KIPS Tr. Software and Data Eng.
Vol.3, No.5 pp.169~178 pISSN: 2287-5905

http://dx.doi.org/10.3745/KTSDE.2014.3.5.169

Decentralized LTL Specifications for Ensuring Quality of Interaction-centralized System

Ryoungkwo Kwon† · Gihwon Kwon††

ABSTRACT

In this paper, we present a research utilizing decentralized LTL specifications for ensuring a quality for interaction-centralized system. In this system, for ensuring the quality, we need to validate interactions between modules of the system and then we should check whether the system achieves the expected requirements. This task remains difficult and labor-intensive and requires an expert. In this paper, we present a method to assist such a task. First of all, the requirements of the system is written as multiple LTL specifications. Interactions between modules mean that behaviors of one module are related with other one's behavior. We generate the automaton model fully achieving specification through GR(1) synthesis. And we simulate them using the simulator based on the software agent for checking behaviors of the system. Finally, we validate the whole system whether it achieves given requirements.

Keywords : Decentralized Specification, Generalized Reactivity(1) Synthesis, Simulation, Formal Method

상호 작용 중심 시스템의 품질 확보를 위한 LTL 분산 명세

권 령 구† · 권 기 현††

요 약

본 논문에서는 상호 작용 중심 시스템의 품질을 확보하기 위하여 LTL 분산 명세를 활용하는 연구를 소개한다. 이러한 시스템의 품질 확보를 위해서는 모듈 간의 상호 작용을 확인하여 기대하는 요구 사항을 달성하고 있는지를 검사해야 한다. 이 작업은 어렵고 노동 집약적이며 전문가를 필요로 한다. 따라서 본 논문에서는 이 검사에 도움을 주기 위한 방법을 제안한다. 먼저, 시스템의 기대하는 요구 사항은 각 모듈별로 분리해서 명세한다. 그리고 모듈 사이의 상호 작용은 다른 모듈이 수행하는 행위가 특정 모듈의 행위와 관련 있음을 의미한다. 여기서는 GR(1) 합성을 이용하여 명세를 만족하는 오토마타 모델이 생성되고 이것들은 소프트웨어 에이전트 기반의 시뮬레이터를 통해 모델의 행위를 확인하여 시스템이 요구 사항을 달성하고 있는지를 검사한다.

키워드 : 분산 명세, Generalized Reactivity(1) 합성, 시뮬레이션, 정형 기법

1. 서 론

IT 기술이 발전함에 따라 PC가 중심이 되는 시대를 넘어서 사물 인터넷(Internet of things) 시대로 진화하고 있다. 사물 인터넷은 1999년 케빈 에쉬튼에 의해 처음 소개되었다[1]. 이 용어의 의미는 '인간, 사물, 환경 등 모든 사물이 네트워크에 연결되어 언제 어디서나 다양한 장치로 관련 정보를 쉽게 이용할 수 있는 통신망'이다[2]. 또 다른 연구에서는 사물 인터넷을 '서비스, 데이터, 네트워크, 센서의 모임'이라고 정의한다[3]. 이러한 정의에 기반하여 사물 인터넷의 가장 중요한 특징들은 복잡한 상호 작용 중심 시스템이라는 것과 지능(intelligence)에 기반한 작업을 수행한다는 것이다.

사실상 사물 인터넷의 개념을 도입하지 않더라도 이미 상호 작용 중심의 시스템들이 많이 존재한다. 즉, 수많은 모듈들이 하나의 시스템을 구성하고 그 모듈들은 끊임없이 상호 작용을 이룬다. 이 상호작용에서 목표를 달성해야 하는데, 목표는 기대하는 시스템의 요구 사항이다. 이러한 시스템의 품질을 확인하기 위해서는 모듈들의 복잡한 상호 작용을 확인할 필요가 있다. 이러한 작업은 어렵고 노동 집약적이며 전문가를 필요로 한다. 또한 모듈들을 효과적으로 분산하고 유지 보수할 방법이 필요할 것이다.

따라서 본 논문에서는 이러한 특징을 가지는 시스템의 품

※ 이 논문은 경기도의 경기도지역협력연구센터사업의 일환으로 수행하였음 [2013-0548, 소셜 서비스 융합 플랫폼 요소기술 개발 및 산업화].
† 준 회 원 : 경기대학교 컴퓨터과학과 박사과정
†† 종신회원 : 경기대학교 컴퓨터과학과 교수
Manuscript Received : February 20, 2014
First Revision : April 10, 2014
Accepted : April 11, 2014
* Corresponding Author : Ryoungkwo Kwon(rkay8496@kyonggi.ac.kr)

질 확보를 위한 다음과 같은 연구들을 소개한다.

1) 전체 시스템의 요구 사항을 개별 모듈의 명세로 분산하여 작성한다.
2) 정형 기법 중 하나인 Generalized Reactivity(1)(GR(1)) 합성을 이용하여 명세를 달성할 수 있는 모델을 자동으로 생성한다.
3) 시뮬레이터를 개발하여 모델 수준의 시뮬레이션을 수행하고 상호 작용을 확인하여 시스템의 요구 사항을 만족하는지 확인한다.

요구 사항을 모듈마다 분산된 명세로 구성함으로써 명세 수준에서 독립성이 강조되고 특정 모듈의 명세가 변경될 때 다른 모듈에 주는 영향을 줄일 수 있다. 모듈 간의 상호 작용은 최소한의 인터페이스로만 고려한다. 그리고 현재 많은 연구가 활발히 진행 중인 합성 기법을 이용하여 명세를 만족하는 모델을 자동으로 생성한다. 마지막으로 생성된 모델들의 시뮬레이션을 위한 도구를 개발하였다. 이 도구를 활용하여 사전에 모델 수준에서 상호 작용을 검사하고 시스템의 요구 사항을 위반하는지를 확인할 수 있다. 따라서 다양하게 변하는 환경에서 모듈 및 시스템의 품질을 보장할 수 있는 수단을 제공한다.

본 논문의 구성은 다음과 같다. 2장에서는 선형 시제 논리, 합성 기법, 오토마타 모델 정의들을 설명한다. 3장에서는 제안 내용의 핵심인 분산된 명세의 개념과 작성 방법, 모델을 생성 및 개발한 시뮬레이터를 이용한 요구 사항 검사 과정을 설명한다. 4장에서는 제안 내용을 시나리오를 대상으로 전체 과정을 설명한다. 그리고 명세가 변경되었을 때 개별 모델은 정확하지만 상호 작용에 문제를 발생될 수 있음을 시뮬레이터를 통해 확인하는 과정을 소개한다. 5장에서는 본 논문의 결론을 맺는다.

2. 배경 지식

2.1 선형 시제 논리(Linear Temporal Logic)

선형 시제 논리는 선형 시간을 기반하고 있으며, 명제 논리와 몇 개의 시제 연산자를 포함함으로써 명제들의 순서들 상에 시스템의 행위를 표현하는 논리이다[4]. 여기서 시제라는 단어의 의미는 시스템 행위들의 순서 및 관계를 표현하지만, 실제 시스템에서 관찰될 수 있는 행위보다 추상적인 수준에서 사용된다. 즉, 시스템 행위들이 추상화된 순서인 것이다.

선형 시제 논리식 ϕ은 기본 명제들의 집합 $\pi \in$ AP(atomic propositions) 상에서 작성이 되며 구문은 다음과 같이 정의된다.

$$\phi ::= true | false | \pi | \phi | \phi_1 \vee \phi_2 | \neg\phi | \bigcirc \phi | \phi_1 \mathrm{U} \phi_2$$

부정 연산자와 논리합 연산자로 다음의 연산자들을 유도할 수 있다.

1) 논리곱($\wedge$) $\phi_1 \wedge \phi_2 \equiv \neg(\phi_1 \vee \phi_2)$
2) 암시($\Rightarrow$) $\phi_1 \Rightarrow \phi_2 \equiv \neg\phi_1 \vee \phi_2$
3) 동치($\Leftrightarrow$) $\phi_1 \Leftrightarrow \phi_2 \equiv (\phi_1 \Rightarrow \phi_2) \wedge (\phi_2 \Rightarrow \phi_1)$

다음($\bigcirc$)과 언틸(U) 연산자를 통해 다른 시제 연산자들을 유도할 수 있다.

1) 언젠가($\Diamond$) $\Diamond\phi \equiv true \mathrm{U} \phi$
2) 항상($\Box$) $\Box\phi \equiv \neg\Diamond\neg\phi$

선형 시제 논리식 ϕ의 의미는 무한한 상태들의 시퀀스 σ 상에서 정의된다. 각 상태에서는 기본 명제들의 값(참 또는 거짓)이 할당되어 있으며 $\sigma(i)$는 i번째 상태에서 참인 기본 명제들의 집합을 나타낸다. $\sigma(i)$에서 논리식 ϕ가 만족됨은 다음과 같이 재귀적으로 정의될 수 있다. 더욱 자세한 의미는 [4]를 참고한다.

1) $\sigma, i \vDash \pi \Leftrightarrow \pi \in \sigma(i)$
2) $\sigma, i \vDash \neg\phi \Leftrightarrow \sigma, i \nvDash \phi$
3) $\sigma, i \vDash \phi_1 \vee \phi_2 \Leftrightarrow \sigma, i \vDash \phi_1 \vee \sigma, i \vDash \phi_2$
4) $\sigma, i \vDash \bigcirc\phi \Leftrightarrow \sigma, i+1 \vDash \phi$
5) $\sigma, i \vDash \phi_1 \mathrm{U} \phi_2 \Leftrightarrow \exists_{k \geq i} \bullet \sigma, k \vDash \phi_2 \wedge \forall_{i \leq j < k} \bullet \sigma, j \vDash \phi_1$

2.2 Generalized Reactivity(1) 합성

합성이라는 것은 논리식을 통해 기대하는 시스템의 행위들을 작성하고 이 논리식을 달성할 수 있는 시스템을 생성하는 정형 기법 중 하나이다. 그동안 많은 연구자들에 의해 합성을 위한 연구가 진행되어 왔다. Church 문제[5]를 통해 처음으로 합성이 다루어졌고 이후에 많은 방법들이 제안 되었다[6, 7]. 이후 [8]에서 제안한 방법들은 선형 시제 논리식을 이용하여 합성을 만족 가능성 문제로 다루었다. 이 방법은 선형 시제 논리식으로 뷰키 오토마타를 만들게 되고 이 오토마타는 결정적 라빈 오토마타로 변환될 수 있는지 결정된다. 하지만 이러한 과정의 계산 복잡도는 논리식 크기에 대해 지수적으로 증가하게 되었다. 이러한 계산 복잡도에 의해 합성을 이용한 반응형 시스템 개발은 불가능하게 여겨졌다. 하지만, [9]에서 GR(1) 형태의 반응형 시스템 행위 명세를 위한 선형 시제 논리식을 제안하였는데, 이로써 계산 복잡도를 상태 공간에 대하여 다항 시간으로 줄였다.

본 논문에서는 모델 생성을 위하여 GR(1) 합성 기법을 사용하였다. 이 기법은 반응형 시스템의 행위를 대상으로 하고 있다. 반응형 시스템은 기존의 변형 시스템(transformational system)과는 달리 시스템 외부 환경과의 지속적인 상호 작용을 이루는 것이라 할 수 있다[8]. 반응형 시스템은 끊임없이 환경과 상호 작용을 통해 시스템의 목표(기대 행위)를

달성하는 것이라고 할 수 있다.

이 기법에서는 주어진 논리식을 항상 달성할 수 있을 때 기대하는 행위들을 완전히 포함하는 모델을 얻을 수 있다. 이 경우를 실현 가능하다고 한다. 반대로 만족하지 않는다면, 우리가 기대하는 시스템의 행위를 달성할 수 없다는 것을 확인한다. 실현 가능하다는 것의 개념은 모델 검증[4]과 비교하여 설명할 수 있다. 모델 검증은 모델과 속성이 주어졌을 때 모델 상에서 속성이 만족되는지를 확인하는 것이다. 만약 속성을 위반하는 하나의 경로(반례)를 발견하여 모델을 수정하였다 하더라도 탐색하지 않은 모델의 상태 공간이 존재하므로 모델이 완전히 속성을 만족하는 행위를 한다고 볼 수 없다. 반대로 실현 가능성은 모델 검증과 달리 모델이 주어지지 않고 논리식으로 표현된 기대 행위를 언제나 달성할 수 있는(어떠한 경우에서도 기대하지 않는 행위가 존재하지 않는) 모델을 생성하는 것이다. 이것을 실현 가능하다고 한다.

GR(1) 합성 기법은 선형 시제 논리의 구문을 제약한 GR(1) 논리식 $\phi = \phi^e \Rightarrow \phi^s$을 사용한다. 논리식 ϕ^e는 환경의 행위, ϕ^s는 시스템의 행위이다. $\alpha \in \{e,s\}$일 때 하위 논리식들의 논리곱은 다음으로 구성된다.

1) ϕ_α^i: 환경과 시스템의 초기 상태들을 표현하는 이진 논리식이다.
2) ϕ_α^t: 환경과 시스템의 변수들의 전이를 표현하는 논리식이다. 논리식의 형태는 $\Box B$와 같으며 B는 입력 및 출력 변수와 ○연산자의 조합으로 구성된 논리식이다.
3) ϕ_α^g: 환경과 시스템이 무한히 자주 만족할 행위를 표현하는 것으로써, GR(1) 합성에서 승리 조건을 의미한다. 논리식의 형태는 $\Box\Diamond B$와 같으며 B는 이진 논리식이다.

3. 분산된 명세로부터 모델 생성

이번 장에서는 분산된 명세의 개념과 각 명세로부터 생성된 모델들이 어떻게 하나의 시스템처럼 실행될 수 있는지에 대해서 설명한다. Fig. 1은 분산된 명세로부터 모델을 생성하고 시뮬레이션하는 전체 과정이다.

요구사항은 시스템을 개발 과정에서 초기에 반드시 필요한 산출물 중에 하나이다. 요구사항은 기능적인 측면에서 개발 대상의 시스템이 수행해야 하는 행위(작업)들에 대한 추상적인 형태(자연어 혹은 비정형적인 언어)로 기술하고 있다. 일반적으로 시스템은 하나가 아닌 다수의 모듈들로 구성이 된다고 하였을 때 각 모듈별 정형적인 명세로 분산될 수 있다. 이 연구에서는 각 명세는 정형 언어인 선형 시제 논리식으로 작성될 것이다. 이렇게 작성된 각각의 명세는 GR(1) 합성에 의하여 명세를 달성하는 오토마타 모델을 생성할 것이다. 생성된 모델들은 실제 모듈들의 상위 수준 표현으로써 시뮬레이터에서 실행되고 실행 트레이스를 확인할 수 있다.

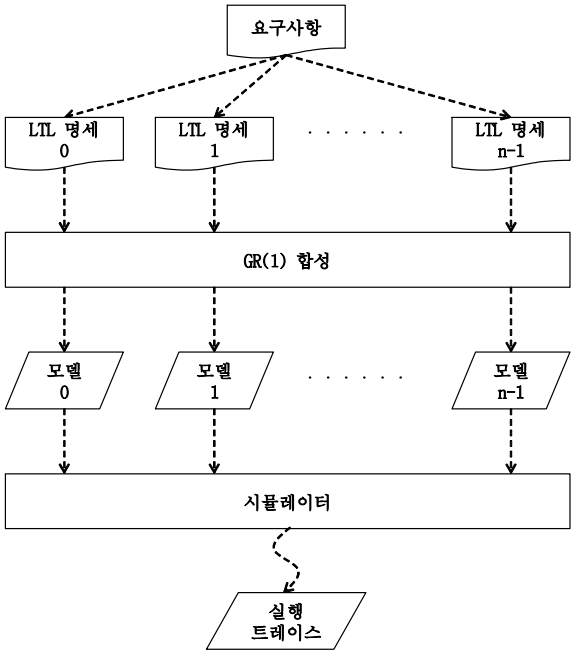

Fig. 1. The overview of a simulation

3.1 분산된 명세

앞서 설명한 것과 같이 본 논문에서 대상으로 하고 있는 시스템은 다수의 모듈로 구성되어 있다. 이러한 시스템에서 각 모듈은 다른 모듈과 상호작용을 통해 요구 사항을 부합해야 한다. 우선 이러한 시스템을 구성하는 요소를 정형적으로 정의하면 다음과 같다. 하나의 시스템은 하나 이상의 모듈들과 상호작용으로 구성되어 있으며 각 모듈은 입력, 출력으로 구성된다. 입력이라고 하는 것은 다른 모듈의 출력으로 정의된다. 출력은 모듈 자신의 액션들로 정의한다. 마지막으로 해당 모듈의 상호 작용은 다른 모듈의 액션과 자신의 입력의 관계로 정의한다. 그리고 입/출력은 참/거짓 값만을 가지는 이진 변수로 정의한다.

1) $Sys = \{MO_0, ..., MO_n, Intr\}$
2) $MO_i = \{In, Out\}$
3) $MO_i^{In} = \{MO_j^{out} | 0 \le j \le n \wedge i \ne j\}$
4) $MO_i^{Out} = \{actions\}$
5) $Intr = \{(MO_j, MO_j^{out}, MO_i, MO_i^{in}) | 0 \le j \le n \wedge i \ne j\}$

하나의 모듈이 수행한 결과, 발생하는 이벤트 또는 메시지들이 다른 모듈에 의하여 받아들여져서 해당 모듈은 자신만의 행위를 수행하게 된다. 상호작용 MO_i^{Intr}을 비유하자면 함수 호출자와 피 호출자의 관계와 유사하다고 볼 수 있다. 호출되는 함수의 행위는 호출자에 있어서는 블랙박스처럼

고려되며 호출할 때 특정 정보(이벤트, 메시지 등)들을 함께 전달한다. 이 정보들은 호출되는 함수에게는 외부의 입력이다.

이러한 상호작용을 반영하여 선형 시제 논리식 명세 ϕ_s은 다음과 같다.

$$\phi^s = \phi^i \Rightarrow \phi^o$$

명세 ϕ^s는 GR(1) 논리식의 형태이고 두 부분으로 나눠진다. 논리식 ϕ^i는 해당 모듈과 상호작용하는 다른 모듈의 행위(외부의 입력 행위)를 나타내고 논리식 ϕ^o는 모듈 자신이 수행해야 하는 행위 그 자체에 대한 것을 나타낸다. 그러나 본 논문에서는 논리식 ϕ^i를 $true$(항상 참)로 고정한다. 그 이유는 다른 모듈의 입력 행위는 블랙 박스로 고려하기 때문에 이 행위가 어떻게 될지는 명세 단계에서 전혀 고려하지 않기 때문이다. 따라서 최종적인 명세 ϕ^s는 다음과 같다.

$$\phi^s = true \Rightarrow \phi^o$$

입력의 행위를 $true$로 제한하였을 때 명세 ϕ^s는 특정 모듈의 행위가 변경되어야 할 때 이점을 가진다. 명세 ϕ^s에 오직 모듈 그 자체의 행위만 작성된다면 명세 자체의 독립성이 유지되므로 이미 작성된 입력의 행위에 따른 비일관성 또는 모순과 같은 문제에 대해 고려할 필요가 없다. 그리고 새로운 모듈을 추가하기 위해서 명세를 작성할 때 우선적으로 해당 모듈 자신의 행위에만 초점을 맞춰서 작성할 수 있기 때문에 명세 작성에 용이하다.

반면에 이러한 명세 형태로 인하여 고려해야 할 측면이 있다. GR(1) 합성 과정의 가장 중요한 작업은 실현가능성 검사이다. 하지만 입력의 행위가 $true$라면 모듈 자신의 행위가 어떻든지 간에 실현 가능하다. 이 점은 위와 같은 명세에서 실현 가능한 모듈들이 상호작용을 통해 하나의 시스템처럼 동작할 때 궁극적으로 시스템의 요구사항을 만족하지 않는 올바르지 않은 행위를 보일 수 있다. 이 문제를 다루기 위해서 모델 수준의 시뮬레이터 도구를 개발하였다. 이 도구를 통해 모델들의 실행 트레이스들을 확인하여 요구사항을 달성하지 못하는 경우를 발견하였다.

3.2 모델 생성 및 시뮬레이션

본 논문에서 GR(1) 합성을 통해 생성된 모델들을 실행하기 위한 시뮬레이터를 개발하였다. 이 도구의 전체 모습은 Fig. 2와 같다.

개발된 시뮬레이터의 기능 및 목적은 다음과 같다.

1) 다수 모델의 행위 확인 및 요구 사항에 부합하지 않는 행위의 발견
2) GR(1) 합성에서부터 에이전트 객체 생성 및 실행의 과정을 자동화

Fig. 2. The screenshot of a simulator

이 중에서 가장 중요하게 생각한 것은 개별 명세들이 GR(1) 합성을 통해 실현 가능하여 오토마타 모델들이 생성되더라도 다른 모델들과 상호작용을 통해 실행하였을 때 문제가 발생할 수 있다는 것이다. 앞서 설명하였던 것처럼 모듈 자신은 요구 사항에 부합하기 위한 행위가 있다 하더라도 다른 모듈과 잘못된 상호 작용이 발생된다면 예측할 수 없는(잘못된) 행위가 일어날 수 있다. 또는 모듈 자신의 행위가 요구 사항에 부합되지 않을 수 있는데 이 경우를 발견할 때에도 용이하다. Fig. 3은 도구를 이용한 시뮬레이션 흐름을 보여주고 있으며, GR(1) 합성부터는 모든 작업들이 자동적으로 진행된다.

개발한 도구에서는 시뮬레이션을 위한 명세 파일들이 필요하다. 해당 명세는 NuSMV 모델 체커에서 사용하는 입력 형식을 따른다. 순수하게 선형 시제 논리식을 작성하는 것은 어려움이 있기 때문에 이 형식을 따르면 명세 작성에 조금 더 용이할 수 있다. 명세 파일에 더불어 상호 작용에 대한 정보를 입력해야 한다. 상호 작용을 입력하는 편집창에서는 한 줄에 4가지 정보(메시지 송신자, 보낼 메시지, 메시지 수신자, 수신 메시지)들을 필수적으로 입력해야 한다. 명세 파일들이 준비가 되면 GR(1) 합성을 통해 각 명세가 실현 가능한지를 자동으로 계산이 된다. 만약 실현 불가능한 명세가 있다면 이 단계에서 확인이 가능하다. GR(1) 합성 알고리즘은 [10]에서 제안된 JTLV 라이브러리를 사용하였다. 모든 명세들이 실현 가능하다면 각 명세에 해당하는 오토마타 모델들이 생성된다. 그리고 처음에 입력하였던 상호 작용 정보와 함께 JADE[11] 플랫폼 상에서 동작하는 소프트웨어 에이전트 객체들을 생성한다. 생성한 모델들은 각 에이전트의 작업 계획 또는 컨트롤러로 포함된다. 즉, 각 에이전트는 때마다 모델의 정보를 해석하여 자신의 행위를 결정한다. 최종적으로 이 도구의 사용자는 각 모델의 실행 트레

이스를 확인하게 된다. 실행 트레이스에는 각 모델에 정의된 이진 변수들의 값이 모델의 상태 전이에 따라 어떻게 변하는지를 보여주고 사용자는 이 정보를 기반으로 하여 전체 시스템이 요구 사항을 부합하고 있는지를 검사할 수 있다.

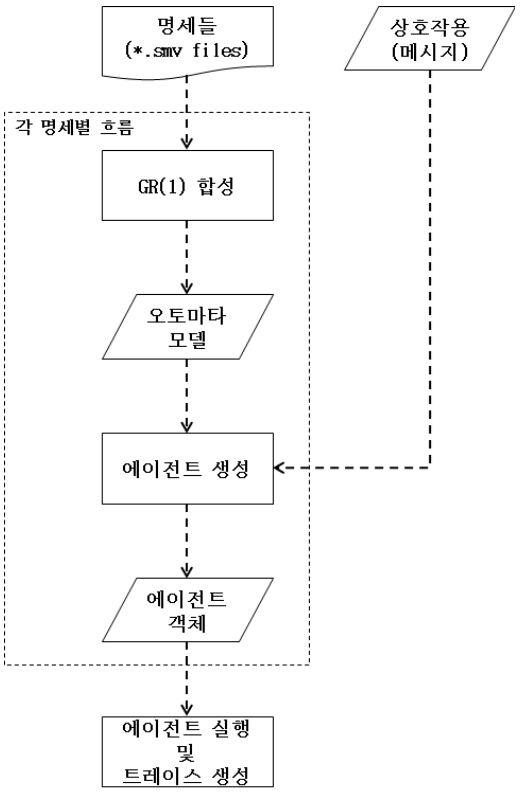

Fig. 3. The flow of a simulation

4. 사례 연구

이번 장에서는 사례 연구를 통해 다수의 시스템 구성 요소들이 상호 작용하면서 요구 사항을 부합하는 시나리오를 소개한다. 또한 시뮬레이터를 통해 요구 사항을 부합하지 못하는 경우를 확인하는 시나리오를 소개한다.

4.1 아비터 시나리오의 요구 사항

첫 번째 시나리오의 모습은 Fig. 4에 묘사되어 있다. 그림으로 볼 수 있듯이 이 예제는 하나의 아비터와 두 개의 로봇(H와 P)으로 구성되어 있다. 그리고 로봇들이 움직일 수 있는 영역은 총 $\{r_1,...,r_8\}$으로 주어진다. 각 로봇은 오직 아비터와 상호 작용을 한다. 이 상황을 하나의 시스템으로 고려하였을 때 다음과 같은 기대하는 요구 사항이 있다.

1) 로봇 H는 영역 $\{r_1,...,r_4\}$를 이동해야 한다.
2) 로봇 P는 영역 $\{r_4,...,r_8\}$를 이동해야 한다.
3) 영역 r_4는 동시에 두 로봇이 위치할 수 없다.
4) 각 로봇은 영역 r_4를 지나가기 위해서는 아비터에게 요청을 해야 하며, 아비터로부터 요청에 대한 승인을 받지 않는 한 해당 요청은 유지한다.
5) 요청에 대한 승인을 받은 로봇만이 언젠가는 영역 r_4를 지나갈 수 있다.
6) 요청에 대한 승인을 받은 로봇은 알람음을 발생한다.
7) 각 로봇은 영역 r_4를 지나가고 나면 요청을 해제한다.
8) 아비터는 로봇이 요청을 해제하기 전까지는 승인을 유지한다.

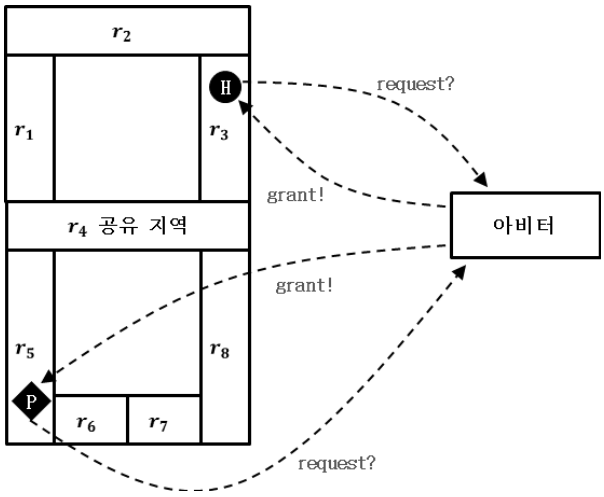

Fig. 4. The setting of an arbiter scenario

4.2 명세 작성 및 모델 생성

앞서 설명한 요구 사항에 따라 아비터, 로봇 H와 P의 총 3개의 명세를 작성해야 한다. 하지만 명세를 작성하기 앞서 이 문제를 형식화한다(특별한 가정이 없는 한 모든 변수들은 이진 값을 가진다). 아래는 아비터의 형식화이다. $request_0$, $request_1$는 참일 때 로봇 H와 P에서 받는 요청을 나타낸다. 그리고 $grant_0$, $grant_1$는 참일 때 로봇 H와 P에게 전달하는 승인을 나타낸다.

1) $X_{Arbiter} = \{request_0, request_1\}$
2) $Y_{Arbiter} = \{grant_0, grant_1\}$

다음은 로봇 H의 형식화이다. 참고로 로봇 P는 H와 영역의 차이만 있으므로 생략한다.

1) $X_H = \{grant\}$
2) $X_H = \{r_1,...,r_4, alarm, request\}$

$grant$는 참일 때 아비터에게 받는 승인을 나타낸다. $\{r_1,...,r_4\}$는 로봇 H가 해당 지역에 위치하면 참이 된다. 그리고 $alarm$은 참일 때 알람음을 울리는 것이고 $request$는 아비터에게 요청을 하는 것이다.

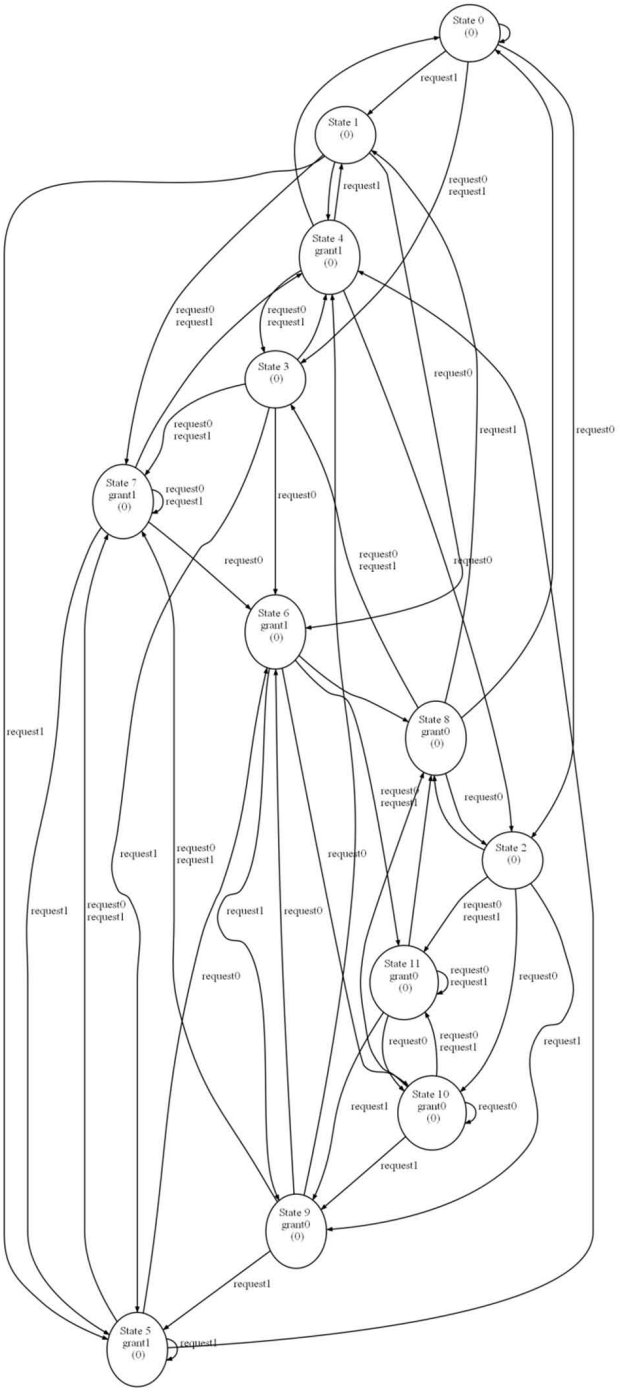

Fig. 5A. The automaton of an arbiter

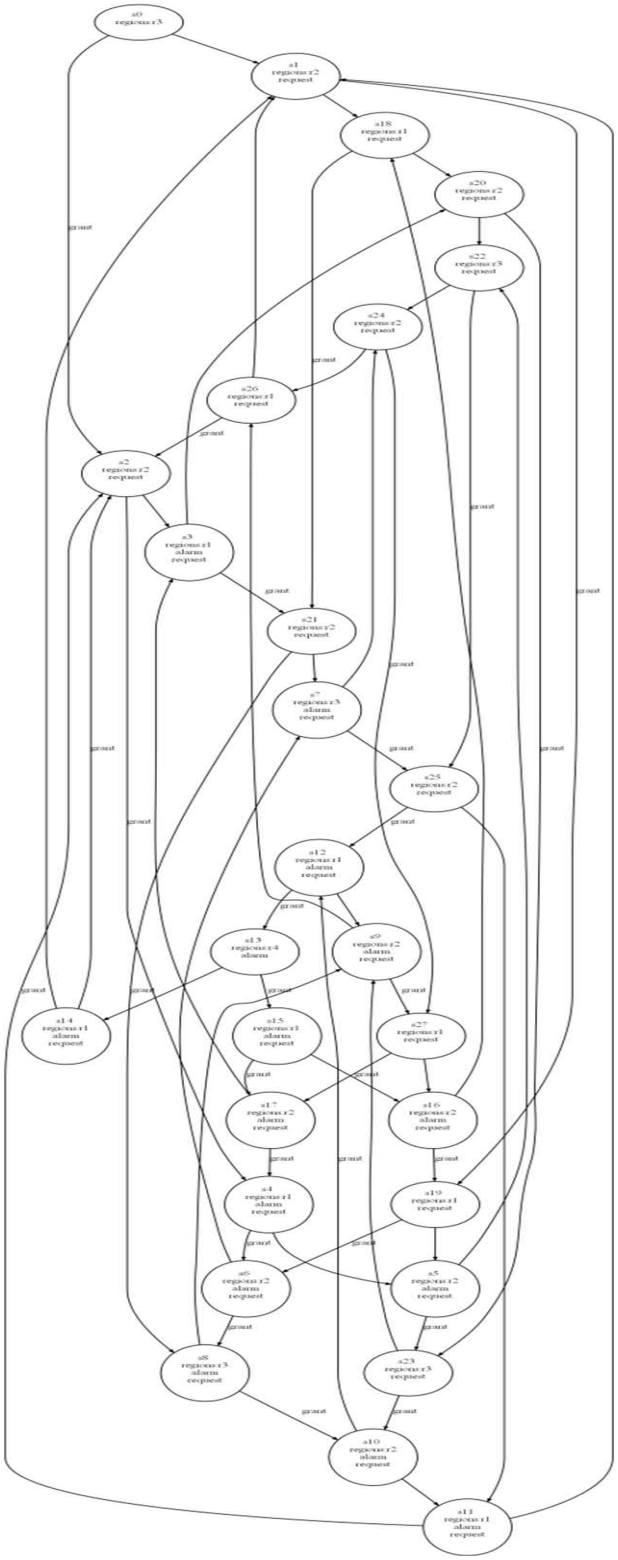

Fig. 5B. The automaton of a robot *H*

다음은 형식화에 기반하여 요구 사항을 부합하는 아비터의 명세 $\phi^s_{Arbiter}$이다.

$$\phi^s_{Arbiter} = \phi^i_{Arbiter} \Rightarrow \phi^o_{Arbiter}$$

$$\phi^i_{Arbiter} = \begin{cases} \bigwedge_{i\in\{0,1\}} \neg request_i \\ \wedge \Box true \\ \Box\Diamond true \end{cases}$$

$$\phi^o_{Arbiter} = \begin{cases} \bigwedge_{i\in\{0,1\}} \neg grant_i \\ \wedge \bigwedge_{i\in\{0,1\}} \Box(\neg request_i \Rightarrow \neg \bigcirc grant_i) \\ \wedge \bigwedge_{i\in\{0,1\}} \Box(request_i \wedge \neg \bigcirc grant_{i-1} \Rightarrow \bigcirc grant_i) \\ \wedge \bigwedge_{i\in\{0,1\}} \Box(grant_i \wedge request_i \Rightarrow \bigcirc grant_i) \\ \Box\Diamond true \end{cases}$$

아비터 명세 $\phi^s_{Arbiter}$에서 $\phi^o_{Arbiter}$의 3행 논리식이 아비터가 로봇과 상호 작용하는 부분이다. 해당 논리식을 풀어서 설명하면 아비터에게 요청이 보내지고 다음에 다른 로봇에게 승인을 하지 않았다면 요청을 한 로봇에게 승인을 해준다.

다음은 로봇 H의 명세 ϕ^s_H이다. 단, 로봇 P의 명세 ϕ^s_P는 H의 명세 ϕ^s_H에 비하여 영역의 차이만 제외하고는 동일하므로 생략한다.

$$\phi^s_H = \phi^i_H \Rightarrow \phi^o_H$$

$$\phi^i_H = \begin{cases} \neg grant \\ \wedge \Box true \\ \Box\Diamond true \end{cases}$$

$$\phi^o_H = \begin{cases} r_3 \wedge \bigwedge_{i=\{1,2,4\}} r_i \\ \wedge \bigwedge_{i=\{1,2,3,4\}} \Box(r_i \Rightarrow \bigcirc r_j \vee \bigcirc r_k) \\ \wedge \Box(\neg grant \Rightarrow \neg \bigcirc r_4) \\ \wedge \Box(\neg grant \Rightarrow \bigcirc request) \\ \wedge \Box(\neg \bigcirc r_4 \Rightarrow \bigcirc request) \\ \wedge \Box(grant \Leftrightarrow \bigcirc alarm) \\ \wedge \Box\Diamond r_2 \\ \wedge \Box\Diamond r_1 \\ \wedge \Box\Diamond r_3 \\ \wedge \Box\Diamond(grant \Rightarrow r_4) \end{cases}$$

로봇 H의 명세 ϕ^s_H에서 ϕ^o_H의 4, 10행 논리식이 로봇 H가 아비터와 상호 작용하는 부분이다. 두 논리식은 로봇 H가 현재 승인받지 않으면 요청을 하고, 승인을 받으면 언젠가는 r_4로 간다는 의미이다.

위 명세들을 GR(1) 합성을 통하여 실현 가능하여 Fig. 5과 같은 오토마타 모델들을 생성하였다. 아비터 오토마타는 총 12개의 상태 수이며 시작 상태는 s_0이다. 로봇 H 오토마타는 총 27개의 상태 수이며 시작 상태는 s_0이다.

4.3 시뮬레이션 결과

생성된 모델들을 개발한 시뮬레이터를 이용하여 시뮬레이션을 수행하기 위해서는 먼저 상호 작용에 대한 정보를 입력해야 한다. 다음은 아비터 시나리오에 해당하는 상호 작용 정보이다. 입력된 상호 작용을 보면 아비터와 로봇들 간에 상호 작용은 로봇들의 요청과 아비터의 승인 사이에 이루어지는 것을 확인할 수 있다.

$$Intr = \left\{ \begin{array}{l} \left(MO_{Arbiter}, MO^{grant_0}_{Arbiter}, MO_H, MO^{grant}_H\right), \\ \left(MO_{Arbiter}, MO^{grant_1}_{Arbiter}, MO_P, MO^{grant}_P\right), \\ \left(MO_H, MO^{request}_H, MO_{Arbiter}, MO^{request_0}_{Arbiter}\right), \\ \left(MO_P, MO^{request}_P, MO_{Arbiter}, MO^{request_1}_{Arbiter}\right) \end{array} \right\}$$

시뮬레이션을 수행하였을 때 우리가 확인한 실행 트레이스의 일부는 다음 Table 1과 같다. 표의 각 열은 오토마타 모델의 상태, 모델의 포함된 변수들이다. 그리고 각 변수들은 참/거짓의 이진 값으로 표현된다. 단, 로봇들의 영역은 이진 값이 아닌 실제 위치로 표현된다. 그리고 이전과 동일한 값을 가지는 값은 생략하였다.

Table 1A. The execution trace of an arbiter

State no	request$_0$	request$_1$	grant$_0$	grant$_1$
0	F	F	F	F
-	-	-	-	-
3	T★	T★	-	-
7	-	-	T★	-
-	-	-	-	-
-	-	-	-	-
-	-	-	-	-
-	-	-	-	-

Table 1B. The execution trace of a robot H

State no	grant	regions	alarm	request
0	F	r_3	F	F
1	-	r_2	-	T★
18	-	r_1	-	-
20	-	r_2	-	-
23	T★	r_3	-	-
10	-	r_2	T★	-
12	-	r_1	-	-
13	-	r_4★	-	F★

Table 1C. The execution trace of a robot P

State no	grant	regions	alarm	request
0	F	r_6	F	F
1	-	r_7	-	T★
21	-	r_8	-	-
23	-	r_7	-	-
25	-	r_8	-	-
27	-	r_7	-	-
29	-	r_6	-	-
31	-	r_5	-	-

위 표들에서 동일한 행을 기준으로 행위를 보았을 때, 별표는 상호 작용에 관련되었다. 모든 로봇은 영역 r_4에 도달하지 못하고 승인을 받기 전까지는 요청을 아비터에게 전달한다. 아비터는 로봇 H에게 승인을 우선 해주고(4행) 로봇 H는 알람음을 울리기 시작(6행)한다. 로봇은 r_4에 도착(8행)하면 그동안의 요청을 해제한다. 반면에 로봇 P는 요청(1-8행)을 계속 하지만 승인을 받지 못하여 영역 r_4로 이동하지 못함을 확인할 수 있다. 이러한 실행 트레이스를 확인하였을 때 앞서 정의한 요구 사항을 충분히 달성하고 있음을 확인할 수 있다.

4.4 요구 사항을 부합하지 못하는 경우

이번 절에서는 앞서 소개한 아비터 시나리오에서 아비터의 명세가 변경되었을 때 전체 시스템의 행위가 요구 사항을 부합하지 않는 경우를 어떻게 발견하는지를 소개한다.

변경되는 명세는 아비터이며 나머지 두 로봇의 명세는 변경되지 않는다고 가정한다. 그리고 형식화도 이전과 동일하다고 한다. 아비터의 명세 $\phi^s_{Arbiter}$에서 아래와 같은 $\phi^o_{Arbiter}$의 논리식이 있었다.

$$\square(request_i \wedge \neg \bigcirc grant_{i-1} \Rightarrow \bigcirc grant_i)$$

해당 논리식이 다음과 같이 변경되었다고 가정할 때, 변경된 논리식의 의미는 단지 요청이 들어오면 해당 로봇에게 승인을 해주는 것이다.

$$\square(request_i \Rightarrow \bigcirc grant_i)$$

변경된 명세 $\phi^s_{Arbiter}$를 GR(1) 합성을 통해 Fig. 6과 같은 오토마타 모델이 생성되었다. 생성된 오토마타는 총 16개 상태수를 가지며 시작 상태는 s_0이다.

시뮬레이션을 수행하였을 때 확인한 실행 트레이스의 일부분은 Table 2와 같으며, 요구 사항을 부합하지 못하는 부분을 보여주기 위하여 실행 트레이스의 중간 부분을 보여주었다. 아비터는 두 로봇 모두에게 요청에 대한 승인(2, 3행)을 하는 행위를 보임으로써, 로봇 H와 P는 이전과 달리 승인으로 인한 영역 r_4를 동시에 지나갈 수 있는 행위(8행)가 발생되었다. 사실 요청에 의한 승인은 아비터의 행위이기 때문에 요구 사항에 부합하지 않는 행위의 발생은 결국 아비터의 책임이라고 확신할 수 있었다.

Table 2A. New execution trace of an arbiter

State no	request$_0$	request$_1$	grant$_0$	grant$_1$
14	F	T	T	F
11	T	-	F	T★
7	-	-	T★	-
-	-	-	-	-
-	-	-	-	-
-	-	-	-	-
5	-	F	-	-
15	-	T	-	F

Table 2B. New execution trace of a robot *H*

State no	grant	regions	alarm	request
15	T	r_1	T	T
19	-	r_2	-	-
13	F	r_1	-	-
21	T	r_2	F	-
8	-	r_3	T	-
10	-	r_2	-	-
12	-	r_1	-	-
13	-	r_4★	-	F

Table 2C. New execution trace of a robot *P*

State no	grant	regions	alarm	request
24	T	r_7	F	T
11	F	r_8	T	-
28	T	r_7	T	-
16	-	r_6	T	-
18	-	r_5	-	-
20	-	r_4	-	F
4	-	r_8	-	T
6	-	r_4★	-	T

5. 결 론

본 논문에서는 시스템이 달성해야 하는 요구 사항을 시스템을 구성하는 모듈별로 분산하여 오토마타 모델을 생성하였다. 그리고 그 모델들을 우리가 개발한 시뮬레이터를 통하여 실행 트레이스들을 확인하고 그것으로부터 시스템이 요구 사항을 달성하고 있는지를 확인하였다.

분산 명세에서 가장 중요한 아이디어는 모듈간의 상호 작용이었으며 상호 작용은 다른 모듈의 행위를 블랙박스처럼 고려한다는 것이다. 개발한 시뮬레이터는 결과로 모델의 실행 트레이스들을 생성하는데 만약 요구 사항을 부합하지 않는 행위가 발생된다면 그 원인을 발견하는 데 중요한 정보를 제공한다.

하나의 명세를 각 모듈에 해당하는 명세로 분리하는 것은 각 모듈의 독립성을 보장할 수 있는데 특히 시스템의 요구 사항이 변경된 경우에 명세 수정 및 모델 생성의 비용이 줄어들므로 효율적이라고 할 수 있겠다. 또한 이 점은 전통적인 개발 방법론에서 유지 보수성을 향상 시킬 수 있는 접근법이다.

본 연구에서 제안하는 방법 및 도구는 궁극적으로 다수의 모듈이 존재하고 복잡한 상호 작용을 가지고 있는 시스템의 품질을 확인할 때 비용 및 용이성 측면에서 도움이 될 수 있을 것이다.

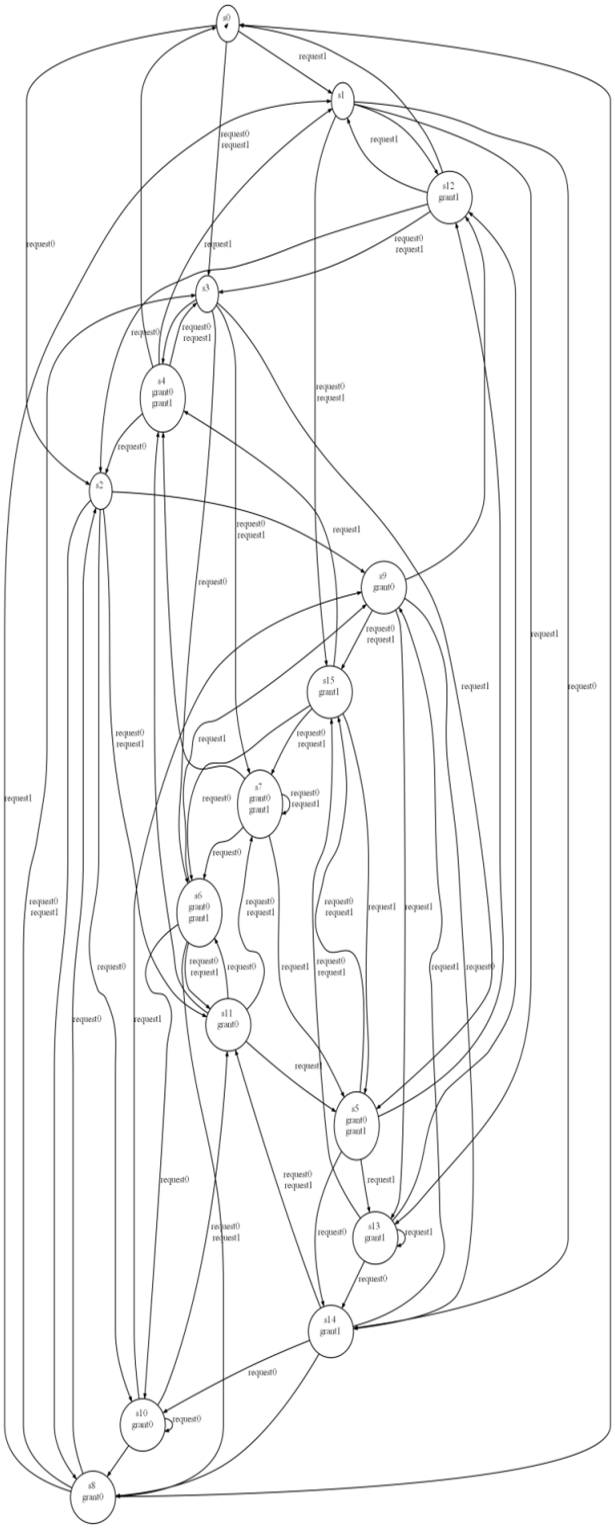

Fig. 6. The automaton of an arbiter which is generated from changed specification

Reference

[1] K. Ashton, “That Internet of Things’ Thing”, RFID Journal, 22 June, 2009.

[2] TTA 용어 사전, http://word.tta.or.kr/terms/terms.jsp

[3] Nick Wainwright, http://vimeo.com/31919813

[4] E. M. Clarke, O. Grumberg, D. A. Peled, “Model Checking”, MIT Press, 1999.

[5] A. Church, Logic, arithmetic and automata. In Proc. Int. Congr. Math., pp.23-25, 1962.

[6] J. R. Buchi and L. H. Landweber, Solving sequential conditions by finite-state strategies, Trans. Amer. Math. Soc., Vol.138, pp.295-311, 1969.

[7] M. O. Rabin, Automata on Infinite Objects and Church’s Problem, Vol.13 of Regional Conference Series in Mathematics. Amer. Math. Soc., 1972.

[8] Pnueli, A., Rosner, R., On the synthesis of a reactive module. In Proc. 16th ACM Symp. Princ. of Prog. Lang., pp.179-190, 1989.

[9] N. Piterman, A. Pnueli, and Y. Sa’ar, Synthesis of Reactive(1) Designs. In Pro-ceedings of the 7th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI’06), LNCS Vol.3855, pp.364-380, 2006.

[10] Pnueli. A, Sa’ar, Y, Lenore D. Zuck, “Jtlv: A Framework for Developing Verification Algorithms”, CAV, pp.171-174, 2010.

[11] JADE(Java Agent Development Framework), http://jade.tilab.com

권 령 구

e-mail : rkay8496@kyonggi.ac.kr

2011년 경기대학교 컴퓨터과학과(학사)

2013년 경기대학교 컴퓨터과학과(석사)

2013년~현 재 경기대학교 컴퓨터과학과 박사과정

관심분야 : Formal Verification, Software Engineering

권 기 현

e-mail : khkwon@kyonggi.ac.kr

1985년 경기대학교 전자계산학과(학사)

1987년 중앙대학교 전자계산학과(석사)

1991년 중앙대학교 전자계산학과(박사)

1991년～현 재 경기대학교 컴퓨터과학과 교수

1999년～2000년 미국 카네기멜론대학 전산학과 방문교수

2006년～2007년 미국 카네기멜론대학 전산학과 방문교수

2014년～현 재 한국정보과학회 소프트웨어공학 소사이어티 회장

관심분야 : Formal Verification, Software Engineering