이다. 작성된 네트워크 모델에는 DNS 서버 외에도 DNS 서버에 DNS 쿼리를 전송하는 클라이언트와 모델 내에서 발생되는 모든 메시지를 메시지 내의 목적지로 전달하는 기능을 수행하는 '인터넷'(Internet) 액터가 포함된다. 이와 같이 통상적인 기능을 수행하는 액터 이외에 모델의 검증을 위한 목적으로 다양한 종류의 메시지를 생성하여 모델 내의 타 시스템에 전송하는 기능을 수행하는 가상 클라이언트 혹은 가상 DNS(Illegal Msg. Gen.)를 추가하여, 시뮬레이션 셋에 포함되는 하나의 시뮬레이션 모델을 구성하였다. 이 가상 클라이언트나 가상 DNS 서버가 전송하는 메시지는 정상적인 메시지 이외에도 타 시스템을 가장하거나 연속적인 공격 메시지 전송 등을 수행 가능하도록 구성하여 시뮬레이션의 목적에 부합시킨다.

그림 5는 인터넷 액터의 내부 구조를 보이고 있다. 모델을 구성하는 모든 액터에서 전달되는 메시지는 인터넷 액터에서 제공하는 인터페이스를 통해 다른 액터로 전달되며, 메시지의 분배 기능은 파이선 스크립트 액터를 이용해 작성된 라우터 액터에 의해 수행된다. 라우터 액터는 비 결정적 합병(non-deterministic merge) 액터로부터 전달되는 메시지를 차례로 파싱하여 메시지 내의 목적지 주소를 확인하고, 확인된 주소에 따라 라우터 액터에 연결된 적절한 인터페이스로 내보내는 역할을 수행한다. 또한 인터넷 액터로 전달되는 모든 메시지는 인터넷 액터의 각 입력 인터페이스 뒤에 위치한 임의 지연 시간 생성기(random delay generator)를 거치면서 0~약 300ms 사이의 임의의 지연 시간을 갖게 된다. 임의 지연 시간 생성기의 내부 구조는 그림 6의 형태를 취하며, 가우시안 분포에 따라 임의의 수를 생성한 후, 이를 적정한 수치로 조절하는 기능을 수행한다.

클라이언트 액터는 그림 7에 나타난 것과 같은 형태를 가지며, 클라이언트에서 수행되는 DNS 질의 기능은 Ptolemy에서 제공하는 파이선 스크립트 액터로 작성되었다. 클라이언트 액터에서는 DNS 서버로 받은 응답을 캐싱하여 자체의 TTL 시간동안 보관하도록 하였고, 존재하지 않아 알 수 없는 호스트에 대한 데이터(negative caching)는 저장하지 않는 것으로 가정하였다. 그 외의 클라이언트 액터에 부수적으로 포함되는 액터는 클라이언트에서 주고받는 메시지를 파싱하여 콘솔에 제시해주는 기능을 수행하는 메시지 변환기(message converter)와 질의의 대상이 되는 임의의 호스트 선택을 위한 액터 등이 있다.

5. 시뮬레이션 및 실험 결과

시뮬레이션을 위한 환경은 다음과 같이 구성한다. 우선

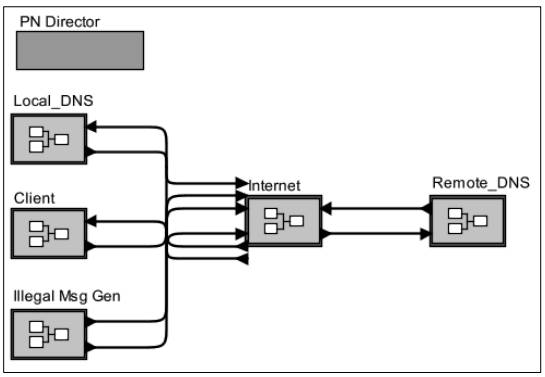

그림 4. Ptolemy로 표현된 네트워크 모델

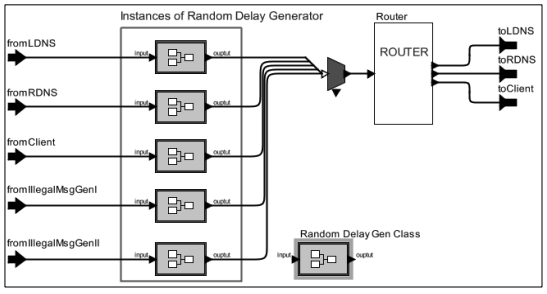

그림 5. 인터넷 액터

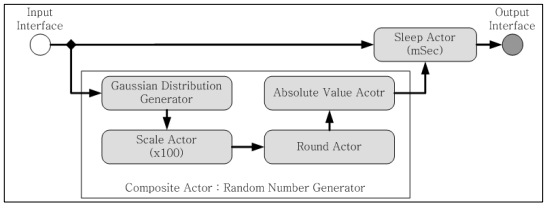

그림 6. 임의 지연 시간 생성기 구조

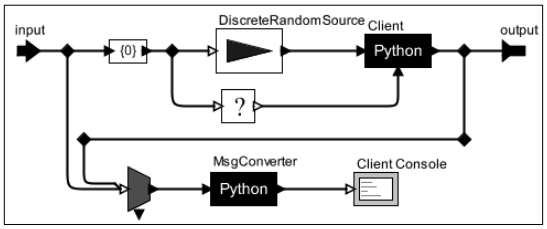

그림 7. 클라이언트 액터

클라이언트는 주어진 네 개의 호스트 가운데 임의의 하나를 선택하여 DNS 서버에 질의를 수행하는데, 이 때 호스트 이름은 내부 및 외부 네트워크에 위치한 호스트와 존재하지 않는 호스트 즉, 잘못된 이름을 가진 호스트(unknown host)를 포함한다. 질의를 수행하는 과정에서 클라이언트에 음성(negative) 캐쉬가 존재하지 않으므로, 존재하지 않는 호스트에 대한 질의가 상대적으로 더 빈번하게 발생하게 된다. 한편 가상 클라이언트와 가상 DNS 서버는 DNS 캐쉬 오염(cache poisoning) 시뮬레이션을 목적으로 구성하였다. 따라서 가상 클라이언트는 특정 호스트에 대한 쿼리를 DNS 서버의 응답과 무관하게 지속적으로 전송하는 기능을 수행하며, 가상 DNS 서버는 자신을 특정 호스트에 대한 권한 DNS(authoritative DNS) 서버로 위장하여 해당 호스트에 대한 잘못된 정보를 내부 DNS 서버의 캐쉬에 삽입하려고 시도한다. 이와 같은 가상 DNS 서버의 시도가 성공되는 경우, 내부 DNS 서버는 자체 캐쉬에 특정 호스트에 대한 잘못된 주소를 캐싱하게 되고, 해당 도메인의 주소를 요구하는 클라이언트의 질의에 오염된 데이터를 기반으로 응답을 수행할 것이다.

그림 8은 빠른 결과 확보를 위해 편의상 DNS 서버 리소스 레코드의 TTL을 5초로 설정하고 모델을 시뮬레이션 하였을 때 클라이언트의 콘솔에 제시되는 실험 결과 중 일부를 나타난 것이다. 클라이언트 콘솔에 제시되는 결과는 클라이언트 액터 내에 위치한 메시지 변환기에 의해 모델 내에서 사용한 IP 주소 대신, 인지가 쉬운 문자열 형태로 변환되어 출력된다.

결과를 살펴보면 내부 DNS 서버는 클라이언트의 질의에 정상적인 응답을 보내오다가, 어느 정도 시간이 흘러 가상 DNS 서버의 시도가 성공하게 되면 클라이언트의 질의에 대해 오염된 내용(IP of Redirection)으로 응답을 보내오게 된다. 이와 같은 오염 현상은 DNS 서버에 설정된 TTL 값에 따라 일정시간 지속된다.

시뮬레이션 결과에 따르면 작성된 모델의 요구사항은 DNS 서버 사이의 메시지 진위 확인에 쉽게 추정 가능하

그림 8. 실험 결과 예제

거나 반복된 단순 시도에 무능화 될 수 있는 수준의 판별 방법을 정의하고 있었고, 이에 따라 비정상적인 메시지를 처리 & 수용하는 행동 양식의 오류를 포함하고 있다는 것이 확인 되었다. 이와 같은 사실은 널리 알려진 DNS 서버가 가지는 취약성 중 하나가 요구사항의 검증을 위한 모델의 시뮬레이션을 통해 발견될 수 있음을 의미한다.

DNS 서버의 경우 오류가 발견이 되어도 시스템 운용 특성상 패치를 위해 가동을 중지하는 것이 쉽지 않은 것이 현실이며, 또한 소수의 DNS 서버 패치로는 궁극적인 문제의 해결이 이루어지지 않는다는 점에서 요구사항 및 모델 검증의 중요성이 다시 한 번 강조될 수 있다.

6. 결론 및 향후 연구

본 논문에서는 소프트웨어의 요구사항 및 모델 검증을 위한 한 방법으로, 소프트웨어의 행동 양식을 반영한 모델을 작성하고 이를 시뮬레이션 하는 방법을 시도하였다. 실험 결과에 따르면 소프트웨어의 요구사항에 정의된 고유한 행동 양식(behavior)을 반영한 모델의 시뮬레이션으로도 요구사항이 가진 잠재적 오류의 검증이 가능하다는 것을 확인할 수 있었으며, 이와 같은 결과는 더욱 정교한 모델을 작성하여 시뮬레이션이 수행되는 경우, 기존에 알려지지 않은 요구사항이나 모델에 내재된 오류도 발견될 가능성이 있음을 시사하기 때문에 의미가 있다고 할 수 있다. 또한 본 논문의 실험에서 제시한 모델에서는 다양한 버전을 가진 소프트웨어가 고려되지 않았으나, 필요한 경우 해당 버전에 대한 액터만 구현하여 기존의 액터와 교체 혹은 연동하도록 구성 가능 하므로 다양한 환경과 복잡한 시스템에 대한 모델링이 용이한 특징을 갖는다.

향후 연구로는 소프트웨어의 버전별 모델이나 벤더별 특징을 갖는 모델을 추가하고, 실제 네트워크 기반 시스템들을 보다 정교하게 구성하는 작업을 수행하고, 시스템들 간의 상호관계에서 발생되는 오류가 있는 경우에 시스템이나 사회적으로 심각한 영향을 줄 수 있는 인프라 시스템에 사용되는 소프트웨어 및 네트워크 프로토콜을 대상으로 시뮬레이션을 수행하여, 제안된 방법의 적용 대상 확장 및 개선 방향 등을 확인할 예정이다.

참 고 문 헌

1. Brooks, C., Lee, E.A., Liu, X., Neuendorffer, S., Zhao, Y., and Zheng, H., Heterogeneous Concurrent Modeling

and Design in Java, vol. 1-3, UCB/EECS-2008-28, 29, 37, EECS Dept. UC Berkeley, 2008.
2. Cruickshank, K. J., Michael, J. B., and Shing, M., "A Validation Metrics Framework for Safety-Critical Software-Intensive Systems," Proc. of the 2009 IEEE International Conference on System of Systems Engineering, NM, USA, pp. 1-8, 2009.
3. Glinz, M., Berner, S., and Joos, S., "Object oriented modeling with ADORA." Information Systems, vol. 27, Issue 6, pp. 425-444, 2002.
4. Householder, A and King, B, Securing an Internet Name Server, CERT®Coordination Center, 2002.
5. IEEE, Glossary of software engineering terminology, IEEE Std. 610.12-1990, IEEE, pp. 80, 1990.
6. IEEE, Guide for developing system requirements specifications, IEEE Std. 1233-1998, IEEE, 1998.
7. IEEE, Software verification and validation, IEEE Std. 1012-2004, pp. 5-11, 2004.
8. Mogyorodi, G., "Requirement-based Testing: An Overview," Proc. of the 39th International Conference and Exhibition on Technology of Object-oriented Languages and Systems, CA, USA, pp. 286-295, 2001.
9. Security Associates Institute, Attacking the DNS Protocol-Security Paper v2, SAI, 2003.
10. Seybold, C. and Meier, S., "Simulation-based Validation and Defect Localization for Evolving, Semi-Formal Requirements Models," Proc. of the 12th Asia-Pacific Software Engineering Conference, Taipei, Taiwan, pp. 408-420, 2005.
11. Tsai, W., Chen, Y., and Paul, R., "Specification-Based Verification and Validation of Web Services and Service-oriented Operating Systems," Proc. of the 10th IEEE International Workshop on Object-oriented Real-Time Dependable Systems, AZ, USA, pp. 139-147, 2005.
12. US-CERT, Multiple DNS implementations vulnerable to cache poisoning, Vulnerability Note VU#800113, United States Computer Emergency Readiness Team, 2008.

신 승 훈 (sihnsh@ajou.ac.kr)

2000 아주대학교 정보 및 컴퓨터공학부 학사
2002 아주대학교 정보통신공학과 석사
2002～현재 아주대학교 정보통신공학과 박사과정

관심분야 : 소프트웨어 테스트 자동화, 멀티미디어 서비스 정책 등

박 승 규 (sparky@ajou.ac.kr)

1974 서울대학교 응용수학과 학사
1976 한국과학원(KAIST) 전산학과 석사
1982 Institut National Polytechnique de Grenoble 전산학과 박사
1992～현재 아주대학교 정보 및 컴퓨터공학부 교수

관심분야 : 임베디드 테스팅, 자가 컴퓨팅/치료 시스템, 차세대 컴퓨터 구조 등

최 경 희 (khchoi@ajou.ac.kr)

1976 서울대학교 수학교육과 학사
1979 프랑스 그랑데꼴 ESEEIHT 석사
1982 프랑스 Paul Sabatier 대학 정보공학과 박사
1982～현재 아주대학교 정보 및 컴퓨터공학부 교수

관심분야 : 운영체제, 분산시스템, 실시간 및 임베디드 시스템 테스팅 등