OCL을 이용한 UML 2.0 행위 모델의 시간 일관성 검사

한아림○ 전상욱 홍장의* 배두환
한국과학기술원 전자전산학과
*충북대학교 전기전자컴퓨터공학부
{arhan○, sujeon, bae}@se.kaist.ac.kr *jehong@chungbuk.ac.kr

Time Consistency Checking on UML 2.0 Behavioral Models Using OCL

Ah-Rim Han○ Sang-Uk Jeon Jang-Eui Hong* Doo-Hwan Bae
Division of Computer Science, Dept. of EECS KAIST
*School of Electric and Computer Engineering, Chungbuk National University

요 약

UML 2.0의 시퀀스 다이어그램은 실시간 특성을 표기할 수 있는 표기법이 추가됨으로써 시간의 엄격성이 중요하게 여겨지는 임베디드 소프트웨어의 행위를 보다 정확하고 상세하게 모델링 할 수 있게 되었다. 시퀀스 다이어그램은 전체 시스템의 부분적인 행위를 기술하여 같은 행위가 여러 다이어그램에 걸쳐서 기술되어지므로 이들 다이어그램들은 서로 일관성을 가져야 하는데 복잡한 시스템에서 일관성을 검사하는 것은 어려운 일이다. 본 논문에서는 행위 모델을 표현하는 시퀀스 다이어그램과 다른 다이어그램에 기술된 실시간 특성들의 일관성을 자동으로 검사할 수 있는 방법을 제안한다. 먼저 서로 다른(inter) 다이어그램들 사이의 시간적인 일관성을 유지하기 위한 제약사항을 파악하여 정의한다. 그리고 모델을 구성하는 인스턴스들에 대한 제약사항을 OCL로 기술하는데 이는 UML 2.0이 발표되면서 모델의 의미를 정확하고 상세하게 표현할 수 있는 메타 모델과 그들의 연관관계가 추가되고 명확하게 정의됨에 따라 가능하게 된 것이다. 이렇게 OCL로 정형화되어 기술된 식은 UML 2.0 메타 모델을 이용한 어느 모델에나 적용 가능하며 모델의 문제점을 발견하고 정확한 모델을 구축하도록 도와준다.

1. 서 론

임베디드 소프트웨어의 크기가 커지고 복잡도가 증가함에 따라서 상세하고 명확한 모델의 기술에 대한 필요성이 대두되었다. 이러한 흐름에 맞추어 최근에 UML을 이용하여 임베디드 소프트웨어를 모델링하는 연구들이 발표되었다[1, 2]. 또한 최근 발표된 UML 2.0[3]은 기존의 UML 1.x보다 정형화된 의미를 가지게 됨으로써 보다 정확한 모델의 기술이 가능하게 되었다. 그 중에서도 시퀀스 다이어그램은 실시간 특성(Real-time property)에 관한 표기법이 추가되면서 실시간 시스템(Real-time system)과 같이 시간의 엄격성이 중요하게 여겨지는 도메인의 모델링이 가능하게 되었다.

UML 시퀀스 다이어그램은 시나리오에 대한 행위를 기술하기 때문에 한 시스템에 대해서 여러 장의 시퀀스 다이어그램이 작성된다. 이 때 각 시퀀스 다이어그램들 마다 시스템이 만족시켜야 하는 실시간 특성들이 기술된다. 실시간 특성들이 여러 다이어그램에 걸쳐 기술되다 보면 모델러의 실수로 인해 이들 특성들 간의 일관성(consistency)이 유지되지 않을 수 있다. 임베디드 소프트웨어 모델링에서 실시간 특성들 간의 일관성이 위배되는 경우 개발 후에 문제점이 발생할 수 있고 그에 따라 소프트웨어 유지 보수비용이 많이 발생할 수 있다. 따라서 여러 다이어그램에 걸쳐 기술되어 있는 실시간 특성들이 일관성을 유지하고 있는지에 대한 검사가 필요하다.

본 논문에서는 UML 2.0 시퀀스 다이어그램에 기술된 실시간 특성이 다른 시퀀스 다이어그램이나 상태머신 다이어그램에서 기술된 실시간 특성과 일관되게 기술되었는지 검사하는 규칙을 정의하고 이를 OCL[4]로 표현한다. UML 2.0이 발표되면서 모델을 구성하는 메타모델과 그들 간의 연관관계가 많이 추가되어 보다 정확한 모델을 표현하게 되었고 이들 정보에 기반하여 메타모델 인스턴스들에 대한 제약 사항을 OCL로 기술함으로써 모델의 정확성을 검증할 수 있다. 본 논문에서 제시하는 제약사항들을 검증함으로써 임베디드 소프트웨어에서 실시간 특성을 기술하는 UML 행위 모델의 일관성(consistency)을 검증할 수 있고 그에 따라 소프트웨어 개발 초기에 문제점을 발견하여 개발 비용을 절감할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 UML 모델이 지켜야 할 일반적인 특성들을 OCL로 기술한 기존 연구들에 대해 언급하고 본 연구와의 차이점에 대하여 기술한다. 3장에서는 UML 2.0 시퀀스 다이어그램에서 시간과 관련된 기술이 다른 다이어그램과 일관되게 기술하기 위하여 만족해야 하는 제약사항을 정의하고 이를 OCL로 정형화 한다. 마지막으로 4장에서는 결론을 맺고 향후 연구에 대해 기술한다.

2. 관련 연구

UML 모델을 구성하는 인스턴스들의 제약 사항을 OCL로 기술하여 모델의 정확성을 검증하는 연구들이 여럿 발표되었다. OCLE[5]는 OCL식에 대한 평가를 지원하는 도구로써 XMI 1.0 형태로 표현된 UML 1.5 모델과 OCL로 표현된 규칙을 입력 받아, OCL 식이 주어진 UML 모델에서 만족되는지를 검사하여 만족되지 않는 목록을 출력한다. Ritcher와 Gogolla[6]는 USE[8] 도구를 UML 메타모델 수준에서 적용하여 UML 1.3 모델의 오류들을 OCL 제약 사항을 통하여 검증하였다. 객체지향 방법론의 분석 모델이 항상 충족해야 하는 구조적 제약 사항들을 OCL로 기술하여 다양한 구현 환경 시스템을 구축할 때 재사용 할 수 있게 한 연구도 있었다[7]. 위에서 살펴 본 연구들은 UML 1.x의 일반적인 하나의 모델에 대한 검증을 수행하고 있다. 본 논문에서는 UML 2.0을 이용하여 임베디드 소프트웨어의 행위를 모델링할 때 표기되는 시간 특성이 여러 다이어그램들 사이에서 일관성 있게 표현되기 위한 제약 사항을 OCL로 기술하였다. UML 2.0을 기반으로 한 모델은 기존의 UML 1.x에 비해 자세하고 정확한 메타모델과 연관관계 정보를 가지고 있어서 메타모델 인스턴스를 이용한 모델 검증이 용이하다.

3. UML 2.0 행위 모델 간의 시간 제약 사항 규칙 정의 및 기술

UML 1.x 시퀀스 다이어그램은 단순히 메시지들의 순서로 행위를 표현하여 정형적인 의미를 나타내지 못했던 것과 달리 UML 2.0 시퀀스 다이어그램은 행동의 연관된 지점을 가리키는 EventOccurrence(이하: *EO*)라는 메타 모델을 추가하여 메시지를 보내는 지점의 *EO*인 sendEvent, 메시지를 받는 지점의 *EO*인 receiveEvent들의 순서쌍으로 표현된다. UML 2.0에서 새로 추가된 시간을 표현하는 메타모델은 <표 1>에 정리되어 있고 이는 SimpleTime이란 패키지에 속한다.

<표 1> UML 2.0 시간 표현 관련 메타모델

종류	표현	설명
duration observations	Code d= duration	메시지 Code를 보내고 받는데 걸리는 시간을 측정하여 d에 기록한다
duration constraints	{d..3*d} {0..13}	두 *EO*가 발생하는 시간 간격의 최소값과 최대값을 나타낸다
time observations	t=now	현재 시점에서의 시간을 t에 기록한다
time constraints	{t..t+3}	*EO*가 발생하는 시간 한계의 최소값과 최대값을 나타낸다

본 장에서는 새롭게 정의된 메타 모델 인스턴스들에 대한 정보를 바탕으로 실시간 특성을 표기할 때 같은 행위에 대하여 일관되게 기술하는 규칙을 정의하고 이를 OCL로 나타낸다.

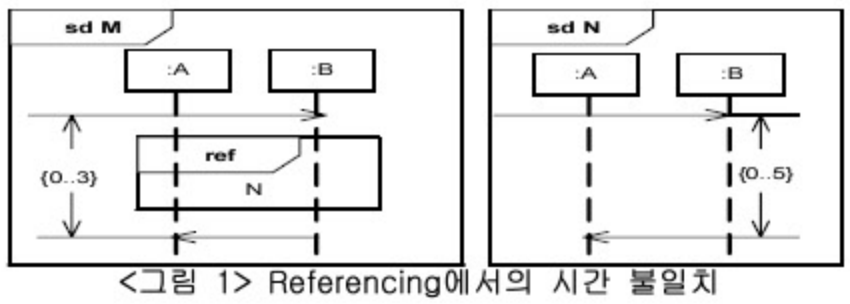
3.1 시퀀스 다이어그램들 간의 불일치

시퀀스 다이어그램에서는 Referencing을 이용하여 여러 인터랙션들에서 공유되는 공통된 부분을 하나의 시퀀스 다이어그램으로 기술한 뒤 참조하는 형태로 다른 인터랙션의 내용을 복사하여 사용할 수 있다. 따라서 Referencing을 사용 할 때에서 인터랙션 안에 또 다른 인터랙션이 있다는 것을 고려하여 불일치가 발생하지 않도록 주의하여 모델링을 하여야 한다. 시퀀스 다이어그램에서 Referencing을 사용하여 시간과 관련된 기술을 할 때 다른 다이어그램과 일관되게 기술하기 위하여 만족해야 하는 제약 사항과 OCL 기술은 다음과 같다.

• 정의: Referencing을 사용하여 시퀀스 다이어그램을 모델링 할 때, 이를 포함하는 인터랙션의 시간 구간 제약이 더 크거나 같아야 한다.

```
Context DurationInterval
inv DurationIntervalConstraint:
 let getInteractonUse(up:EventOccurrence,
    down:EventOccurrence):Set(InteractionFragment)   =
      InteractionUse.allInstances->select(iu|
         iu.refersTo.fragment->exists(up,down|up and down))
 getLargerRegionDurationInterval(self)->forAll(
    l_durationInterval:DurationInterval|
    let interactionUse : Set(InteractionUse) =
      getInteractonUse(
      getEventOccurrenceFirstDuration(self),
      getEventOccurrenceLastDuration(self))
-- use single Referencing
    interactionUse->size() = 1 and
    isInInteractionUse(l_durationInterval,
      interactionUse) = false implies
         l_durationInterval.max >= self.max and
         l_durationInterval.min >= self.min)
```

Referencing이 정의된 곳에서의 시간 간격 제약 단위의 최소, 최대값이 실제 참조 되는 곳에서 더 큰 구간의 시간 간격 최소, 최대값보다 크지 않아야 한다. 제약 사항을 위반하는 모델의 예는 <그림 1>과 같다.

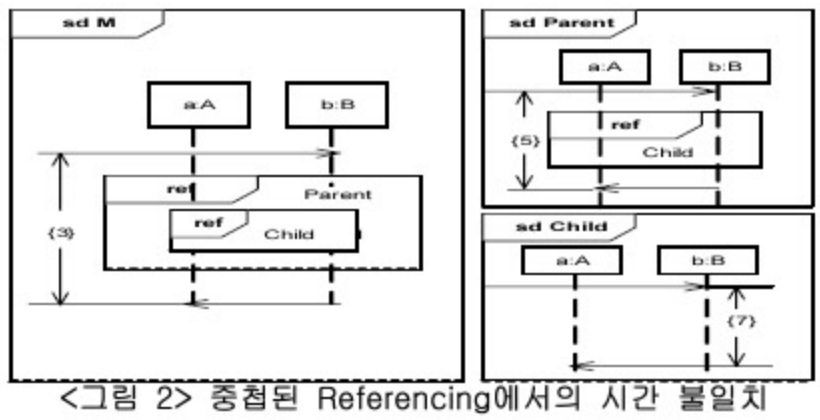


<그림 1> Referencing에서의 시간 불일치

• 정의: 중첩된 Referencing을 사용할 때 자식 프래그먼트를 포함하는 부모 프래그먼트에서의 시간 구간 제약은 더 크거나 같아야 한다.

```
-- use Nested Referening
    InteractionFragment.allInstances->forAll(childFrag,
parentFrag: InteractionFragment |
    childInterFrag.enclosingInteraction = parentInterFrag
implies    getDurationInterval(parentFrag)->
 forAll(parentDurationInterval: DurationInterval|
getDurationInterval(childFrag)->
 forAll(childDurationInterval: DurationInterval|
parentDurationInterval.max >= childDurationInterval.max and
 parentDurationInterval.min >= childDurationInterval.min)))
```

제약 사항을 위반하는 모델의 예는 <그림 2>와 같다.

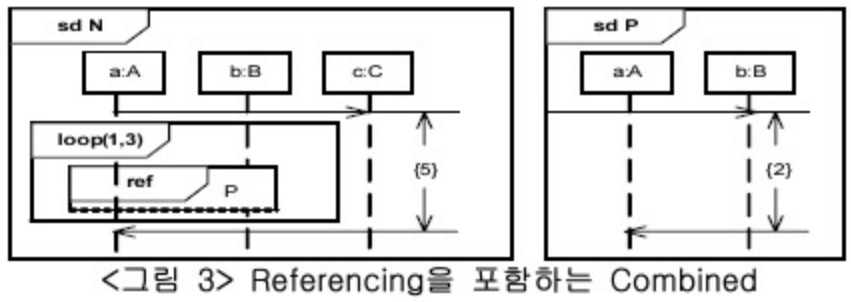


<그림 2> 중첩된 Referencing에서의 시간 불일치

• 정의: Referencing을 포함하는 프래그먼트가 *loop* Combined Fragment일 때 해당 반복회수만큼 루프를 수행하여 걸리는 시간의 총 합보다 프래그먼트 밖의 시간 구간 제약이 더 크거나 같아야 한다.

```
-- Referencing with outer loop CombinedFragment
 InteractionFragment.allInstances->forAll(reference,
combinedFrag: InteractionFragment |
    reference.enclosingInteraction = combinedFrag
and  combinedFrag.interactionOperator = #loop
implies
getLargerRegionDuration(combinedFrag)->forAll(l_durationInt
erval: DurationInterval |
getDurationInterval(reference)->forAll(referenceInterval:
DurationInterval|
l_durationInterval.max >=
(combinedFrag.enclosingOperand.guard.maxint -
combinedFrag.enclosingOperand.guard.minint +1)  *
referenceInterval.max and
l_durationInterval.min >=
(combinedFrag.enclosingOperand.guard.maxint -
combinedFrag.enclosingOperand.guard.minint +1)  *
referenceInterval.min)))
```

제약 사항을 위반하는 모델의 예는 <그림 3>과 같다.



<그림 3> Referencing을 포함하는 Combined Fragment에서의 시간 불일치

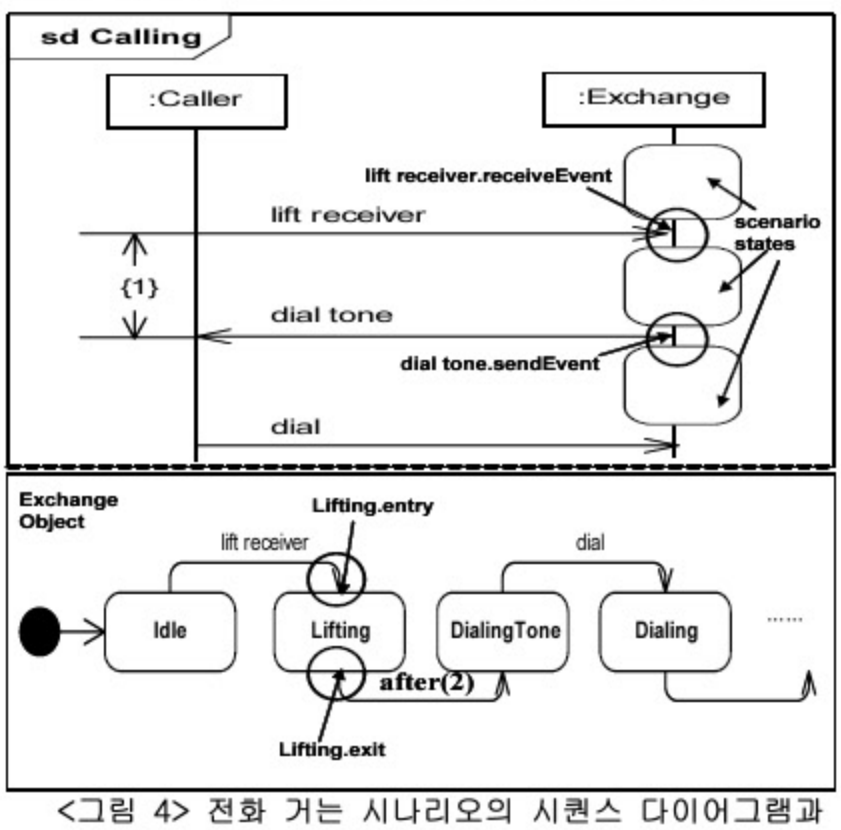
3.2 시퀀스 다이어그램과 상태머신 다이어그램 간의 불일치

시퀀스 다이어그램은 객체들 간(inter-object)에 메시지를 어떻게 주고받는지를 나타내는 반면 상태머신 다이어그램은 하나의 객체(intra-object)가 생성되어 소멸 될 때까지 어떤 상태를 가지는지를 나타낸다. 이들 두 다이어그램에서 같은 상태를 나타내는 시점에 대한 시간 제약은 일관성을 가져야 하는데 이를 정의하고 OCL로 기술하면 다음과 같다.

• 정의: 시퀀스 다이어그램에서의 두 *EO*가 같은 생명선(Lifeline) 상에 있고 순서를 가지며(ordered) 이들이 각각 상태머신 다이어그램에서 어떤 상태의 들어오는(entry) 점과 나오는(exit) 점과 일치할 때 시퀀스 다이어그램에서 두 *EO*사이의 시간 제약과 상태머신 다이어그램에서 각 *EO*에 상응하는 진입점 상태부터 출구점 상태까지 도달하는데 소요되는 시간은 같아야 한다.

```
-- check time constraint of the same state between sequence
diagram and statemachine diagram
EventOccurrence->forAll(e1, e2: EventOccurrence |
 e1.covered = e2.covered and
  GeneralOrdering->exists(GO: GeneralOrdering |
  GO.before = e1 and GO.after = e2 or
  GO.before = e2 and GO.after = e1) and
   State->exists(s1, s2 : State|
    e1 = s1.entry and e2 = s2.exit)
implies
   getDurationSequece(e1,e2)
          = getDurationStateMachine(s1.entry, s2.exit)
)
```

<그림 4>는 전화를 거는 시나리오를 기술한 것으로 사용자와 교환기의 관계를 나타내는 시퀀스 다이어그램과 교환기의 상태 변화를 나타낸 상태머신 다이어그램에서 같은 행동에 대한 시간 제약 기술이 다르게 나타나고 있는 예를 보여준다.



<그림 4> 전화 거는 시나리오의 시퀀스 다이어그램과 교환기 객체의 상태 머신 다이어그램

상태머신 다이어그램에서 교환기는 Idle상태에 있다가 사용자가 수화기를 들면 Lifting상태로 변하고 다이얼 톤을 보낸 후 DialingTone상태로 들어가게 된다. 이 때 수화기를 들고 다이얼 톤을 보내는데 걸리는 시간, 즉 Lifting상태에 머물러 있는 시간은 시퀀스 다이어그램에서 1 시간 단위(time unit)라고 명시되어 있다. 반면 상태머신 다이어그램에서는 *after*(2) 라고 명시되어 Lifting상태로 들어온 시점부터 2 시간 단위 후에 DialingTone상태로 전이된다고 명시되어 있다. 이는 시퀀스 다이어그램에서 나타나는 같은 상태에서의 시간 제약 사항과 일관되지 않는다. 기존 UML 1.x 에는 이벤트 각 끝점의 구분이 없었기 때문에 이런 추적이 불가능하였지만 UML 2.0 시퀀스 다이어그램과 상태머신 다이어그램은 각각 메시지를 받고 주는 지점과 상태로 들어오고 나가는 지점이 연관관계를 가지고 있기 때문에 이들 메타모델 인스턴스 정보를 비교함으로써 바로 검사가 가능하다.

4. 결론

본 논문에서는 UML 2.0 행위 모델에서 실시간 특성을 기술할 때 지켜져야 될 규칙을 정의하고 이를 OCL로 기술하였다. UML 2.0은 기존 UML 1.x에 비해 모델을 구성하는 메타모델과 그들의 연관관계가 많이 추가되어 보다 정확한 모델을 표현하게 되었고 이들 정보에 기반하여 메타 모델 인스턴스들에 대한 제약사항을 OCL로 기술함으로써 모델의 정확성 검증할 수 있다. UML 모델이 지켜야 할 일반적인 제약 사항을 OCL로 기술한 기존의 연구들과 달리 본 연구에서는 UML 2.0에 새로 추가된 시간 특성이 여러 다이어그램들 사이에서 일관성 있게 표현되기 위하여 필요한 실시간 특성과 관련한 제약 사항을 기술하였다. 현재 OCL로 기술된 제약 사항을 해석하여 특정 UML 2.0 모델이 이를 만족하는지 검증하는 도구를 개발하고 있다. 이를 통해 UML 기반 임베디드 소프트웨어 모델링에서 위배되어질 수 있는 일관성을 보다 쉽게 검증할 수 있게 될 것이다. 또한 규칙이 OCL로 기술되어 있기 때문에 새로운 규칙의 추가 및 기존 규칙의 변경 및 삭제가 용이하다.

본 논문의 향후 연구는 다음과 같다. 우선 UML 2.0 시퀀스 다이어그램에 기술되는 실시간 특성의 일관성과 관련하여 보다 많은 규칙을 찾아내고자 한다. 또한 실시간 특성 뿐 아니라 다중 프로세서 환경에서 고려해야 하는 병렬성(concurrency), 분산성(distributed) 등과 같이 임베디드 소프트웨어 모델링에 필요한 여러 특성들이 지켜야 할 제약 사항들에 대해서도 파악하고자 한다. 마지막으로 현재 개발 중인 도구를 완성하여 본 논문에서 제시한 다양한 규칙들을 여러 UML 모델에 적용해 보고자 한다.

참고문헌

[1] Gjalt de Jong, "A UML-Based Design Methodology for Real-Time and Embedded Systems", DATE 2002, 2002
[2] Grant Martin, Luciano Lavagno, and Jean Louis-Guerin, "Embedded UML: a merger of real-time UML and co-design", CODES 2001, Copenhagen, 2001
[3] UML 2.0 Superstructure Specification (ptc/04-10-02), Available from : http://www.omg.org
[4] UML 2.0 OCL Specification (ptc/03-10-14), Available from : http://www.omg.org
[5] OCLE (OCL Evaluator), Internet: http://lci.cs.ubbcluj.ro/ocle/index.htm
[6] Mark Richters, Martin Gogolla, "Validating UML Models and OCL Constraints", Third International Conference, York, UK, Proceedings, volumn 1939 of LNCS, Springer, 2000
[7] 채흥석, 염근혁, "UML 분석 모델의 구조적 제약사항에 대한 OCL 기반의 명세 및 검증", 정보과학회논문지 제 33권 제2호, 2006.
[8] University of Bremen, USE (UML-based Specification Environment) tool, Internet : http://db.informatik.uni-bremen.de/projects/USE