게임 소프트웨어를 위한 정형기법의 적용성 분석

Applicability Analysis of Formal Methods for Game Software

손한성
(주)에네시스

Han-Seong Son(hsson@enesys.co.kr)

요약

게임 소프트웨어 개발에는 기획과 프로그래밍 및 그래픽의 조화가 필요하다. 이 중 기획과 프로그래밍 사이에는 소프트웨어 분석 및 설계가 필수적 가교 역할을 해야 한다. 본 논문에서는 이러한 게임 소프트웨어 분석 및 설계를 위해 일반 소프트웨어 공학에서 많이 적용되고 있는 정형기법을 적용하는 것에 대한 가능성에 대하여 고찰한다. 정형기법은 일반적으로 게임 소프트웨어 분야보다는 고신뢰도를 요구하는 소프트웨어 분야에 적용하여 많은 성공을 거두어왔다. 최근 게임 소프트웨어 분야에서도 게임 기획과 게임 프로그래밍 사이에서 효율적인 의사 소통을 통하여 게임 소프트웨어 개발의 경제성과 품질을 향상시켜야 할 필요성이 대두되고 있다. 이러한 배경으로, 본 연구에서는 이러한 정형기법을 게임 소프트웨어의 분석 및 설계에 적용하기 위한 제안으로서 정형기법의 대표적인 기법들을 대상으로 적용성을 분석한다.

■ 중심어 : | 정형기법 | 게임 소프트웨어 분석 및 설계 | 적용성 |

Abstract

The game software development involves planning, programming and graphics. Between planning and programming, software analysis and design is essential and plays the role of a bridge. This article analyzes, for game software, the applicability of formal methods, which are widely used in general software engineering fields. Since the effective communication between game planners and game developers is crucial, appropriate application of formal methods give us a lot of benefits in view of development cost and software quality.

■ keyword : | Formal Methods | Game Software Analysis and Design | Applicability |

I. 서 론

게임 소프트웨어는 일반 소프트웨어와 마찬가지로 특정한 생명주기(life cycle)를 따라 개발된다. 이 가운데 가장 중요한 공정은 요구사항 분석으로서 게임 소프트웨어의 경우, 게임 기획에서 생산된 여러 산출물들을 분석하여 소프트웨어 개발자를 위한 요구사항 명세서로 표현하는 공정으로 정의할 수 있다. 이 때 특별히 중요한 것은 소프트웨어 요구사항에 사용되는 표현 형식이며, 표현 형식에는 명세에 대한 이해능력이 요구된다. 이는 소프트웨어 분석 및 설계가 게임 기획과 게임 프로그래밍의 가교 역할로서 게임 기획자와 게임 프로그래머는 바로 이 소프트웨어 요구사항 명세서를 근간으로 의사소통을 하기 때문이다.

접수번호 : #071029-008
접수일자 : 2007년 10월 29일
심사완료일 : 2007년 11월 14일
교신저자 : 손한성, e-mail : hsson@enesys.co.kr

정형기법은 요구명세 및 설계명세를 모두 수학적인 문자나 기호로 명세하여 설계 내에 모호함을 없애고 수학적인 증명방법으로 설계된 시스템의 다양한 특성 및 안정성과 신뢰성 등의 완전성을 검증하고 있다. 이러한 기법은 일반적으로 게임 소프트웨어 분야보다는 고신뢰도를 요구하는 소프트웨어 분야에 적용하여 많은 성공을 거두어왔다[1]. 최근 게임 소프트웨어 분야에서도 게임 기획과 게임 프로그래밍 사이에서 효율적인 의사소통을 통하여 게임 소프트웨어 개발의 경제성과 품질을 향상시켜야 할 필요성이 대두되고 있다. 이러한 배경으로, 본 연구에서는 이러한 정형기법을 게임 소프트웨어의 분석 및 설계에 적용하기 위한 제언으로서 정형기법의 대표적인 기법들을 대상으로 적용성을 분석한다.

II. 정형기법

정형기법이란 소프트웨어 공학의 많은 기법의 일종으로 오류가 없는 시스템을 설계하여 시스템의 신뢰성을 높이려는데 목적이 있다[2]. 정형기법에서는 시스템의 구성 요소를 수학적 객체 모듈로 취급하며, 이러한 객체의 성질과 동작을 묘사하고 예측하기 위해 수학적 모델을 제공한다. 이러한 수학적 기호를 사용함으로 시스템의 명세를 작성하는데 있어서 자연어가 일으킬 수 있는 애매모호함이나 불확실성을 최소한으로 줄일 수 있고, 설계된 사용자의 요구조건과 동일한지 수학적인 성질을 이용하여 증명할 수 있다. 정형기법은 크게 정형명세(Formal Specification)와 정형검증(Formal Verification)으로 나눌 수 있는데 정형명세는 시스템이 달성해야 하는 요구 사항과 그러한 요구 사항을 만족시키는 설계를 모사하는데 그 목적이 있고, 정형 검증은 그 설계가 요구사항을 만족하는지를 검사하는 일련의 과정을 의미한다.

정형기법에서 정형명세 언어는 다음과 같은 특징을 지니고 있다:

첫째, 수학적 기호나 문자 혹은 수학적으로 잘 정의된(Well-defined) 도식적 언어를 사용한다. 따라서 설계 내에 애매모호함을 줄일 수 있으며, 시스템을 유일한 Interpretation으로 변환할 수 있다.

둘째, 시스템의 객체를 수학적 객체로 다루기 때문에 시스템의 완전성을 증명하기 위해 수학적 증명방법을 이용할 수 있다. 즉 시스템에 요구되는 요구 사항을 모델링 되거나 설계된 시스템이 만족하는지를 수학적 증명방법으로 증명할 수 있다.

셋째, 이러한 증명방법으로 완전성이 증명된 설계를 통해 시스템의 완전성을 도모할 수 있다. 즉 시스템의 Safety, Liveness, Deadlock 과 같은 특성들을 사람이 개입하지 않고 자동으로 검증할 수 있다.

정형기법에는 프로세스 대수(Process Algebra)기반의 명세 및 검증기법, 상태 기반의 모델체킹(Model Checking)의 명세 및 검증기법, 그리고 수학적 논리를 기반으로 한 명세 및 검증 기법인 정리증명(Theorem Proving) 기법이 있다.

1. 정형 명세

정형 명세는 시스템이 달성해야 할 요구사항과 그러한 요구사항을 구현할 수 있는 설계를 묘사하는데 그 목적이 있다. 정형명세는 요구명세(Requirement Specification)과 설계명세(Design Specification)로 나눌 수 있다. 요구명세는 시스템이 무엇을 만족해야 하는가를 즉 사용자가 시스템에 무엇을 요구하는가 하는 것을 정의해 놓은 명세이다. 반면 설계 명세는 시스템이 어떻게 이루어져 있는가를 나타낸다. 즉 설계명세는 시스템의 행태를 구현할 목적으로 작성되며 시스템의 다양한 측면의 세부적인 사항을 포함한다. 그리고 설계명세가 요구명세를 비교하므로 설계하고자 하는 시스템이 정확한지를 검증할 수 있다.

정형명세 언어로서는 크게 State-diagram과 같은 도식적인 명세 언어와 논리식이나 프로세스 대수와 같은 수학적 기호와 문자를 사용하는 언어가 사용된다. 앞에서 언급한 것처럼 정형명세는 요구명세와 설계명세로 나뉘기 때문에 정형기법에 따라 각각의 요구명세 언어와 설계명세 언어를 가질 수 있다.

요구명세와 설계명세 언어와 다를 수도 같을 수도 있다. 즉 논리식에 근거한 정형기법인 Theorem proving 이나 프로세스 대수에 기반한 ACSR(Algebra Communicating with Sharing Resources)와 같은 명세 언어는 요구명세 언어와 설계 명세를 같은 언어로 구현

한다. 하지만 모델체킹은 요구명세와 설계명세 언어는 다르다. 즉 요구명세 언어는 시제논리(Temporal logic)을 이용한 명세를 하여, 시스템 가질 수 있는 모든 상태를 만들어서 현재부터 미래, 혹은 과거부터 현재까지의 시스템의 행위를 분석함으로 검증을 시행한다. 모델체킹에서 설계명세 언어는 명세 및 검증도구에 따라 다를 수 있지만 대부분 내부적으로 유한 상태 기계로 구현하여 오토마타 기반의 증명방법을 사용하여 검증하게 된다.

2. 정형검증

정형검증은 설계명세가 요구명세를 만족시키는지를 검사하는 것을 말한다. 이는 수학적 증명방법으로 설계명세가 요구명세를 만족시키는 지를 검사하는 것을 말하는데 이때, 설계명세가 요구명세를 만족하지 않을 경우, 검증도구는 반례(Counter Example)를 만들어줌으로 어떤 경우 만족하지 않는지를 추적할 수 있게 한다.

III. 정형기법의 적용성 분석

게임 소프트웨어는 그 특성상 사용자와의 상호작용이 중요한 인터랙티브 소프트웨어로 분류할 수 있다. 이러한 게임 소프트웨어의 특성을 고려할 때 적용 가능한 정형기법은 Statechart, Petri-net, SyncCharts, Message Sequence Chart, SMV, VIS 등 매우 다양하다. 본 연구에서는 위와 같은 인터랙티브 소프트웨어 시스템에 적용되어 온 정형 기법 중 대표적인 것을 선별하여 게임 소프트웨어에의 적용성을 분석하였다.

1. Statechart

STATECHART[3]는 상태기반의 시각적 명세언어로 도식적으로 표현되기 때문에 이해하기가 편하고, 인터랙티브 시스템의 행위적인 면을 표현하는데 장점을 가진다. Statechart는 다음과 같이 간략하게 정의할 수 있다.

STATECHART = mealy machine + depth(layer) + orthogonality(concurrency) + broadcast-communication

Statechart는 1986년 전부터 개발되어 인터랙티브 시스템의 설계 및 시뮬레이션에 응용되었으며 현재에는 내장형 시스템의 설계 및 구현에 널리 이용되고 있다. 시스템의 입출력은 시그널과 데이터의 입출력으로 한다. 그리고 시스템의 행위는 상태도의 전이로 표현하게 된다.

1.1 Statechart의 적용성 분석

Statechart는 전체적으로 도식적인 언어를 사용하기 때문에 소프트웨어 설계 및 설계의 이해에 있어서는 상당히 장점을 갖는다. [그림 1]은 자동차 경주 게임을 하는 게이머가 GUI에 나타나는 신호등을 임의로 켜거나 끄고자 할 때를 표시하는 Statechart이다. [그림 1]을 통해 Statechart는 매우 도식적이고 직관적임을 알 수 있다. 또한 Statechart 역시 정형적으로 Semantics가 정의되어 있음으로 Semantics를 정확히 이해하고 있다면 애매모호함 없이 설계하고 이해할 수 있다고 사료된다.

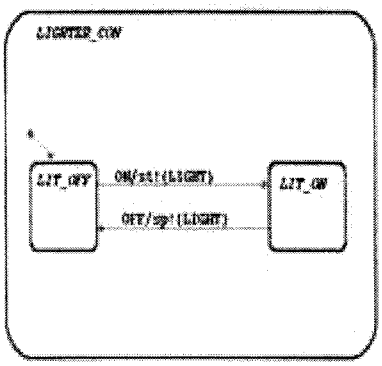


그림 1. 신호등 켜기/끄기 Statechart

특히, Statechart는 수년간의 적용 이력이 증명하듯이 게임 소프트웨어와 특성이 같은 인터랙티브 시스템의 설계 및 시뮬레이션에 적용이 가능하며, 게임 소프트웨어 또한 내장형 시스템이 많이 있음을 감안할 때 Statechart의 게임 소프트웨어 분석 및 설계에 대한 적용성은 충분하다고 판단된다. 하지만 모든 객체를 도식적인 하나의 개체로 표현해야 하기 때문에 복잡한 게임 소프트웨어를 설계한다면 설계도 자체의 복잡도가 증가할 수밖에 없다. 따라서 모든 시스템의 움직임 등을 분석하기 위해서는 상당한 부담을 안아야 한다.

2. SyncCharts

도식적인 명세 언어 두 번째로 SyncCharts[4]를 소개하고자 한다. Statechart 등의 상태 기반 모델은 시각적인 표현 방법의 특성상 인터랙티브 시스템의 행위를 명시적으로 기술하기에 매우 적합하다. 그러나 상태 기반 모델은 그러한 시각적 특성상의 한계로 인해 준정형적이거나 비정형적인 의미론(Semantics)만을 가질 수밖에 없다. 상태 기반 모델은 이러한 빈약한 의미론이 가지는 모호함으로 인해 디지털 시스템 등의 반응성 시스템을 설계 및 모델링 하는데 있어 어려움을 가져오기도 하며, 이로 인해 그래픽한 정형기법에서도 정형적인 의미론의 필요성이 대두되게 되었다. 이러한 배경에서 Esterel에서는 자사의 반응성 시스템의 모델링 도구인 Esterel Studio와 완벽히 호환되는 의미론의 토대 위에 Statechart의 외형적 특성을 결합한 새로운 그래픽 모델링 도구인 SyncCharts를 개발하였다.

2.1 SyncCharts의 적용성 분석

1) 계층성(Hierarchy): Macrostate 내부의 또 다른 Macrostate들을 통해 계층성 표시
2) 직교성(Orthogonality): Constellation들 간의 병렬 수행으로 인한 동시성 구현
3) 통신(Communication): 신호들의 즉각적인 Broadcast에 의한 통신
4) 우선권(Preemption): 중단(Abortion)과 미결(Suspension) 등의 보다 향상된 우선권 제공

계층성과 직교성 그리고 통신은 Statechart에서 제공하는 그것들과 거의 동일한 개념으로 사용된다. 우선권은 Esterel Studio와 SyncCharts에서만 제공하는 개념인데, 강한 중단, 약한 중단 그리고 미결의 세 가지 종류의 우선권 방법을 말한다. 우선 첫 번째 강한 중단은 특정 상태에서 전이 조건을 만족시키는 신호가 발생했을 때, 현재 작업을 중지하고 곧바로 다른 상태로 전이되는 기능을 수행하고, 약한 중단은 신호 발생 시 자신이 하던 작업을 해당 클럭까지 완수한 후에 다른상태로 전이한다. 그리고 마지막으로 미결은 전체 반응성 시스템의 부구성요소(Subcomponent)를 특정 조건이 만족될 때까지 동결(Frozen) 상태로 유지시키는 기능을 수행한다. 이러한 다양한 우선권 방법들을 통해 SyncCharts는 복잡한 반응성 시스템의 모델링을 좀 더 효율적으로 할 수 있게 하는 것이다. 따라서, SyncCharts의 게임 소프트웨어에 대한 적용성은 Statechart와 유사하되, 복잡한 게임 소프트웨어에의 적용성이 향상되었다고 정리할 수 있다.

3. VIS

VIS(Verification Interacting with Synthesis)[5]는 검증, 시뮬레이션, 유한 상태 하드에어 시스템의 합성을 종합적으로 수행하는 도구이다. 이 도구는 입력 언어를 Verilog를 이용하고 Fair CTL 모델체킹, Language Emptiness 검사, 조합 순차 동등 검사(Combinational and Sequential Equivalence Checking), Cycle-based 시뮬레이션과 계층적 합성(Hierarchical Synthesis)을 지원한다. VIS는 U.C 버클리대학과 콜로라도 대학(Boulder)과 협력하여 개발되고 있으며 1세대 도구인 HSIS나 SMV와 비교하여 개선된 프로그래밍 환경 및 새로운 호환성을 제공하며 경우에 따라 성능 또한 향상시켜준다.

VIS는 BLIF-MV라는 중간 형태에서 작동되며 BLIF_MV 파일은 vl2mv라는 컴파일러에 의해 생성된다. VIS가 사용하는 입력언어는 Verilog로서 일반적인 Verilog와 다른 점은 비결정성과 기호적 변수를 사용한다는 것이다. 즉 비 결정성 구성자인 $ND는 Wire 변수에서 비결정성을 나타내기 위해서 Verilog에 추가되었다. 이것이 VIS에서 비결정성을 자연스럽게 사용할 수 있도록 하였다. 또한 변수의 값을 기호적으로 명세하고 참조하는 것이 직접적으로 표현하는 것보다 바람직한 경우가 있다. vl2mv가 Veilog에서 C를 사용하는 것과 유사한 열거형을 사용하여 기호적 변수를 표현할 수 있도록 하였다.

BLIB-MV 묘사가 VIS로 읽혀질 때, 계층적 트리 형태로 저장된다. 이는 차례로 하위 모듈을 구성하고 모듈 기능은 Arbitrary Functionality 와 Latch를 가진 게이트의 네트워크에 의해 나타내어진다. 이 계층에 대한 순회는 Unix의 디렉토리에 대한 순회와 유사하고 시뮬레이션과 검증 작업은 어느 계층에서나 가능하다.

3.1 VIS를 이용한 검증

VIS를 이용하여 검증하기 위해서는 먼저 간단하게 검증할 수 있는 형태로 바꾸어야 한다. 상태와 상태 상이의 전이는 유한 상태 기계로 구성되고 완전한 시스템은 각 구성 요소와 연관된 유한 상태 기계를 구성함으로써 얻어지는 유한 상태 기계이다. 그러므로 검증할 때, 첫번째 단계는 시스템을 유한 상태 기계로 나타내는 것이다. 유한 상태 기계로 나타낸 시스템을 VIS의 입력 언어인 Verilog로 설계한 후, VIS를 이용하여 현재 자동 정형 검증에서 가장 많이 쓰이는 두 가지 방법인 Language Containment와 모델체킹 기법으로 검증한다.

3.2 CTL을 이용한 요구 명세 및 모델체킹

모델체킹은 시스템에 대한 시간적 흐름에 따른 행위를 분석한다. 즉 시스템을 유한 상태 기계로 표현하고 이러한 시스템이 만족해야 하는 요구 명세를 시제논리로 표현하게 된다. 따라서 시스템의 Safety 혹은 Fairness, Deadlock을 검증할 수 있다. 또한 자동화된 기법으로Theorem Proving과 같이 사람의 개입이 거의 되지 않는 도구로서 이러한 자동화된 기법은 상당한 이점을 갖는다.

모델체킹은 시스템에 대한 유한 모델을 만들고 이 모델 상에서 만족해야 하는 특성을 검사하는 것으로 주로 하드웨어와 프로토콜 검증에 이용되고 있다. 따라서 어떤 시스템을 정형 검증하기 위해서는 먼저 시스템이 검증 가능한 간단한 형태로 변환되어야 한다. 이를 위해 시스템을 하나의 큰 유한 상태 기계로 구성하고 이 속에는 각각의 상태를 갖고 있는 작은 유한 상태 기계가 있어 이들 간의 상호 관계로 시스템이 작동한다. 이런 성질은 우리가 실세계에서 접하는 시스템을 모델링하기 좋은데 그 이유는 계층을 쉽게 표현할 수 있기 때문이다. 모델체킹 기법은 유한 상태 기계로 표현된 모델에 대해서 검사하므로 종료가 보장되고 이로써 상태 공간의 모든 경우에 대해서 검사하게 된다. 시스템에 대해 검사하려는 요구명세는 시제 논리(Temporal Logic)로 표현한다.

시제 논리는 시간을 표현할 수 있는 연산자를 사용하여 특성을 시간에 따른 시간들의 순서를 표현할 수 있다. 시제 논리에는 크게LTL(Linear Temporal Logic)과 CTL(Computational Tree Logic)이 있다. LTL은 하나의 시간 흐름에 대해서만 고려하는 방법이고 CTL은 트리 형태로 여러 갈래로 나뉘는 시간 흐름에 대한 표현 방법이다.

VIS에서 사용되는 방법은 CTL이다. CTL은 상태 전이 그래프로부터 유도된다. 상태 그래프는 초기 상태로부터 무한한 경우를 나타내는 트리로 표현된다. 이 트리에서 나타나는 모든 경로(Path)는 모델링 되는 모든 가능한 계산을 표현한다. CTL은 이러한트리와 같은 분기 구조를 묘사하는 연산자를 가지고 있기 때문에 분기 시간 논리로 분류된다. CTL 로 표현되는 식은 단위 명제와 명제 논리의 부울 합성자(Boolean Connective), 시제 연산자로 구성된다. 각 시제 연산자는 두 부분으로 구성되는데, 경로 한정자(Path Quantifier: A, E)와 시제 구성자(Temporal Modality: F, X, G, U)이다. 각 Formula가 q라는 상태에서 적용된다고 가정하며 경로 한정자에서 A는 "q"에서 시작하는 모든 경로를 의미한다. E는 "q"에서 시작하는 어떤 한 경로를 나타낸다. 시제 구성자에서 F는 "해당 경로의 어떤 한 상태"를, X는 "다음에, G는 "해당 경로의 모든 상태"를 의미하며, U는 "~일 때까지"를 나타낸다. 그래서 AG safe가 q에서 만족되려면, q에서 시작하는 모든 경로(A)의 모든 상태(G)들이 safe라는 Formula를 만족해야 한다. 또한AF safe 가 q에서 만족되려면, q에서 시작하는 모든 경로(A)에 대해, 각 경로에서 적어도 하나의 상태(F)가 safe를 만족해야 하며, EG safe가 q에서 만족되려면, q에서 시작하는 경로 중 적어도 하나의 경로(E)의 모든 상태(G)가 safe를 만족해야 한다. 마지막으로 EF safe 가 q에서 만족되려면, q에서 시작하는 경로 중 적어도 하나의 경로(E)에서 적어도 하나의 상태(F)가 safe를 만족해야 한다.

3.3 VIS의 적용성 분석

VIS 또한 상태(State)를 기반으로 하는 정형 기법이라는 측면에서 게임 소프트웨어에의 적용성은 Statechart 및 SyncCharts와 유사하다고 할 수 있다. 다만, VIS 또는 유사한 모델체킹을 지원하는 정형기법은 게임 알고리즘 설계 분야에서 그 적용성을 추가적으로

발견할 수 있다. 간단한 자동차 경주 게임을 설계한다고 가정하자. 사용자가 도로의 상황과 신호등의 상태 등을 파악하고 제어하면서 결승점에 도달하는 모든 과정을 상태로 정의하여 VIS로 모델링할 수 있다. [그림 2]는 이러한 VIS 모델의 일부를 보여 준다.

```
typedef enum {YES, NO} boolean;
typedef enum {START, SHORT, LONG}timer_state;
typedef enum {GREEN, YELLOW, RED} color;

module main(clk);
input clk;
color wire farm_light, hwy_light;
wire start_timer, short_timer, long_timer;
boolean wire car_present;
wire enable_farm, farm_start_timer, enable_hwy,
hwy_start_timer;
assign start_timer = farm_start_timer ||
hwy_start_timer;
timer timer(clk, start_timer, short_timer, long_timer);
sensor sensor(clk, car_present);
farm_control farm_control(clk, car_present,
enable_farm, short_timer, long_timer,
farm_light, farm_start_timer, enable_hwy);
hwy_control hwy_control (clk, car_present,
enable_hwy, short_timer, long_timer,
hwy_light, hwy_start_timer, enable_farm);
endmodule
```

그림 2. 자동차 경주 게임에서 신호등 부분에 대한 VIS 모델 일부

또한 결승점에 먼저 도달하는 데 필요한 여러 가지 판단 자료 및 제어 동작들이 모델이 만족해야 하는 특성이라고 할 수 있으며, 이것들은 시제 논리로 표현될 수 있다. 이제 이러한 시제 논리에 대하여 모델체킹을 수행한다. 이 때 모델체커에 의해 생성되는 반례는 어떤 특성을 만족해야만 목적지에 이르는 일련의 상태의 집합을 찾을 수 있는가를 알려주게 된다. 바로 이러한 상태의 집합이 길 찾기 게임의 솔루션이고 이 솔루션을 염두에 두고 게임 소프트웨어의 알고리즘을 설계할 수 있다.

IV. 결론 및 토론

본 논문에서는 인터랙티브 소프트웨어에 일반적으로 적용되어 온 정형기법을 중심으로 게임 소프트웨어에의 적용성을 살펴보았다. 정형기법으로는 도식적인 명세 언어와 문자 기반의 명세 언어가 있다. 도식적인 명세 언어 중 게임 소프트웨어 분석 및 설계에 적용하기에 적합한 정형기법에는 State-diagram 기반의 Statechart와 SyncCharts가 있다. 이러한 기법은 게임 기획자나 설계자, 프로그래머가 쉽게 이해할 수 있으며, 전체적인 구조나 운용을 이해하기 쉽다는 장점을 지니고 있다. 하지만 시스템의 구조나 다양한 시스템의 특성을 복잡하게 설계할 경우, 설계를 이해하고 분석하기가 까다로워진다.

VIS 등의 텍스트 기반의 정형명세 언어는 주로 유한 상태 기계로서 시스템의 행위를 명세한다. 또한 이러한 도구들은 대부분 SRS를 시제논리 등의 논리 형태로 표현한다. 논리 형태로 표현된 요구명세는 요구명세 자체를 다양한 방법으로 검증할 수 있으나 자연어를 논리 형태로 바꾸는 번거롭고 복잡한 작업이 요구되며, 또한 논리에 대한 지식이 요구된다. 한편, VIS와 같은 정형기법은 모델체킹을 통해 행위에 대한 검증이 가능하다. 시제 논리를 요구명세로 받아들여 설계 명세된 시스템의 성질을 검증할 수 있다. 이것은 정형기법을 게임 알고리즘 설계 분야에 적용할 수 있게 해 준다.

각 정형기법은 [표 1]에 나타나는 바와 같이 나름대로 장단점을 지니고 있으며, 또한 게임 소프트웨어에의 적용성도 그 각도가 다양하다. 따라서 본 연구에서는 이러한 기법을 통합적으로 게임 소프트웨어 분석 및 설계에 사용할 것을 제안한다.

표 1. 각 정형기법의 장단점

정형기법	장점	단점
Statechart	도식적, 직관적, 높은 이해도, 알고리즘 표현의 용이성	복잡도 처리의 어려움 알고리즘 도출 능력 부족
Syncchartes	도식적, 직관적, 높은 이해도, 알고리즘 표현의 용이성, 복잡도 처리 능력 향상	알고리즘 도출 능력 부족
VIS	알고리즘 도출의 용이성	표현의 어려움

참 고 문 헌

[1] J. Rushby, *Formal Methods and the Certification of Critical Systems*, Technical Report CSL-93-7, SRI International, Menro Park, CA, 1993

[2] A. Diller, *Z : An Introduction to Formal Methods*, John Wiley Sons, 1992.

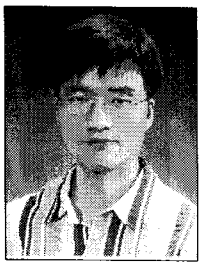
[3] D. Harel, "Statechart : A Visual Formalism for Complex Systems," Science of Computer Programming Vol.8, pp.231-274, 1987.

[4] http://www.esterel-technologies.com/

[5] R. Alur and T. Henzinger, "VIS: A system for Verification and Synthesis," Proceedings of the 8th International Conference on Computer Aided Verification, New Brunswick, NJ, pp.428-432, July 1996.

저 자 소 개

손 한 성(Han-Seong Son) 정회원



- 1993년 2월 : 서울대 원자핵공학과 (공학사)
- 1995년 2월 : KAIST 원자력공학과 (공학석사)
- 2000년 2월 : KAIST 원자력공학과 (공학박사)
- 2000년 3월 ~ 2002년 3월 : KAIST 신형원자로연구센터 위촉연구원
- 2002년 4월 ~ 2004년 7월 : 한국원자력연구원 선임연구원
- 2003년 3월 ~ 2005년 8월 : 중부대학교 게임학과 강사

<관심분야> : 게임 기획, 게임 소프트웨어공학, 소프트웨어 분석 및 설계