이벤트 위주의 실시간 OCL과 그 응용
(An Event-Driven Real-Time OCL and Its Application)

최 성 운 † 이 영 환 ††
(Sung Woon Choi) (Young Whan Lee)

요 약 OCL(Object Constraint Language)은 UML 메타모델을 정밀하게 명세화하기 위해서 UML 의미론의 도큐먼트에 사용되어졌다. 그리고 UML은 실시간 UML, 웹 개발 UML 등과 같이 다양한 시스템을 개발하기 위해서 확장되었다. 특히 실시간 시스템을 개발할 때 적시성, 동시성, 예측성, 신뢰성이 고려되어야 한다. 이에 따라 실시간 UML을 정밀하게 표현하고 구현을 쉽게 하기 위해서 OCL을 사용해야 하지만 현재의 OCL로 실시간을 묘사하기에는 부적합하다. 본 논문에서는 실시간 시스템을 개발하는데 있어서 실시간 언어로 쉽게 변환이 가능하도록 이벤트 위주로 실시간 OCL을 제안하였고 그 효용성의 검증으로서 권선기 시뮬레이터 개발에 응용하였다.

Abstract OCL was used in the UML Semantics document to specify the well-formedness rules of the UML metamodel. UML was extended to apply it to system developments of several fields, for example, real-time UML and web applications with UML. In particular, the dependability is important in designing and building hard real-time system. Thus OCL is needed to express real-time UML formally and so it must extended. In this paper, we extend OCL to define event-driven real-time OCL that can be easily translated to Real-Time languages. Also we apply event-driven real time OCL to the development of Nrcl simulator.

1. 서 론

실시간 시스템은 그 정확성이 결과의 정확함과 아울러 결과가 산출되는 시간에 의해 결정되는 시스템으로서 관점에 따라 구분되는데 응답시간에 따라 하드 실시간 시스템(Hard Real-time System)과 소프트 실시간 시스템(Soft Real-time System)으로 구분되며 시스템 구성 형태에 따라 독점 시스템(Proprietary System)과 개방 시스템(Open System)구분되고 시스템의 구조에 따라 중앙 집중화 시스템과 분산 실시간 시스템으로 구분할 수 있다.

실시간 시스템은 기능 요구사항과 시간 요구사항을 만족해야한다. 기능 요구사항은 실시간 시스템의 목적을 만족하도록 정의해야 하며 시간 요구사항은 이벤트 발생 사이의 시간 제한을 만족하도록 정의해야 한다. 시간 제한은 시스템의 요구사항으로부터 발생하거나, 어플리케이션 환경에 따라 일어난다. 실시간 어플리케이션은 빠른 계산 속도, 고속 인터럽트 처리, 다량의 I/O처리, 탄력적인 고장 방지 능력을 요구하며 정확한 결과를 내기 위해서는 높은 시스템 사양을 요구한다. 따라서, 한정된 시스템에서도 정확한 결과를 얻기 위해서 신뢰할 수 있는 모델링을 하여야 하며 실시간 언어를 사용하여 구현해야한다[1, 2].

이에 따라 UML을 사용한 실시간 시스템 소프트웨어 개발 방법론이 이루어 졌다[3, 4]. 그러나 정밀한 모델링을 할 때에는 OCL을 사용해야 함에도 불구하고 아직까지 OCL을 사용한 실시간 시스템의 개발 방법론은 이루어지지 않았다.

본 논문에서 우리는 UML 중에서 OCL을 이벤트 위주의 실시간 시스템에 맞추어 확장하여 시스템 요구사항을 가시적으로 보장하고 실시간 언어로 쉽게 구현이 가능하도록 하고자한다. 그리고 사례연구로서 권선기 시뮬레이터 모델링에 응용하여 우리가 제안한 이벤트 위주의 실시간 OCL의 효과를 얻는다.

본 논문의 구성은 다음과 같다. 1절에서는 문제를 제기한 후 2절에서 실시간 시스템 요구사항과 OCL에 대

† 비 회 원 : 명지대학교 컴퓨터학부 교수
choisw@mju.ac.kr
†† 정 회 원 : 대전대학교 기초과학부 교수
ywlee@dju.ac.kr
논문접수 : 2001년 5월 8일
심사완료 : 2001년 9월 11일

하여 기술한다. 3절에서는 OCL을 사용하는 객체지향 개발 방법론과 실시간 시스템 개발 방법론에 관련된 연구에 대하여 기술한다. 4절에서는 본 논문의 핵심으로서 이벤트 위주의 실시간 OCL을 제안한다. 여기서는 기존의 OCL을 실시간 시스템에 설계에 적용할 수 있도록 확장한 것이다. 5절에서는 본 논문에서 제안한 방법에 따라 OCL을 이용하여 권선기 시뮬레이터를 모델링한다. 마지막으로 6장에서 결론을 도출한다. 특히 부록으로 확장된 OCL 구문과 권선기 프로그램을 첨부한다.

2. 실시간 시스템 요구사항과 OCL

2.1 실시간 시스템 요구사항

실시간 컴퓨팅 시스템(Real-time Computing System)이란 외부의 자극(비동기 이벤트)에 대해 예상되는 시간 내에 혹은 제한된 시간 내에 기능을 수행하고 응답하는 시스템을 말한다[1]. 실시간 시스템은 시스템에 과부하가 걸린 상태에서도 다음의 조건을 만족해야 한다[2].

- 적절한 반응이 적시에 일어나야 하는 적시성(Timeliness)
- 병렬 처리 능력을 제공할 경우 만족해야하는 동시성(Simultaneousness)
- 외부의 입력에 대해서 시스템의 반응을 예측할 수 있는 예측성(Predictability)
- 시스템의 신용 상태에 대한 일반적인 요구사항인 신뢰성(Dependability)-정확성, 견고성, 지속성-등의 조건

2.2 OCL(Object Constraint language)

OCL은 UML 메타모델(Matamodel)의 적격인 규칙(Well-formedness Rule)을 명세화하기 위하여 UML 의미론의 도큐먼트에 사용되어졌다.

객체지향 모델링에 있어서 클래스 모델과 같은 그래픽 모델은 명료하지 못하고 애매 모호하여 명세화가 충분히 이루어지지 않는다. 모델의 객체에 대하여 부가적인 제약들을 묘사할 필요가 있다. 그러한 제약들은 가끔 자연어로 묘사되어 진다. 이러한 습관은 항상 애매모호성을 불러일으킨다. 애매 모호성이 없는 제약들을 쓰기 위해서 소위 형식언어라는 것이 개발되었다. 전통적인 형식언어들은 수학적 지식을 가진 사람이 사용할 수 있으나 사업가나 시스템 설계자들이 사용하기는 어렵다. OCL은 이러한 차이가 없도록 개발되어졌다. OCL은 읽고 쓰기가 쉬운 형식언어이다.

OCL은 순수한 표현언어이다. 이것은 모델에서 어떤 것도 변하지 않는다는 것이다. 즉, 시스템의 상태가 OCL의 표현 때문에 결코 변하지 않을 것이라는 것을 의미한다. 모든 링크를 포함한 모든 객체에 대한 모든 값들은 변하지 않는 것이다. OCL표현이 값을 구할 때마다 OCL표현은 단지 값을 전달할 뿐이다.

OCL은 프로그래밍 언어가 아니다. OCL이 모델링 언어이기 때문에 직접적으로 실행가능하지 않다.

OCL은 타입언어다. 그래서 OCL표현은 타입을 가지고 있다. 올바른 OCL표현에서 모든 타입은 일치되는 타입이어야 한다. 예를 들어, 문자열과 정수를 비교할 수 없다. OCL범위에서 타입은 UML범위에서 일종의 분류가 되어질 수 있다.

OCL은 여러 목적으로 사용되어 진다. 요약하면 다음과 같다.

- 클래스 모델에서 클래스와 타입에 대한 불변성을 구체적으로 보임
- 스테레오 타입에 대한 타입 불변성을 구체적으로 보임
- 연산과 메소드에 대한 "pre- and post condition"을 기술
- 경계(Guards) 조건을 기술
- 제어할 수 있는 언어
- 연산에 대한 제약들을 구체적으로 기술함

UML의미론의 도큐먼트에서, OCL은 추상구문에서 메타클래스(Meta-Classes)에 대한 불변성으로 적격규칙에 사용되었다.

3. 관련연구

UML 제안서[5]에서 UML 메타모델을 정밀하게 묘사하기 위해 UML 의미론의 도큐먼트에 OCL을 사용하였다. J. Wermer와 A. Kleppe[6]는 정밀한 모델링을 하는 데에는 OCL을 사용하여 모델링 하기를 권장하고 있다. D. F. D'souza와 A. C. Willis[7]는 컴포넌트 개발 방법론으로서 Catalysis를 만들면서 기존의 OCL를 확장하여 사용하고 있으며 분석 및 설계를 토대로 시스템을 구현할 때 일관성과 편리성을 가지게 하였다.

실시간 시스템에서는 시간제약(Timing Constraints)의 처리 방법이 중요하다[8]. 실시간 시스템에서 시간제약을 처리하는 많은 연구가 있었는데 어셈블러와 같은 저급언어를 사용할 때 시간제약의 처리방법 및 Ada와 Modula와 같은 언어를 사용할 때 시간제약의 처리방법이 제안되었으며 시간제약을 직접 처리하는 언어들도 출현하였다. 특히 ESTEREL은 모든 이벤트가 동기화 되어 시간을 갖지 않고 즉각 실행하도록 하는 실시간 언어이다[9].

B. P. Douglass[3, 4]는 실시간 UML을 제안하였는데 H. Gomaa[10, 11]가 제시한 실시간 시스템의 시간 제약 조건과 이벤트의 반응시간을 고려하여 기존의 UML에 시간개념을 포함시켰는데 이벤트의 반응시간은 요구사항 획득 시기에 작성하도록 하고 이벤트간 시간 부여는 순서 다이어그램 작성시기에 부여하도록 하였다. 그리고 도착시간, 준비시간, 실행시간, 완료시간 등에 관한 시간제약을 고려하여 스케줄링을 하도록 하였다. 그러나 OCL을 사용한 정밀한 모델링은 간과하고 있다.

이에 따라, 본 논문에서는 D'souza와 Willis가 제안한 Catalysis와 같이 이벤트에 OCL을 사용하여 트랜잭션을 표기하는 방식으로 실시간 시스템 모델링에 적합하도록 OCL을 확장하였다. 특히 이벤트 위주로 동기화시켜서, 구현할 때 시스템 상태의 변화에 따라 동기화 관계에 의해서 시간제약 조건을 처리하도록 하였다. 효용성의 검증으로서 권선기 시뮬레이터에 응용하였다.

4. 실시간 OCL

실시간 OCL은 실시간 컴퓨팅 시스템을 정밀하게 명세화하기 위해서 우리가 OCL을 확장한 것이다. 모델링에서 OCL이 가장 중요하게 사용되는 부분은 활동 다이어그램의 표현에 있다. 이때 표현 방법은 메시지에 대한 전제조건(Precondition)과 결과조건(Postcondition)을 사용하여 객체의 상태를 표현하는 것이다. 다음은 기존의 OCL 표현방법이다.

<u>Typename::operationName(parameter1: type1, ...): ReturenType</u>

<u>pre</u>: parameter1>...

<u>post</u>: result= ...

실시간 시스템에서 각각의 객체의 상태가 갈라지고(forking) 합쳐지는(joining) 상태를 표현하기 위해서 우리는 동기(Synchronous) 및 비동기(Asynchronous) 메시지를 구분하는 방법으로 OCL 표현을 확장하였다.

4.1 메시지 동기화

실시간 UML은 객체간 통신의 기본 단위로서 메시지를 정의하고 있다. 메시지가 갖는 중요한 속성은 자료내용, 도착패턴, 동기화 패턴이다. 이벤트는 메시지에 의해서 일어난 시공간 위치를 가진 중요한 발생을 명세화한 것이다. 이벤트를 클래스로 표시하면, 이벤트 클래스들은 선택적으로 자료를 포함한다. UML에서는 이벤트를 신호라고 부르고 있으며, 이벤트 클래스는 스테레오타입 <<signal>>로 표현된다. 그림 1은 메시지 클래스 메타 모델을 일반화/특수화 관계를 사용하여 메시지를 도착패턴과 동시성 패턴으로 구분한 다이어그램이다[3].

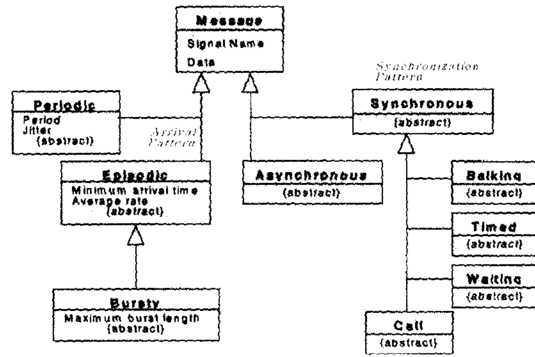


그림 1 UML 메시지 클래스 모델

메시지 도착 패턴은 메시지의 시간 행위를 기술한다. 도착 패턴의 정의는 예정 가능 분석과 제한 시간 분석에 결정적인 요인이 된다. 메시지 도착 패턴은 일시도착과 주기도착으로 분류한다. 동기화 패턴은 두 객체가 만나서 메시지를 교환하는 동안 발생하는 특징을 정의한다. 본 논문에서 우리는 동기화 패턴에 관심이 있다.

동기화 패턴은 두 객체가 만나서 메시지를 교환하는 동안 발생하는 특징을 정의한다. 동기화 패턴은 호출, 비동기, 대기, "balking", 타임아웃, 동기로 분류할 수 있다. 호출은 함수나 메소드 호출을 하는 동안 제어의 산출을 모델화 한다. 대기는 메시지 처리가 완료될 때까지 무한정 기다린다. "balking"은 상대 객체가 메시지를 받을 준비가 되어 있지 않으면 메시지를 전하지 않는다. 타임아웃은 정해진 시간 동안 상대 객체가 메시지를 받지 않으면 발송 객체는 메시지를 버린다. 비동기는 상대 객체가 메시지를 받을 준비가 되었는지 확인하지 않고 상대 객체로 메시지를 보내고 곧바로 프로세스를 계속한다.

• 비동기(Asynchronous) 메시지: 자동응답 전화기와 같이 송신자가 메시지를 보낼 때 수신자가 메시지를 받을 준비가 되어있지 않은 경우로서 "m" 객체가 수신자인 "r"에게 "aMessage()"를 보낼 때 다음과 같이 OCL을 확장한다.

<u>Asyn</u>:: aMessage(m.sender, r.receiver)

<u>pre</u>: r.someState

<u>post</u>: ...

그러나 수신자가 결정되어 있지 않으면

<u>Asyn</u>:: aMessage(m.sender)

<u>pre</u>: ...

<u>post</u>: ...

와 같이 정의한다.

• 동기(Synchronous) 메시지: 송신자와 수신자가 메시지를 주고받을 준비가 되어있을 때 메시지를 보내는 경우

Syn::messageName(m.sender, r.receiver):aState
pre: r.someState
post: ...

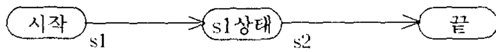
로서 OCL로 표현하자. "receiver"는 준비된 상태에서 메시지를 받고 리턴 값을 "aState"로 표현한 것이다.

4.2 순차실행

이벤트가 발생하면 각각의 메시지가 순차적으로 실행되는 것을 OCL로 표현하기 위해서

a.Message1(); b.Message2()

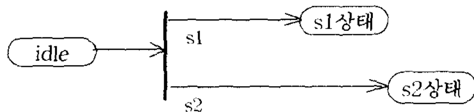
를 사용하자. 이것은 "a.Message1()"가 실행된 다음 "b.Message2()"가 실행됨을 의미한다. 즉, [[s1; s2]]는 아래 그림과 같이 표시된다. "[[ ]]"은 순차표현의 시작과 끝을 나타낸다.



4.3 비동기 병렬실행

비동기 메시지가 동시에 여러 개의 행위를 실행하게 하지만 종료하는 시간이 각 행위의 특성에 따라 다른 경우 "||"를 사용하여 OCL로 표현한다.

예를 들어 갈라지는(Forking) 상태 다이어그램을 설명한다.

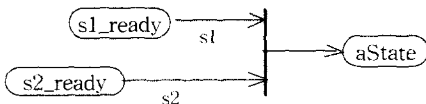


이것을 우리가 확장한 OCL로 표현하면 다음과 같다. 여기서 "aMessage"는 구현단계에서 그대로 사용하는 메시지 이름이어야 한다.

Asyn::aMessage(m.sender, (a,b).receiver)
pre: a.idle || b.idle
post: s1 || s2

4.4 동기 병렬 실행

동기 메시지는 여러 개의 행위를 동시에 실행하고, 동시에 한 곳에서 합쳐지도록(Joining) 하는 메시지이다.
예를 들어 a와 b 두 객체가 "aMessage"라는 메시지를 받을 준비가 된 상태에서 "aState"로 합쳐지는 상태다이어그램을 예로 들어본다.



이것을 다음의 OCL로 표현하면 다음과 같다.

Syn::aMessage(m.sender, (a,b).receiver):aState
pre: s1_ready || s2_ready
post: s1 || s2

여기서 메시지 "aMessage"에 의한 상태전이가 완료될 때까지 기다리기 위해서는 OCL 표현으로서 **waitOnTransition(aMessage)**를 사용하고 "aMessage"를 호출하여 상태전이가 이루어질 때까지 기다리기 위해서는 OCL 표현으로서 **waitOnCall(aMessage)**를 사용한다. 특히 일정시간 예를 들어 "TTime" 시간 동안 행위를 지연시키기 위해서는 OCL 표현으로서 **wait(tTime)** 을 사용한다.

4.5 반복문

OCL에서 반복문 기능은 iterate로서 다음과 같이 표현하고 있다[2,12].

```
collection->iterate(element : Type1 ;
        result : Type2 = <expression>
    | <expression-with-element-and-result>)
```

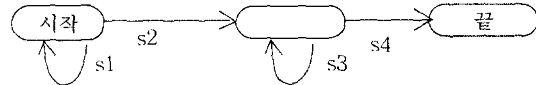
이것을 의사코드(pseudocode)로 표현하면 다음과 같다.

```
result = <expression>;
while( collection.notEmpty() ) do
  element = collection.nextElement();
  result = <expression-with-element-and-result>;
endwhile
return result;
```

그러나 반복적으로 메시지를 수행을 표현할 때에는 "*"나 "+"를 사용하는 것이 간편하다.

s* -- s의 임의 수만큼 반복
s+ -- s의 한번이상 반복

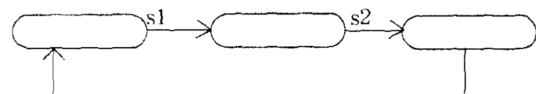
예를 들어 "[[s1*; s2; s3+; s4]]"은 다음의 상태 다이어그램을 표현한다.



특히 지정한 곳으로 제어를 보내어 반복을 나타낼 경우에는

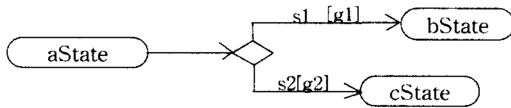
Send(반복할 메시지 이름)

와 같이 반복위치를 지정해준다. 예를 들어 아래의 상태 다이어그램에 대하여 OCL로 표현하면 **[[s1; s2; Send(s1)]]**와 같다.



4.6 조건문 실행

경계조건에 따라 활동 다이어그램의 분기(Branch)를 나타내기 위해서 "if"문을 사용한다.



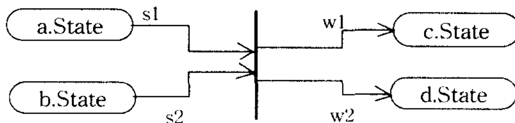
위의 활동 다이어그램을 OCL로 표현하면,

Asyn::aMessage(m.sender, (a,b).receiver)
pre: self.aState
post: if [g1] then s1;
else if [g2] then s2

와 같다. 여기서 "if - else" 문은 기존의 OCL에서 정의된 표현이다.

4.7 복합 실행

동기와 비동기 메시지에 의해서 합쳐지고 갈라지는 활동 다이어그램이 복합되어 있는 활동 다이어그램을 예로 들면 다음과 같다.



이러한 상태 다이어그램을 OCL로 표현하면 다음과 같다.

Syn::uMessage(m.sender, (a,b).receiver): tState
pre: a.State || b.State
post: s1 || s2 ;
waitOnTransition(uMessage);
Syn::vMessage(m.sender, (a,b).receiver)
pre: a.tState || b.tState
post: w1 || w2;

우리는 OCL 표현에 의해서 위의 상태 다이어그램을 자세히 분석할 수 있다. 즉, a와 b 두 객체가 동기화 메시지 "uMessage"에 의해서 s1과 s2를 동시에 실행하고 합쳐져서 "tState"가 되기를 기다렸다가 이어서 동기화 메시지 "vMessage"에 의해서 w1과 w2를 동시에 실행하며 갈라지게 된다.

4.8 실시간 시간제약 처리방법

이벤트 도착 방법에 따라 그림 1과 같이 이벤트를 분류하고 이벤트의 반응 시간과 지연 시간은 메시지 동기화 OCL 구문과 함께 wait(Time)을 사용하고, 이벤트의 완료시간도 메시지 동기화 OCL 구문과 함께 waitOnTransition(MessageName)과 waitOnCall(Messagename)에 의해서 처리한다. 이벤트의 준비시간과 실행시간은 이벤트의 동기화 시점의 순서로서 처리한다. 이벤트 사이의 시간 처리는 순서다이어그램(Sequence Diagram)에서 처리한다. 이에 따라 태스크(Task)의 시간제약(도착시간, 준비시간, 실행시간, 완료시간)에 따른 실시간 처리 방식은 앞에서 정의한 메시지 동기화의 관계에 따라 스케줄러에 의해서 이루어져야 한다. 즉, 메시지 동기화 순서와 시간처리에 따라 스케줄러가 시간제약을 처리한다. 이러한 처리 방법은 주로 PCL(Process control language)에 의해서 처리되며, 한 예로서 실시간 언어인 ESTEREL은 모든 이벤트를 동기화 시켜서 시간 제약을 처리하고 있다[9].

5. 권선기 시뮬레이터 모델링

권선기 시스템은 코일을 원하는 모양으로 감아주는 기계로, 제어기부분, 장치부분, 사용자 인터페이스부분의 구조를 갖는다. UML과 OCL을 사용하여 모델링 하기 위해서 권선기의 각 부분(Unit)들을 컴포넌트화 하여, 인터페이스(입출력 및 작동 순서 결정), 상태/동작 컴포넌트, 커널 (Kernel-하드웨어와 소프트웨어 인터페이스), 스케줄러(scheduler)의 형태를 갖도록 설계를 한다.

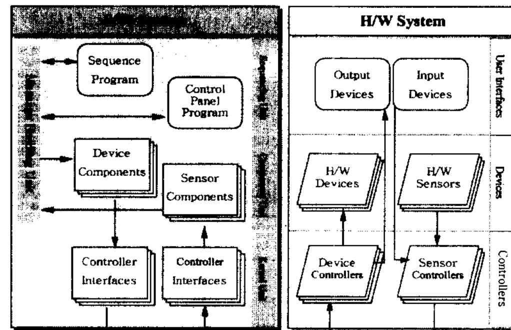


그림 2 권선기 시뮬레이터 구조

그림 2는 권선기 시뮬레이터의 구조를 나타낸 것으로, 하드웨어 시스템과 소프트웨어 시스템을 직접 연결해 주는 커널 단위(Kernel Unit), 기구의 상태를 나타내거나 기구를 작동시키는 컴포넌트 단위(Component Unit), 작동 순서를 결정하거나 입력을 및 출력을 담당하는 순차 단위(Sequencing Unit)와 시스템 내의 이벤트(메시지)를 관리하는 메시지 처리 단위(Message Handling Unit)로 구성된다.

권선기의 각 부분을 컴포넌트화 함으로써 개발 및 유지 보수상의 이점, 확장의 용이성을 얻을 수 있다. 또한 실시간 UML과 OCL을 사용하여 시간 처리 기능을 통해 실시간 시스템의 시간 제약 요구사항을 정밀하게 모델링 할 수 있다.

본 논문에서는 실시간 OCL을 다루고 있음으로 클래스 다이어그램과 순서(Sequence) 다이어그램은 생략하고 실시간 상태 다이어그램을 중심으로 전개한다.

5.1 활동 다이어그램

활동 다이어그램은 행위와 객체 상태 변화에 대한 행위의 결과를 나타낸다. 활동 다이어그램에서 상태는 모두 행위상태이고 상태전이는 모두 이전의 원시상태에서부터 행위의 수행에 의해서 이루어진다. 전체 활동 다이어그램들은 클래스에 첨가되거나 사용사례 및 연산의 구현에 첨가된다. 활동 다이어그램은 작업흐름(Work-flow)을 모델링하고 많은 병렬처리를 가진 행위를 묘사하는데 유용하다.

활동 다이어그램으로 권선기 시스템의 행위를 표현하여 보자. 그림 3은 권선기 부품 조작과 관련한 상태 다이어그램의 한 일면을 나타낸 것이다. ① 초기화 메시지 "MInitialize"가 발생을 하면 3개의 유닛을 동시에 구동 준비위치(Ready Position)에 위치시킨다. ② 세 개의 유닛이 이동을 모두 마치면, ③ 일련의 제작 작업을 마치고, ④ 정해진 작업이 끝나면 초기 작업으로 옮겨서 이 과정을 반복한다. 이때 에러가 발생하게 되면 에러 처리를 한다. 5절에서는 정밀한 시스템 설계를 위하여 각각의 상태를 우리가 정의한 OCL에 의해서 표현한다.

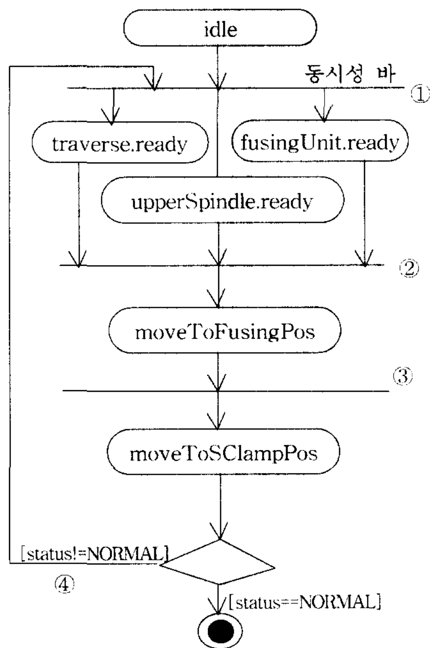


그림 3 권선기 활동 다이어그램

5.2 비동기 병렬실행 부분의 OCL 표현

그림 4는 권선기 모델링에 있어서 비동기 병렬실행 부분을 OCL로 표현한 것이다.

-- **Forking**

<u>Asy</u>::MInitialise(g_msgHandler.sender,
(traverse, upperSpindle, fusingUnit).receiver)
<u>pre</u>: traverse.idle || upperSpindle.idle ||
fusingUnit.idle
<u>post</u>: traverse.MoveToReadyPos() ||
upperSpindle.MoveToReadyPos() ||
fusingUnit.MoveToReadyPos()

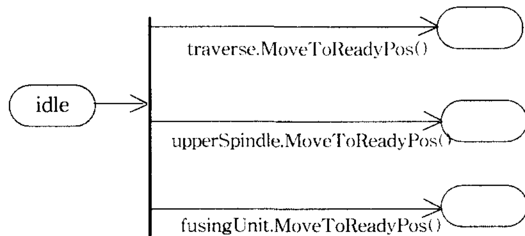


그림 4 비동기 병렬 실행 OCL 표현

"traverse, upperSpindle, fusingUnit"는 쉬고있는 상태에서 비동기 병렬 실행 메시지 "MInitialize"에 의해 행위 "MoveToReadyPos()"가 동시에 실행되며, 종료되는 시점은 각 기구에 따라 틀리다.

5.3 동기병렬 실행 부분의 OCL 표현

다음 그림5는 권선기 모델링에 있어서 동기 병렬 실행의 OCL 표현을 나타낸다.

동기 메시지 "MTraverseMoveToFusingPos"가 발생하면 "traverse, upperSpindle, fusingUnit" 등 세 기구는 동시에 "Mtraverse.MoveToFusingPos()"를 실행하게 되어 동시에 합쳐지는 위치에 놓이게 된다. 이어서 동기 메시지 "MTraverseMoveToSClampPos"가 발생하면 세 기구는 동시에 "Mtraverse.MoveToSClampPos()"를 실행하여 합쳐지게 된다.

5.4 경계조건에 따른 종료와 반복 부분의 OCL 표현

그림 6은 권선기 활동 다이어그램에서 경계조건에 의한 반복과 종료를 나타내는 OCL 표현 나타낸다.

```
--LocationOfJoin
waitOnTransition(MInitialise);
 Syn:: MTraverseMovedToFusingPos(
   g_msgHandler.sender,
   (traverse, upperSpindle, fusingUnit).receiver):
   statusOfMovedToFusingPos
 pre: traverse.ready || upperSpindle.ready ||
      fusingUnit.ready
 post:[[traverse.MtraverseMoveToSClampPos||
      upperSpindle.MtraverseMoveToSClampPos||
       fusingUnit.MtraverseMoveToSClampPos ]]
--Joining
waitOnTransition(MTraverseMovedToFusingPos);
Syn:: MTraverseMovedToSClamPos(
  g_msgHandler.sender,
  (traverse, upperSpindle, fusingUnit).receiver):
 ststus
pre: traverse.MovedToFusingPos ||
     upperSpindle.MovedToFusingPos ||
     fusingUnit.MovedToFusingPos
post: [[traverse.MovedToSClampPos()||
       upperSpindle.MovedToSClampPos()||
       fusingUnit.MovedToSClampPos()]]
```

그림 5 동기 병렬 실행의 OCL 표현

```
-- Iteration
waitOnTransition(MTraverseMovedToSClamPos);
action iterate(state)
pre: status=MTraverseMoveToSClampPos()
post: if ( status==NORMAL )
      then stopMessage();
      else errorMessage();
        Send(MInitialise)
```

그림 6 조건문과 반복문을 나타내는 OCL

그림 6은 "traverse"의 행위 "MoveToSClampPos()"가 실행된 후 결과 값이 "status"에서 받게 된 경우 "status"의 값이 "NORMAL"이면 "stopMessage()"를, 그렇지 않으면 "errorMessage()"를 발생시키고 권선기 세 기구의 행위를 반복시키기 위해서 "Send (MInitialize)"를 발생시킨다.

5.5 활동 다이어그램의 OCL 표현

그림 7은 권선기 상태 다이어그램(그림 3)에 대한 OCL 표현을 종합한 것이다.

6. 결 론

실시간 어플리케이션은 비 실시간 어플리케이션에 비해 보다 정밀한 시스템 사양을 요구한다. 그러나 시스템 사양이 이를 충분히 제공해 줄 수 없다면, 실시간 어플리케이션의 결과는 만족스럽지 못하게 될 것이다. 정밀한 실시간 시스템을 모델링 하기 위해서는 UML과 더불어 OCL을 함께 사용하는 것이 필수적이다. 이에 따라 실시간 시스템을 모델링 하기 위해서 기존의 OCL은 확장되어져야 한다.

실시간 모델링에서 활동 다이어그램의 표현에 OCL 표현을 부가적으로 표기하는 것은 객체의 상태를 자연어가 아닌 형식언어 성격을 가진 언어로 정밀한 표기를 하여 애매 모호성을 없애주고 있다.

```
-- Forking
Asy::MInitialise(g_msgHandler.sender,
 (traverse, upperSpindle, fusingUnit).receiver)
pre: traverse.idle || upperSpindle.idle ||
     fusingUnit.idle
post: traverse.MoveToReadyPos()||
      upperSpindle.MoveToReadyPos()||
      fusingUnit.MoveToReadyPos() --LocationOfJoin
waitOnTransition(MInitialise);
Syn:: MTraverseMovedToFusingPos(
  g_msgHandler.sender,
  (traverse, upperSpindle, fusingUnit).receiver):
  statusOfMovedToFusingPos
pre: traverse.ready || upperSpindle.ready ||
     fusingUnit.ready
post: [[ traverse.MtraverseMoveToSClampPos||
         upperSpindle.MtraverseMoveToSClampPos||
          fusingUnit.MtraverseMoveToSClampPos ]]
--Joining
waitOnTransition(MTraverseMovedToFusingPos);
Syn:: MTraverseMovedToSClamPos(
  g_msgHandler.sender,
  (traverse, upperSpindle, fusingUnit).receiver):
 ststus
pre: traverse.MovedToFusingPos ||
     upperSpindle.MovedToFusingPos ||
     fusingUnit.MovedToFusingPos
post: [[traverse.MovedToSClampPos()||
        upperSpindle.MovedToSClampPos()||
        fusingUnit.MovedToSClampPos()]]
--Iteration
waitOnTransition(MTraverseMovedToSClamPos);
action iterate(state)
pre: status=MTraverseMoveToSClampPos()
post: if ( status==NORMAL )
      then stopMessage();
      else errorMessage();
        Send(MInitialise)
```

그림 7 권선기 상태 다이어그램의 OCL 표현

우리가 제시한 실시간 OCL 확장은 동기 메시지와 비동기 메시지를 구분하고 여기에 전제조건과 결과조건을 기술하여 객체의 상태를 표현하며 병렬 실행을 나타내는 기호를 주어 실시간 상태 다이어그램에 의한 정밀한 표현을 하였다. 이러한 확장은 시스템의 제어가 윈도우 환경과 유사한 이벤트 위주 방식을 채택하고 있어서 산업용 실시간 언어로 쉽게 변환이 가능하다. 이것은 주로 활동 다이어그램으로 잘 표현되는 시스템을 구축하는데 효과적이다.

우리가 제공하는 메커니즘은 프로세스가 시스템 상태에 대한 정보를 공유하며 프로세스 각각에 대한 정보는 공유하지 않고 시스템의 상태를 변화시키도록 하고 있다. 그리고 실시간 이벤트 처리방식은 이벤트가 일어나는 시간과 반응시간 사이의 지연을 처리하고 서로 다른 프로세스가 똑같은 이벤트를 기다릴 수 있으며 한 프로세스가 서로 다른 이벤트를 병렬로 처리할 수 있다. 따라서 스케줄러는 각 태스크(Task)의 시간제약 조건에 따라 메시지 동기화의 순서에 의해서 스케줄링 하여야 한다.

특히 우리가 확장한 OCL을 권선기 실시간 시스템 개발에 적용하여 실시간 모델링과 실시간 언어 사이의 중간 역할을 하도록 하였다. 부록으로 확장 OCL 구문과 실시간 언어에 의한 Nrcl 프로그램 실예를 들었다.

참 고 문 헌

[1] R. Gerber and S. S. Hong, "Compiling Real-Time Programs with Timing Constraint Refinement and Structural Code Motion," Transactions on Software Engineering, IEEE, 1995.

[2] B. Hoogeboom and W. A. Halang, "The Concept of Time in the Specification of Real Time Systems," Real-Time Systems, IEEE Computer Soc. Press, pp. 26-28, 1992.

[3] B. P. Douglass, Real-Time UML, Addison - Wesley, 1998.

[4] B. P. Douglass, Doing Hard Time, Addison - Wesley, 1999.

[5] UML Proposal to the Object Constraint Language, http://www.rational.com/uml, 1997.

[6] J. B. Warwer and A.G. Kleppe, The object constraint language, Addison-Wesley, 1999.

[7] D. F. D'souza and A. C. Willis, Object, Constraint, and Frameworks with UML, Addison-Wesley, 1998.

[8] Y. Ishikawn, H. Tokuda, and C. W. Mercer, Objected-oriented Real Time language Design: Construct for Timing Constraints, Canegie-Mellon Techical Report CMU-CS-SO-111, 1990.

[9] K. J. Lin, Expressing and Maintaining Timing Constraints in FLEX, Proc. IEEE Real Time Systems Symposium, pp. 96-105, 1988.

[10] H. Gomma, Software design for distributed real time applications, Journal of Systems and Software, pp. 81-94, 1989.

[11] H. Gomaa, Software design method for concurrent and real-time systems, Addison - Wesley, 1993.

[12] S. S. Hong and R. Gerber, "Slicing Real-Time Programs for Enhanced Schedulability," Transactions on Programming Languages and Systems, ACM, Volume 8, pp.526, 1997.

[13] B. Furht et al., Real-time UNIX systems - design and application guide, Kluwer, 1991.

[14] N. Nissanke, Realtime Systems, Prentice Hall, 1997.

[15] B. Selic, G. Gullekson, and P. T. Ward, Real Time Objected Modeling, John Willey & Sons, 1994.

Appendix A: Extended OCL Syntax

```
expressions:= expressions expression
              | expression
expression:= logical_expression
              | control_expression
              | synchronization_expression
              | asynchronization_expression
logical_expression:= -- OCL과 동일control_expression:=
              if_expression
              | concurrency_expression
              | iterate_expression
if_expression:=         -- OCL과  동일
concurrency_expression:= " || " | "[[ || ]] "
iterate_expression:=  iterate | "*" | "+"|
                    Send(message_name)
synchronization_expression:=
      Syn::MessageName (object.sender, object.receiver):
          return value
           | waitOnTrasition(MessageName)
           | waitOnCall(MessageName)
           | wait(Time)
asynchronization_expression:=
     Asy::MessageName (object.sender, object.receiver)
          | MessageName(object.sender)
```

Appendix B: 권선기 Nrcl 프로그램

```
1  PROGRAM_BEGIN (NDV7)
2
3  ON (MInitialize) BEGIN
4      traverse.MoveToReadyPos();
5      upperSpindle.MoveToReadyPos();
6      fusingUnitMoveToReadyPos();
7  END
8
9  ON (MTraverseIsReady) BEGIN
10     SYNC (MTraverseMoveToSClampPos);
11 ON (MUpperSpindleIsReady)
12     SYNC (MTraverseMoveToSClampPos);
13 ON (MFusingUnitIsReady)
14     SYNC (MTraverseMoveToSClampPos);
15
16 ON (MTraverseMoveToSClampPos)
17     traverse.MoveToSClampPos();
18
19 ON (MTraverseIsAtSClampPos)
20     SEND (MInitialize);
21
22 ON (MTraverseMoveToSReadyPosError)
23 errorHandler.TraverseMoveToReadyPosError();
24
25 PROGRAM_END
```

최 성 운

1985년 한국외국어대학교(상학사). 1988년 미국 오레곤주립대학교 컴퓨터공학과(석사). 1992년 미국 오레곤주립대학교 컴퓨터공학과(박사). 1993년 ~ 현재 명지대학교 컴퓨터학부 부교수, OMG KSIG 회장, 한국정보컨설팅(KIC) 기술자문. 관심분야는 컴포넌트 프레임워크, 객체지향 소프트웨어공학

이 영 환

1982년 충남대학교 (학사). 1984년 충남대학교 수학과(석사). 1988년 충남대학교 수학과(박사). 2001년 명지대학교 컴퓨터공학과 박사과정 수료. 1988년 ~ 현재 대전대학교 기초과학부 교수. 관심분야는 실시간 시스템, 객체지향 소프트웨어공학, 형식언어, 정보보호