시나리오 기반 명세 모델로부터 반응형 시스템 모델 개발☆

Developing a Reactive System Model from a Scenario-Based Specification Model

권 령 구* 권 기 현**
Ryoung-Kwo Kwon Gi-Hwon Kwon

요 약

다수의 오브젝트로 구성된 반응형 시스템을 모델링 하거나 디자인 하기 위해 외부의 입력 및 오브젝트들간의 상호작용을 분석하는 것은 중요하고 어려운 문제이다. 또한, 반응형 시스템이 요구 사항들을 모든 가능한 환경 하에서 올바르게 만족하는지를 확인하는 것은 많은 노력이 필요 하다. 본 논문에서는 요구 사항들을 기존에 다양한 분야에서 사용 되는 시나리오 명세 언어인 *MSC(Message Sequence Chart)*에 대해 구문 및 의미를 확장한 *LSC(Live Sequence Chart)*를 이용하여 반응형 시스템에 적합한 시나리오 기반 명세 모델을 만든다. 그리고 *LTL Synthesis*를 통해 각 오브젝트에 대하여 모든 요구 사항을 올바르게 만족하는 반응형 시스템 모델을 자동으로 생성한다. 마지막으로 생성된 반응형 시스템 모델로부터 의미적으로 동일한 코드로 변환하는 과정을 반복함으로써 전체 반응형 시스템을 개발하는 방법을 제안한다.

ABSTRACT

It is an important and a difficult task to analyze external inputs and interactions between objects for designing and modeling a reactive system consisting of multiple object. Also the reactive system is required huge efforts on confirm it can satisfy requirements under all possible circumstances. In this paper, we build from requirements to a scenario-based specification model using LSC(Live Sequence Chart) extending MSC(Message Sequence Chart) with richer syntax and semantic. Then the reactive system model satisfying all requirements for each object in this system can be automatically created through LTL Synthesis. Finally, we propose a method of reactive system development by iterative process transforming a reactive system model to codes.

☞ keyword : LTL Synthesis, Reactive System(반응형 시스템), Scenario Based Specification(시나리오 기반 명세), Live Sequence Chart

1. 서 론

다수의 오브젝트들로 구성된 복잡한 반응형 시스템을 모델링 하거나 디자인하기 위해 외부의 입력 및 오브젝트들 간의 상호 작용을 분석하는 것은 중요하고 어려운 문제이며 반응형 시스템이 요구 사항들을 모든 가능한 환경 하에서 올바르게 만족하는지를 확인하는 것은 많은 노력이 필요 하다. 때때로 반응형 시스템은 하드웨어에 임베디드된 형태일 수 있거나 실시간 또는 동시성에 큰 영향을 받을 있다. 또한 반응형 시스템을 구성하는 오브젝트의 수가 증가 할수록 반응형 시스템의 복잡도는 급증하게 된다. 특히 차량 및 항공 산업에 사용되는 반응형 시스템의 경우 인간의 생명 또는 안전과 상당한 관련이 있으므로 올바른 반응형 시스템을 개발하는 것은 중요하고 어려운 문제이다.

따라서, 반응형 시스템의 요구 사항들로부터 바로 그 시스템을 자동으로 생성해내는 것은 많은 연구자들에게 꿈 같은 일로 여겨 졌다.

[1]에서 *Harel*은 복잡한 반응형 시스템을 개발하기 위한 새로운 방법론을 제안 하였다. *Harel*은 반응형 시스템에 대한 요구 사항을 사용자 친화적 방법을 통해 시나리오들로 구체화하고 모델 검증 또는 *Synthesis*를 통해 시스템 모델을 진화 시켰다. 이 후 진화한 모델은 실행 가능한 코드로써 이해 될 수 있다. 또한 [2]에서는 *Harel*이 제안한 방법에 대하여 이론적 가능성을 연구 하였고, [3]에서는 하나 이상의 *Live Sequence Chart*의 명세들을 조합하여 *LTL Synthesis*를 수행하는 알고리즘을 제안 하였다. [4]에서 *LTL Synthesis*를 통해 인공 지능 로봇의 목적을

* 준 회 원 : 경기대학교 컴퓨터과학과 석사 과정(교신저자)
rkay8496@kyonggi.ac.kr
** 정 회 원 : 경기대학교 컴퓨터과학과 교수
khkwon@kyonggi.ac.kr
[2011/08/30 투고 - 2011/09/08 심사 - 2011/11/18 심사완료]
☆ 본 연구는 2011학년도 경기대학교 대학원 연구원장학생 장학금 지원에 의하여 수행되었음.
☆ 본 논문은 2011년도 한국인터넷정보학회 하계학술발표대회 우수논문의 확장버전임.

달성하게끔 하는 로봇 컨트롤러를 자동으로 생성하였다.

본 논문에서는 *Harel*이 제안한 방법에 기반하여 *LTL Synthesis*를 통한 반응형 시스템 모델을 생성하기 위해 다음과 같은 내용을 서술 한다.

- 반응형 시스템 개발 방법에 대한 소개
- 반응형 시스템에 적합한 시나리오 기반 명세 언어의 선택 및 시나리오 모델 작성
- 반응형 시스템 모델 생성을 위한 *LTL Synthesis* 이론과 모델 생성
- 생성된 반응형 시스템 모델로부터 실제 코드로 변환

또한, 논문을 걸쳐서 실제 반응형 시스템의 예를 소개하며 시나리오 기반 명세 모델로부터 *LTL Synthesis*를 통해 곧 바로 반응형 시스템 모델을 생성하고 그것을 의미적으로 동일한 코드로 작성하는 과정을 소개 한다.

본 논문 구성은 다음과 같다. 2장에서는 *Live Sequence Chart*와 *Synthesis*에 관하여 소개하고 3장에서는 제안하는 반응형 시스템 개발 방법에 대한 개요 및 관련 연구를 설명한다. 4장에서는 제안 방법을 통하여 반응형 시스템을 개발하는 예제를 보이고, 5장에서는 결론 및 향후 연구를 기술한다.

2. 배경 지식

Live Sequence Chart(*LSC*)는 *Message Sequence Chart* (*MSC*)을 확장한 것이다. *MSC*와 같이 *LSC*는 오브젝트들을 나타내는 생명선, 하나 또는 그 이상의 생명선들을 포함하는 이벤트를 포함하고 있다. *LSC*의 가장 기본적인 요소는 메시지다. 메시지는 두 개의 생명선 사이(또는 자기 자신의 생명선)의 화살표로 나타나며 이벤트를 보내는 오브젝트와 그 이벤트를 받는 오브젝트를 의미 한다. 일반적인 *LSC*는 *Prechart*(육각형 점선)와 *Mainchart*(사각형 박스)로 구성 된다. 가장 중요한 *LSC*의 의미들은 언제든지 반응형 시스템의 실행 동안에 *Prechart*가 만족 되면 *Mainchart* 또한 반드시 만족되어야 한다는 것이다.

*LSC*에 표현되는 오브젝트들은 시스템에 의해 제어되거나 환경에 의해 제어 될 수 있다. 어떠한 메시지가 시스템(환경)에 의해 제어되는 오브젝트로부터 보내졌다면 해당 메시지는 시스템(환경) 메시지라고 한다.

본 논문에서는 *Synthesis*를 시스템과 환경 사이의 *two-player* 게임으로써 고려 한다. 따라서 이 게임을 해결하기 위하여 *Game Structure*를 정의할 필요가 있다. 이것의 정의는 [5]에서 참고 하였다. *Game Structure*는 $G = < V, X, Y, \theta, \rho_s, \rho_e, \phi >$로 표현 된다. V는 상태 변수들의 집합일 때, X는 시스템에 의해 제어되는 변수들이고 Y는 환경에 의해 제어되는 변수들이다. θ는 초기 조건이다. ρ_s와 ρ_e는 시스템과 환경의 천이 관계를 각각 나타낸다. 마지막으로 ϕ는 시스템이 무한히 언젠가는 만족해야 하는 승리 조건을 의미 한다.

*Strategy*는 상태들의 연속을 가능한 시스템 액션들의 집합으로 매핑하는 부분 함수(*Partial Function*)이다. 시스템이 취한 각 스텝이 *Strategy*에 의해 허용되는 것이라면 어떠한 시스템의 실행은 *Strategy*를 충분히 따른다고 할 수 있다. 만약에 어떠한 시스템의 실행이 승리 조건을 만족하도록 *Strategy*를 따른다면 해당 *Strategy*는 시스템을 위해 승리하며, *Game Structure*의 초기 상태에서 그와 같은 *Strategy*가 존재 할 때 *Game Structure*는 실현 가능하다고 할 수 있다.

Linear Temporal Logic(*LTL*)은 명제 논리와 몇 개의 시제 연산자를 포함함으로써 명제들의 순서들 상에 가능한 행위 표현이 가능하게 하는 논리이다. 명제 집합 상에 *LTL* 논리식의 구문은 다음과 같이 구성 된다.

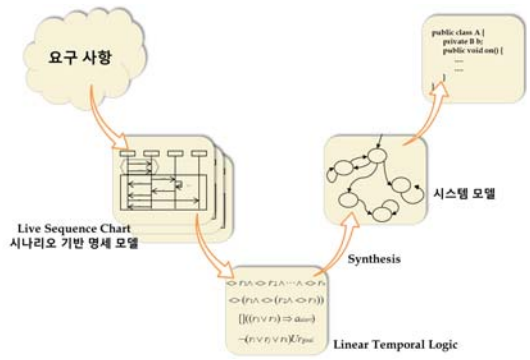
$$\varphi ::= true \mid false \mid p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \mathrm{O}\varphi \mid <> \varphi \mid [\,]\varphi \mid \varphi_1 U \varphi_2$$

일반적으로 이진 상수 *true*는 *true* = $\phi \vee \neg\phi$, *false*는 *false* = $\neg$*true*로 정의 된다. $\neg$와 $\vee$이 주어지면 $\wedge$, $\Rightarrow$, $\Leftrightarrow$를 정의할 수 있다. 시제 연산자 "*Eventually*" $\Diamond\phi$ = *true*Uϕ, "*Always*" $\Box\phi$ = $\neg\Diamond\neg\phi$로 유도할 수 있다.

LTL 논리식 ϕ의 모델 σ는 명제들의 무한한 시퀀스이다. 즉, ϕ에 나타나는 명제 집합을 P'이라면, $P' \subseteq P$인 모든 집합 P에 대하여 $(2^P)^\omega$는 모델이다. 이 때 $\sigma(i)$는 무한한 시퀀스의 위치 i에서의 명제들의 집합을 나타낸다. *LTL* 논리식 ϕ가 모델 σ의 어느 시점에서 만족되는지의 여부와 세부적인 의미 해석은 [6]을 참고한다.

3. 반응형 시스템 개발 방법

이 연구에서는 다수의 오브젝트로 구성된 반응형 시스템을 대상으로 한다. 반응형 시스템은 환경과 끊임없는 관계를 유지하며 상호작용 하는 것이라고 정의 한다. 반응형 시스템은 환경의 입력에 대하여 자신의 상태를 변경시키고 어떠한 메시지 혹은 이벤트를 생성한 후 환경의 다음 입력에 반응하기 위하여 준비한다. 환경의 입력은 사용자의 시스템에 대한 조작 및 시스템이 관찰한



(그림 1) 반응형 시스템 개발 방법

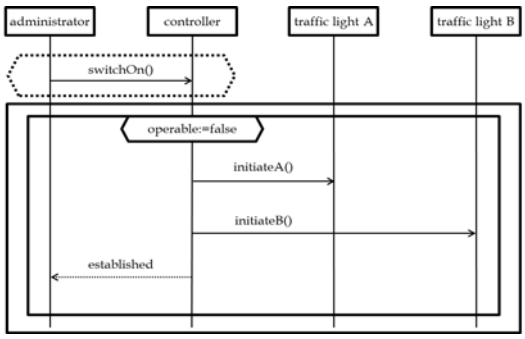
조건의 변화를 외부 환경의 입력이라고 할 수 있으며, 또한 특정 오브젝트가 다른 오브젝트로부터 받은 메시지 혹은 이벤트를 특정 오브젝트에 대한 외부 환경의 입력이라고 할 수 있다. 궁극적으로 반응형 시스템은 외부 환경의 입력이 발생하였을 때 다수의 오브젝트들간의 상호작용을 통해 이벤트의 시퀀스를 그 응답으로써 생성하게 되고, 그 이벤트의 시퀀스는 요구 사항을 완전히 실현할 수 있어야 한다.

반응형 시스템을 개발한다는 것은 다음 질문과 같다. "구현물이 요구 사항을 충분히 실현하는가?". 이 질문의 근본적인 원인은 개발 단계에서 요구 사항과 실제 구현물 사이의 차이가 크기 때문이다. 따라서 요구 사항과 실제 구현물 사이에 요구 사항을 포함하는 시나리오 기반 명세 모델과 시스템의 구조와 행위를 표현하는 시스템 모델을 배치함으로써 이 차이를 좁힐 수 있다. 본 논문에서 제안하는 반응형 시스템 개발의 전체는 (그림 1)과 같다.

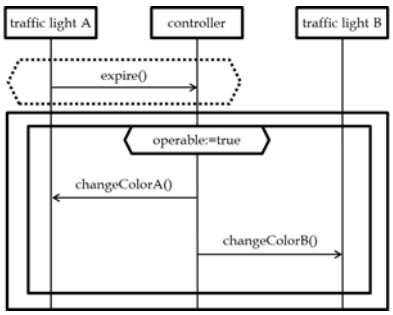
이러한 과정을 실현하기 위하여 두 가지 필수적인 조건이 필요 하다. 하나는 표현력이 풍부하면서 직관적인 명세 언어를 선택하는 것이며 이 명세 언어로부터 반응형 시스템 모델을 생성하기 위한 효율적인 *Synthesis* 알고리즘의 개발이 필요 하다. 그러나 *Synthesis*를 적용하려 할 때 알고리즘 자체의 복잡성과 개발 방법론적인 측면에서 바라볼 필요가 있다. 일반적으로 요구 사항들은 다수의 인원 혹은 팀에 의해서 작성되어지고 새로운 요구 사항들이 식별되어 요구 사항이 수정 됨에 따라 그것들은 진화 한다. 따라서, 요구 사항들이 수정 될 때마다 완전한 전체 시스템을 *Synthesis* 알고리즘을 통해 생성하지 않고 요구 사항의 부분들을 생성하고 그것들을 조합함으로써 *Synthesis* 알고리즘의 실행 시간을 뚜렷하게 줄일 수 있다. 또한 이전에 생성했던, 존재하는 반응형 시스템 모델들을 재 사용함으로써 생산성 향상에 큰 도움을 줄 수 있다.

반응형 시스템을 개발하고자 할 때 모든 요구 사항을 포함하는 시나리오 기반 명세 모델과 그것으로부터 시스템의 구조와 행위를 표현하는 시스템 모델은 필수적이다. 시나리오 기반 명세 모델은 요구 사항에 가까운 형태라고 할 수 있으며, 표현력이 풍부하면서 직관적인 명세 언어를 선택하는 것은 중요 하다. 요구 사항이 시스템에 대하여 기대하는 행위나 제약조건이라면 시나리오들의 집합은 모든 요구 사항을 포함하고 있다고 고려할 수 있다. 시나리오 기반 명세 모델을 생성하기 위해 *MSC*를 보편적으로 널리 쓰이지만 본 논문에서는 *LSC*를 이용했다. *LSC*는 시스템의 입력에 해당하는 이벤트와 입력에 대한 응답인 이벤트들의 시퀀스를 구문 및 의미적으로 명확히 구분할 수 있는 능력을 가지고 있는 반면, *MSC*는 *LSC*와 달리 그러한 구분이 명확하지 않고 단지 암묵적으로 이해되어야 한다. 따라서, 다수의 모듈로 구성된 반응형 시스템의 시나리오를 *LSC*를 이용해 명세 하는 것과 이 후에 설명할 *Synthesis*를 적용함에 있어서 적합한 선택이라고 할 수 있다.

반응형 시스템 모델은 구현물과 가까운 형태이며 시스템의 구조와 행위를 포함하고 있다. 일반적으로 시스템 모델을 요구 사항에 대하여 완전히 만족하는지를 확인할 목적으로 다양한 방법을 사용한다. 대표적으로 테스팅이 있는데 이 방법은 시스템의 무한히 많은 실행들을 모두 수행할 수 없으므로 확인되지 않은 실행을 통해 요구 사항을 위반할 수 있는 문제가 있다. 그리고 모델 검증은 검사할 속성에 대하여 상태 공간을 탐색하고 속성을 만족하는 실행이 존재하는지를 확인한다. 하지만 모델 검증은 속성을 만족하는 하나의 실행을 발견할 때까지만 탐색을 수행하므로 미 탐색된 상태 공간에 의해 요구 사항이 위반될 수 있는 실행이 존재할 수 있다[7]. 따라서, 상태 공간의 완전한 탐색을 행하기 위해서는 *Synthesis*를 고려할 수 있다. *Synthesis*는 시스템과 환경 사이의 *two-player* 게임으로 고려한다[6]. 시스템은 모든 요구 사항들을 만족함으로써 게임을 승리할 수 있고 환경은 게임을 요구 사항을 실현하지 못하게 하는 상태로 이끌어나감으로써 시스템이 승리하지 못하게 방해한다. *Synthesis*를 수행하기 위해서는 요구 사항을 포함하는 *LTL*(*Linear Temporal Logic*) 논리식을 작성해야 한다. *Synthesis*는 작성된 *LTL* 논리식에 대해 반응형 시스템의 실현 가능성을 결정 한다. 즉, 반응형 시스템이 모든 입력 이벤트와 모든 이벤트 시퀀스상에서 요구 사항을 위



(그림 2) LSC01 컨트롤러를 활성화 하는 시나리오



(그림 3) LSC02 신호등의 색깔을 바꾸는 시나리오

반하지 않는지를 판단하고, 만약 그럴 수 있다면 천이 시스템을 추출 가능 하다. 추출된 천이 시스템은 *Winning Strategy*를 내포 하고 있는데, 이것은 모든 환경의 입력에 대한 시스템의 올바른 응답이라고 정의 된다. 천이 시스템은 *Safety Property*를 위반하는 천이들을 제거하고 *Liveness Property*를 만족하기 위한 조건을 추가한 것으로써 반응형 시스템은 추출된 천이 시스템의 천이들을 따름으로써 요구 사항을 완전히 실현 가능하다는 것이 보장된다.

[7]에서는 시나리오 기반 명세 모델로부터 *Synthesis*를 통해 반응형 시스템을 개발하는 과정을 보여주고 있다. 선행 연구는 시나리오 기반 명세 모델에 등장하는 전체 오브젝트들의 행위를 표현하는 *LTL* 논리식을 완전히 작성하고 *Synthesis*를 통해 천이 시스템을 생성한다. 따라서, 반응형 시스템의 오브젝트의 수가 증가함에 따라 한 번에 작성해야 하는 *LTL* 논리식의 크기도 증가하므로 이 과정의 복잡성은 크게 증가하며 *Synthesis*를 통해 생성된 천이 시스템을 오브젝트 단위의 코드로 변환하기 위한 노력이 많이 요구 된다. 본 논문에서는 시나리오 기반 명세 모델로부터 오브젝트 단위의 *LTL* 논리식을 작성하고 *Synthesis*를 통해 천이 시스템을 생성하게 되며, 이 천이 시스템으로부터 오브젝트 단위의 코드를 변환 하게 된다. 결국 오브젝트 단위의 개발을 반복함으로써 전체 반응형 시스템을 개발하게 되는데 이 개발 방법은 선행 연구에 비하여 전체 개발 과정의 용이성을 향상시키고 복잡성을 줄일 수 있다.

4. 신호등 시스템 컨트롤러

4.1 시나리오 기반 명세 모델

요구 사항은 개발 대상의 반응형 시스템에 대하여 기대하는 행위 및 목적, 그리고 그것들을 달성하기 위한 제약 조건들을 내포하고 있다. 요구 사항은 자연어로 작성되는 것이 보편적이다. 시나리오 기반 명세 모델을 만들기 전에 요구 사항으로부터 대상 시스템을 구성하는 오브젝트들을 식별해야 한다. 그리고 시스템의 각 모듈들이 기능들을 수행하기 위하여 어떠한 이벤트들의 시퀀스를 갖는지를 시나리오로써 구체화 한다.

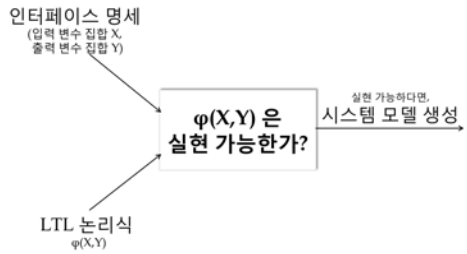
신호등 시스템 컨트롤러에 대한 요구 사항은 다음과 같다: 1) 신호 시스템 컨트롤러는 보행자 신호등과 차량 신호등을 관장한다. 2) 컨트롤러 관리자는 컨트롤러의 스위치를 *On*으로 하면 각 신호등과 연결이 활성화 되고 컨트롤러가 운용 가능한 상태가 된다. 3) 운용 가능한 상태에서 컨트롤러는 신호 표시 시간이 만료 되었다는 이벤트를 받으면 다른 색의 신호를 표시 하라는 이벤트를 보낸다.

요구 사항을 통해 식별한 오브젝트들은 다음과 같다: 1) 컨트롤러. 2) 보행자 신호등(신호등 *A*). 3) 차량 신호등(신호등 *B*).

이제 기능별 시나리오를 *LSC*를 통해 구체화 시킨다. *LSC*에 대한 구문 및 의미는 [8]을 참고한다.

4.2 반응형 시스템 모델

시나리오 기반 명세 모델로부터 시스템 모델을 생성하는 것을 *Synthesis*로 고려할 수 있다. *Synthesis*는 시스템의 인터페이스 명세 - 입력 변수 *X* 집합, 출력 변수 *Y* 집합 - 와 인터페이스 상의 시나리오 기반 명세 모델을 표현하는 시제 논리식을 입력으로 받아 들인다. *Synthesis* 루틴은 두 가지의 입력을 기반으로 하여 실현 가능한 시스템 모델이 존재 하는지를 결정 한다. 만약 실현 가능하다면 모든 요구 사항을 만족하는 시스템 모델을 생성 한다. 본 논문에서는 시나리오 기반 명세 모델을 시제 논리의 한 종류인 *LTL*을 이용하여 작성 한다. (그림 4)는 *LTL*



(그림 4) *LTL Synthesis*

*Synthesis*의 개요를 보여준다.

controller 모듈의 인터페이스는 다음과 같이 정의 된다.인터페이스는 *controller* 모듈이 받거나 보내는 이벤트들로 구성이 되어 있는데, 출력 변수 집합 *Y*에는 입력 변수 집합 *X*에 없는 *operable*과 *inputable* 변수가 추가 된다. *operable* 변수는 *controller*가 현재 운용 가능한 상태인지를 나타낸다. *inputable* 변수는 연속된 출력 이벤트들의 시퀀스를 보장하기 위해 사용된다. 설명하자면, *Synthesis*는 *two-player* 게임이면서 환경과 시스템이 번갈아 가면서 자신의 상태를 변화 시킨다. 따라서 이 변수가 참인 경우에만 입력을 허용하지만 거짓일 경우에는 시스템의 출력 이벤트들만을 허용할 수 있다.

*k*는 인터페이스에 정의된 변수의 수이며, 각 변수에 대하여 상태 변화를 *LTL* 논리식으로 작성 해야 한다. 이 때 *c*는 변수의 상태가 변하는 조건식이다.

$$M = \text{a set of modules}$$

$$X = \bigcup_{j=0}^{M} event^{r}_{m_j \to controller}$$

$$Y = \bigcup_{k=0}^{M} event^{s}_{controller \to m_k}$$

마지막으로 *controller*의 초기 조건과 무한히 언젠가는 만족해야 할 목표를 *LTL* 논리식으로 작성해야 한다. 초기 조건은 모든 변수들이 거짓인 상태이다. 즉, *controller*가 운용 가능한 상태가 아니라는 것이다.

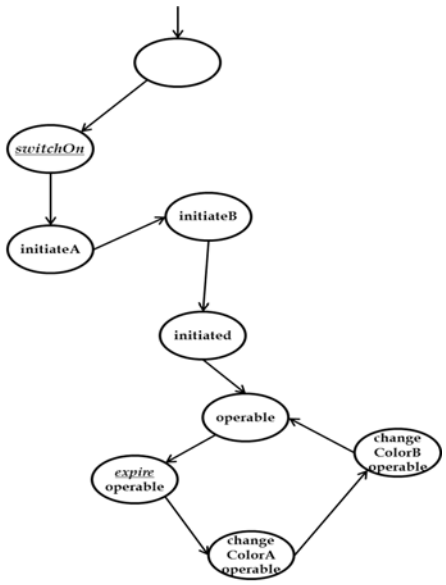
$$\varphi_t = \left\{ \begin{array}{c} \Box(c_1 \Rightarrow \bigcirc v_1) \wedge \\ \vdots \quad \vdots \\ \Box(c_k \Rightarrow \bigcirc v_k) \end{array} \right\} \qquad \varphi_i = \left\{ \begin{array}{c} \neg v_1 \wedge \\ \vdots \\ \neg v_k \end{array} \right\}$$

여기서 식별 가능한 목표, 즉 승리 조건은 두 가지이다: 1) 스위치가 *On*이 되면 모든 신호등이 활성화 되어야 한다. 2) 신호 표시 시간이 만료 되었다는 이벤트를 받으면 모든 신호등에게 다른 색의 신호를 표시하라는 이벤트를 보낸다. 목표들은 다음의 *LTL* 논리식으로 작성 된다.

$$\varphi_g = \left\{ \begin{array}{l} \Box\Diamond(switchOn \Rightarrow initiated) \wedge \\ \Box\Diamond(\exp ire \Rightarrow changeColorB) \end{array} \right\}$$

$$\varphi_{sys} = \left\{ \begin{array}{l} \neg operable \wedge \neg inputable \wedge \neg initiatedA \wedge \neg initiatedB \wedge \neg initiated \wedge \neg changeColorA \wedge \neg changeColorB \wedge \\ \\ \Box(initiated \Rightarrow \bigcirc(operable \wedge inputable)) \wedge \\ \Box(operable \Rightarrow \bigcirc operable) \wedge \\ \Box(operable \wedge changeColorB \Rightarrow \bigcirc inputable) \wedge \\ \Box(inputable \wedge \bigcirc \exp ire \Rightarrow \neg \bigcirc inputable) \wedge \\ \Box(switchOn \Rightarrow \bigcirc initiateA) \wedge \\ \Box(initiateA \Rightarrow \neg \bigcirc initiateA \wedge \bigcirc initiateB) \wedge \\ \Box(initiateB \Rightarrow \neg \bigcirc initiateB \wedge \bigcirc initiated) \wedge \\ \Box(initiated \Rightarrow \neg \bigcirc initiated) \wedge \\ \Box(\exp ire \wedge operable \Rightarrow \bigcirc changeColorA) \wedge \\ \Box(changeColorA \Rightarrow \neg \bigcirc changeColorA) \wedge \\ \Box(operable \wedge changeColorA \Rightarrow \bigcirc changeColorB) \wedge \\ \Box(changeColorB \Rightarrow \neg \bigcirc changeColorB) \wedge \\ \\ \Box\Diamond(switchOn \Rightarrow initiated) \wedge \\ \Box\Diamond(switchOn \Rightarrow initiated) \end{array} \right\}$$

(그림 5) 작성한 LTL 논리식



(그림 6) 생성된 시스템 모델

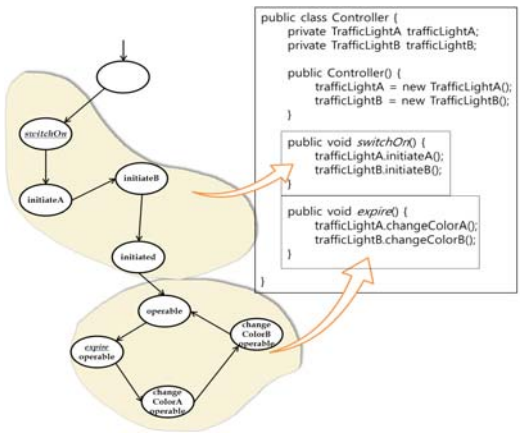
(표 1) 기능별 요구 사항의 유한 시퀀스

요구 사항 번호	유한 시퀀스
2	*switchOn*-initiateA-initiateB-initiated-operable
3	operable-*expire*,operable-changeColorA,operable-changeColor,operable

앞서 설명한 내용으로 ***LTL*** 논리식을 작성하였고 (그림 5)에 반응형 시스템에 대한 ***LTL*** 논리식을 보여주고 있다. *Synthesis* 루틴을 수행하여 실현 가능한지를 검사하고 가능하다면 다음과 같은 천이 시스템, 즉 시스템 모델이 생성 된다.

(그림 6)은 결과로 얻은 시스템 모델이다. 시스템 모델은 *Winning Strategy*를 내포 하고 있는데, 이것은 모든 환경의 입력에 대한 시스템의 올바른 응답으로써 반응형 시스템이 이 모델의 천이에 근거하여 가능한 입력들에 대해 언제나 올바른 응답을 선택할 수 있다. 이 때 각 상태 안에 이탤릭 체로 작성된 것은 환경으로부터의 입력들이고 나머지는 시스템의 반응 시퀀스라고 해석할 수 있다.

생성된 반응형 시스템 모델에서 기능별 요구 사항을 만족할 수 있는 유한 시퀀스의 예를 (표 1)에서 보여 주고 있다. 4.1절에 소개된 각 요구 사항의 번호를 나타내었으며, 1)번 요구 사항은 기능에 관련된 것이 아니므로



(그림 7) 시스템 모델에서 코드 변환

생략 하였다. 아래의 표에서 확인할 수 있듯이 반응형 시스템의 무한한 시퀀스 하에서 환경에 대하여 어떠한 입력이 발생되더라도 모든 요구 사항을 달성할 수 있는 반응형 시스템 모델을 생성할 수 있었다.

4.3 반응형 시스템 모델에서 코드 변환

*Synthesis*를 통해 생성된 시스템 모델을 기반으로 하여 의미적으로 동일한 코드로 변환 될 수 있다. (그림 7)은 코드로 변환하는 모습을 묘사하고 있는데 천이 시스템에서 환경에서의 입력은 곧 *Controller* 클래스가 다른 클래스에서 제공해야 할 인터페이스로 고려할 수 있다. 그리고 입력에 이은 시스템 응답들의 시퀀스는 해당 인터페이스를 구현할 로직으로써 변환 될 수 있다.

5. 결론 및 향후 연구

본 논문에서는 시나리오 기반 명세 모델로부터 반응형 시스템을 개발 하는 방법을 제안하였다. 시스템에 대한 요구 사항을 ***MSC***를 확장한 ***LSC***를 이용하여 시나리오 기반 명세 모델을 생성하였으며 *Synthesis*를 통해 모든 요구 사항을 만족하는 시스템 모델을 자동으로 생성 하였다. 생성된 하나의 시스템 모델은 하나의 오브젝트이며 이것에 기반하여 의미적으로 동일한 코드를 변환하였다. 제안한 반응형 시스템 개발 방법은 ***LSC***와 *Synthesis*를 통해 수학적으로 정밀하고 엄격한 모델을 생성할 수 있었다. 다수의 오브젝트로 구성된 복잡한 반응형 시스템에 대하여 모든 요구 사항을 완전히 올바르게 만족할 수 있

음을 보장 함으로써, 그러한 시스템을 올바르게 모델링 하거나 디자인하기 위한 수고를 덜어주고 올바름과 안전성을 보장해 준다. 또한 제안한 반응형 시스템 개발 방법은 요구 사항에서부터 실제 구현물 개발까지 각 단계의 이동이 자연스럽고 엔지니어 친화적이므로 생산성을 향상시키는데 도움을 줄 수 있다.

본 논문에서 제안한 방법의 미래를 밝게 하기 위해서는 중요한 연구가 남아있다. 그것은 좀 더 사용자 친화적이고 상위 수준의 자동화 도구의 지원일 것이다. *LSC*를 작성하기 위해 사용자가 *LSC*에 대한 구문 및 의미에 대한 지식을 일정 수준 이상 필요하다. 만약에 개발될 반응형 시스템 인터페이스의 *mock-up* 형태를 지원하여 사용자는 그것을 단지 조작하여 *LSC*를 작성할 수 있다면 시간 및 비용의 소모가 줄 것이다. 또한 *Synthesis*를 위한 *LTL*의 작성 및 코드의 변환 과정을 자동화 하기 위한 연구들 또한 중요 하다. 궁극적으로 이러한 연구들이 결실이 맺어져 실제 개발 방법에 유용하게 적용 된다면 개발 생산성에 큰 향상을 이룰 것이라고 확신한다.

참 고 문 헌

[1] D. Harel, “From Play In Scenarios to Code: An Achievable Dream”, Computer, vol. 34, no. 1 pp. 56-60

[2] Y. Bontemps, P. Schobbens, and P. Y. Schobbens, “From Live Sequence Charts to State Machines and Back: A Guided Tour”, IEEE Trans. Software Eng., 31(12):999-1014, 2005.

[3] H. Kugler, and H. Segall, “Compositional Synthesis of Reactive Systems from Live Sequence Chart Specifications”, In: Proc. 15th Intl. Conference on Tolls and Algorithms for the Construction and Analysis of Systems, LNCS. Springer, Heidelbers, 2009.

[4] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, “Temporal Logic-based Reactive Mission and Motion Planning”, IEEE Transactions on Robotics, vol. 25, No. 6, pp. 1370-1381, 2009.

[5] A. Pnueli. “Extracting Controllers for Timed Automata. Technical report, NYU, 2005.

[6] A. Pnueli, and R. Rosner, “On the Synthesis of a Reactive Module”, In Proceedings of the 16th ACM Symposium Principles of Programming Language, 1989.

[7] H. Kugler, C. Plock, and A. Pnueli, “Controller Synthesis from LSC Requirements”, in Proc. 12th International Conference on Fundamental Approaches to Software Engineering (FASE 2009), 2009, pp. 79-93.

[8] D. Harel, and R. Marelly, “Come, Let’s Play: Scenario-Based Programming Using LSC’s and the Play-Engine”, Secaucus, NJ, USA:Springer-Verlag New York, Inc., 2003.

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

◐ 저 자 소 개 ◑

권 령 구 (Ryoung-Kwo Kwon)

2011년 경기대학교 컴퓨터과학과 학사
2011년~현재 경기대학교 컴퓨터과학과 석사 과정
관심분야: 소프트웨어 공학, 소프트웨어 자동 생성, 정형 검증, 소프트웨어 테스팅
E-mail : rkay8496@kyonggi.ac.kr

권 기 현 (Gi-Hwon Kwon)

1985년 경기대학교 컴퓨터과학과 졸업(이학사)
1987년 중앙대학교 전자계산학과 졸업(이학석사)
1991년 중앙대학교 전자계산학과 졸업(공학박사)
1999년 미국 카네기멜론대학 전산학과 방문 교수
2006년 미국 카네기멜론대학 전산학과 방문 교수
1991년~현재 경기대학교 컴퓨터과학과 교수
관심분야: 정형 기법, 정형 검증, 소프트웨어 테스팅, 소프트웨어 모델링, 소프트웨어 자동 생성
E-mail : khkwon@kyonggi.ac.kr