MSC로 작성된 통신 프로토콜 명세의 의미론 연구

방기석*○류광열* 오정기** 최진영*
*고려대학교 컴퓨터학과 정형기법연구실
**극동 정보통신 대학교
{kbang, kyryu, choi}@formal.korea.ac.kr

Study on the Semantics of Communication Protocols in Message Squence Charts

Ki-Seok Bang*, Kwang-Yul Ryu*, Jung-Ki Oh**, Jin-Young Choi*
Dept. of Computer Science & Engineering, Korea University

요 약

메세지 순서도(Message Sequence Chart,MSC)는 ITU-T 에서 국제적인 표준으로 제안되어 주로 전기 통신 교환 시스템과 같은 실시간 시스템을 위한 통신 행위에 대한 개괄적인 표현 방법으로서 널리 사용되어지고 있으며 요구 명세, 인터페이스 명세, 시뮬레이션 및 검정을 위해 사용되어지고 있다. MSC의 장점이라면 표현된 시스템의 행위를 직관적으로 이해할 수 있게 해주는 그래픽 표현을 제공하는 것이다. 의미론 입장에서 보면 MSC는 프로세스 대수 ACP의 변형인 PA_ϵ에 의해 의미를 부여받고 있긴 하지만 이해하기가 난해한 것이 사실이다. 본 논문에서는 MSC의 동작적 의미를 분석하며 ACSR로 변환하여 그 의미를 보다 쉽게 파악하는 방법론에 대해 다룬다.

제 1 절 서론

소프트웨어나 시스템을 설계하고 구현하는데 있어서 오류라는 것은 배제할 수 없는 존재가 되어 버렸다. 이러한 오류를 설계 단계에서 찾아내는 것은 시간과 비용 면에 있어서 대단히 중요해 지고 있으며 이러한 오류의 조기 발견을 위해서 다양한 방법론이 제기 되고 있다. 많은 시뮬레이션 및 베타 테스트 등을 이용하여 그러한 오류들을 찾아내고 있긴 하지만 그러한 접근 방법 역시 완벽하지는 못하다. 정형기법은 이러한 오류를 찾아내기 위하여 논리적인 접근방법을 택하고 있다. 현재 다양한 정형기법이 제시되고 있고 실제로 사용되고 있는데 본 논문에서 소개하고 있는 메세지 순서도(Message Sequence Chart,이하 MSC)[4]도 이러한 맥락으로 파악할 수 있겠다. MSC는 ITU-T에서 정의된 메세지 표현 방법으로서 통신 시스템 정의 및 명세등에 사용되고 있다. MSC은 주로 여러 개의 프로세스 사이의 메세지 교환에 대해서 명세를 하게 되며 이는 병렬 시스템, 통신시스템의 명세 상의 오류를 찾아내는데 유용하다. MSC의 의미론은 프로세스 대수 ACP의 변형인 PA_ϵ에 의해서 정의되어 있으며 이해하기가 난해하다[6].

ACSR(Algebra of Communicating Shared Resources)[1]은 프로세스 대수의 일종으로서 논리적인 기반을 CCS(Calculus of Communicating Systems)[5]에 두고 있다. ACSR에는 시간, 자원, 우선순위, 동시성 등을 지원하며 시스템의 모델을 자원을 사용하는 통신 프로세스들의 집합체로 파악하고 있으며 ACSR에서 제공하는 여러가지 기능들은 MSC로 명세한 시스템을 표현하기에 충분하다. 이해하기 힘든 PA_ϵ을 이해하기 쉽게 하기 위해 ACSR로 변환하는 것이 본 논문의 목적이다. ACSR과 PA_ϵ은 모두 레이블된 전이 시스템(labeled transition system)으로 표현되어 있으며 의미론적으로 동등하기 때문에 MSC를 ACSR로 변환하여도 그 의미를 상실하지 않게 된다. 또한 VERSA[2]라는 정형검증도구를 이용하여 equivalencs check를 할 수 있기 때문에 MSC의 동작적 의미를 보다 명확히 하고 정확성을 검증함으로써 보다 안정한 동작의 명세가 가능하다.

본 논문의 구성은 다음과 같다. 2장에서는 MSC에 대한 소개 및 문법적 의미등을 살펴보고, 3장에서는 MSC를 ACSR로 변환하는 방법에 대해서 살펴 보겠다. 마지막으로 4장에서 결론을 맺겠다.

제 2 절 메세지 순서도

2.1 소개

MSC는 시스템 구성요소(component)들 사이의 상호작용을 기술하고 명세를 위한 그래픽 표현과 텍스트 표현을 동시에 지원하는 언어이다. MSC가 이용되어지는 응용 프로그램의 주요 영역은 실시간 시스템-특히 전기 통신 시스템(telecommunication) 내의 교환 시스템 -의 통신 행동양식(communication behaviour)의 개괄적인 명세를 하는 것이다. MSC는 또한 요구 명세(requirement specification), 인터페이스 명세(interface specification), 시뮬레이션(simulation), 검정(validation), 테스트 케이스 명세 및 실시간 시스템의 문서화 등에 사용될 수 있다.[3, 4]

MSC는 그래픽 표현과 텍스트 표현 모두를 가지고 있다. 그래픽으로 표현되어지는 것이 가장 좋은 방법이긴 하지만

정형적인 의미론의 정의 등과 관련된 부분에서는 텍스트 표현이 선호되고 있다[6]. MSC의 핵심 언어를 보통 기본 메세지 순서도(Basic Message Sequence Charts, 이하 BMSC)라고 한다. BMSC는 상호 메세시 교환과 로컬 액션에 중점을 두고 있다.

2.2 기본 메세지 순서도

BMSC는 인스턴스들의 유한 집합이라고 말할 수 있다. 하나의 인스턴스는 추상적인 개체로서 메세지의 출력, 메세지의 입력, 로컬 액션 등을 포함하게 된다. 인스턴스는 수직선으로 표시되며, 시간은 위에서 아래로 흐르게 된다. 인스턴스 안에 명시된 이벤트들은 시간적으로 순서화되어 있게 된다. 하나의 인스턴스 안에서는 두 개의 이벤트가 동시에 발생할 수도 없다. 인스턴스는 이름을 가지게 되며, 이름은 수직축의 윗부분에 표시된다.

메세지는 메세지 출력(message output)과 메세지 입력(message input)으로 나누어지게 된다. 인스턴스에서 주위환경(environment)로 보내지는 메세지는 송신 인스턴스에서 시작해서 MSC의 외부로 향하는 화살표로 표시된다. 주위환경에서 인스턴스에게 보내지는 메세지는 외부에서 시작하여 수신 인스턴스에서 끝나는 화살표로 표시되게 된다. 메세지는 인자 리스트와 함께 표시될 수 있으며 인자는 메세지의 이름 다음에 오는 브레킷안에 표시된다.

MSC의 응용 프로그램들은 주로 그래픽 표현에 중점을 두고 있으나 컴퓨터 응용 프로그램 사이의 상호 전환 등을 위해 텍스트 표현도 제공하고 있다. BMSC의 텍스트 표현에서는 주로 인스턴스를 중심으로 표현된다. 다시 말하면 BMSC는 모든 인스턴스의 행동양식을 정의함으로써 명세 된다는 것이다. 메세지의 출력은 `out m1 to i2;`, 입력은 `in m1 from i1;`로 표현된다.

2.3 비순서영역

각각의 인스턴스 안에 명세된 이벤트들은 전순서화(totally ordered)되어 있다. 인스턴스 안에서 순서가 지정되어 있지 않는 것을 가능하게 하기 위해서 비순서영역(coregion)이 도입되었다. 비순서영역은 인스턴스 축에서 줄선으로 표시되며, 비순서영역내에서 발생하는 이벤트에 대해서는 순서가 정해지지 않은 것으로 파악되게 된다. 또한 비순서영역내에서는 오직 메세지 이벤트만이 정의 가능하다.

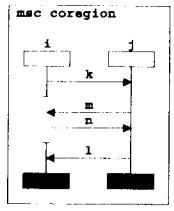

그림 1: 비순서영역을 가진 메세지 순서도

제 3 절 ACSR을 이용한 MSC의 표현

본 장에서는 ACSR을 이용하여 MSC를 표현하는 방법에 대해서 살펴보겠다.

3.1 메세지의 전달

메세지의 전달을 전환하는 기본적인 방법은 새로운 버퍼 프로세스를 생성하는 것이다. 이는 ACSR의 메세지의 전달이 바로 소모되어 버리기 때문에 버퍼를 두어서 바로 소모하지 않도록 하게 한다.

그림 2은 기본적인 메세지의 전달을 MSC로 표현한 것으로 인스턴스 i가 메세지 m을 j에게 보내는 것을 명세한 것이다.

그림 3는 그림 2의 ACSR 표현을 나타내고 있다. ACSR로 표

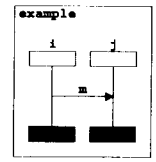

그림 2: (예제 1) 기본적인 MSC

현하는데 있어서의 차이점은 MSC에서 메세지의 흐름을 표시하는 화살표에는 버퍼의 개념이 내포되어 있지만 ACSR에는 버퍼의 개념이 없다는 것으로 이러한 차이를 없애기 위해 버퍼 m_b를 새로이 구성하게 된다. 또한 새로이 추가된 버퍼 m_b에 대해서 입력과 출력에 해당하는 m_b_in과 m_b_out은 외부에서 관찰되지 않기 때문에 은닉을 시키게 된다.

```
i = m!. m_b_in!.NIL ;
j = m_b_out?.m?.NIL;
m_b = m_b_in?.m_b_out!.m_b;
example1 = ((i||j)||m_b)
        \{m_b_in,m_b_out}
```

그림 3: (예제 1) MSC와 ACSR 코드

3.2 인스턴스의 생성 및 종료의 변환

인스턴스의 생성에 관한 그래픽 표현은 그림 4와 같다.

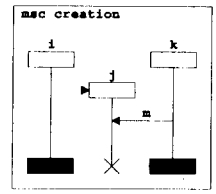

그림 4: 프로세스 생성 및 종료를 표현한 메세지 순서도

그러면 변환된 ACSR코드를 살펴보기로 하자. 우선 기본적인 방법을 알아보겠다. MSC에서는 프로세스의 동적 생성이 가능하지만 ACSR에서는 프로세스의 동적 생성이 불가능하다. 이를 해결하기 위해서 본 논문에서는 다음과 같은 방법을 제안한다. 우선 동적으로 생성될 프로세스도 하나의 프로세스로 파악한다. 동적으로 생성되는 프로세스는 활성화 메세지를 받기전까지는 활성화가 되지 않는 것으로 한다. 동적 프로세스를 생성하는 프로세스에서는 MSC의 인스턴스 생성 이벤트인 create가 들어오게 되면 ACSR에서는 해당하는 프로세스에게 활성화 메세지를 보내게 된다. 이렇게 함으로써 MSC의 프로세스 생성을 ACSR에서 표현하게 된다. 동적 프

로세스의 생성에서는 시간의 흐름이 없기 때문에 ACSR에서도 이를 위해서 버퍼를 두지 않게 된다.

그림 5에서는 그림 4에 대해 ACSR로 변환한 것을 표현하였다. MSC에서는 동적으로 생성된 프로세스는 stop이라는 토큰으로써 해당 프로세스의 종료를 표시하게 된다. 이를 ACSR로 전환하면 NIL을 사용하게 된다. NIL은 ACSR에서 아무것도 하지 않는 프로세스의 종료 상태를 표시하게 된다. 그림 5에서 ACSR의 프로세스 j를 보면 NIL로 끝나는 것을 볼 수 있다.

```
i = create_j!.NIL;
j = create_j?.m_b_out?.m?.NIL;
k = m!.m_b_in!.NIL;
m_b = m_b_in?.m_b_out!.m_b;
creation = ((i||j||k)||m_b)
   \{m_b_in,m_b_out}
```

그림 5: 프로세스 생성 동작 MSC와 ACSR 코드

3.3 비순서영역의 변환

ACSR에서는 비결정적 선택을 지원하긴 하지만 MSC의 비순서영역과는 성격이 다르다. ACSR에서는 다음과 같은 방법으로 비결정적 선택을 지원한다.

$$P = Q + R$$

위에서 프로세스 P는 프로세스 Q 또는 R로의 전이가 가능하다. 이는 둘 중에 하나를 선택하는 개념이다. 만약 Q가 선택되면 R은 선택되어지지 않고 제어는 Q로 넘어가게 된다. 만약 R이 선택되면 마찬가지로 Q는 제어를 가질 수 없게 되는 것이다. MSC에서 비순서영역은 이벤트의 순서에 대한 선택이 없을 뿐이며 반드시 실행이 되게 된다. 즉 이벤트에 순서가 없다는 것이다.

```
i = k!.k_b_in!.i';
i' = ((m_b_out?.m?.co_end!.IDLE)
     || (n!.n_b_in!.co_end!.IDEL)
     || (co_end?.co_end?.end!.IDLE))
    .scope(end?,infty,NIL,i'',NIL);
i'' = l_b_out?.i;
j = k_b_out?.k?.m!.m_b_in!
    .n_b_out?.n?.l!.l_b_out!.k;
k_b = k_b_in?.k_b_out!.k_b;
l_b = l_b_in?.l_b_out!.l_b;
m_b = m_b_in?.m_b_out!.m_b;
n_b = n_b_in?.n_b_out!.n_b;
IDLE = {}:IDLE;
coregion = ((i||j)||k_b||l_b||m_b||n_b)
    \\{k_b_in,k_b_out,l_b_in,l_b_out,
       m_b_in,m_b_out,n_b_in,n_b_out};
```

그림 6: 비순서 영역 표현 MSC와 ACSR 코드

그림 6는 비순서영역의 MSC 표현을 ACSR로 변환한 것이다. 인스턴스 i는 인스턴스 j에게 메세지 k를 전송한 후 비순서영역 안으로 진입한다. 비순서영역 안에는 인스턴스 j에서 메세지 m을 받아들이는 이벤트와 인스턴스 j에게 메세지 n을 내보내는 이벤트가 존재한다. 그런 후 비순서영역을 빠져나가며 인스턴스 j로부터 메세지 l을 받아들이게 된다. 여기에서 비순서영역안에 있는 부분만 ACSR로 전환하면 아래와 같게 된다.

$$i = ((m\_b\_in?.m?.co\_end!.0^{\infty} \| n!.n\_b\_out!.co\_end!.0^{\infty})$$

$$\|(co\_end?.co\_end?.end!.0^{\infty}))\Delta_{\infty}^{end?}(NIL, ACSR(i'), NIL)$$

우선 인스턴스의 각각의 이벤트를 병렬로 수행되는 프로세스로 전환을 하게 된다. 위에서는 $m_b_in?.m?.co_end!.0^{\infty}$와 $n!.n_b_out!.co_end!.0^{\infty}$를 ||로 연결함으로써 병렬적으로 수행하게 하였다. 병렬적으로 수행하게 되면 두 개의 프로세스 중에는 순서가 존재하지 않기 때문에 MSC에서의 비순서영역과 같은 역할을 수행하게 된다. 하지만 둘 중에 하나라도 끝나면 다른 프로세스를 무시하고 다음 단계로 넘어가기 때문에 $(co_end?.co_end?.end!.0^{\infty})$에 같이 두개의 프로세스가 모두 종료되기 까지 기다리는 감시 프로세스를 함께 실행하게 된다. 즉 메세지 m을 받는 프로세스와 메세지 n을 내보내는 프로세스에서 각각 co_end라는 메세지를 내보내기 전까지 기다리게 함으로써 비순서영역을 가능하게 한다. 그림 6은 비순서영역의 텍스트 표현 및 ACSR로 변환한 것을 나타내고 있다.

제 4 절 결론

본 논문에서는 통신 시스템의 명세에 사용되어 지는 MSC의 이용분야 및 문법적 의미들에 대해서 살펴보았다. MSC는 다양한 통신 프로토콜의 명세에 사용될 수 있으며 인터페이스를 정의하는데도 사용될 수 있다는 것에 대해서도 살펴보았다. 하지만 MSC로 명세된 모델의 정형적인 검증을 위한 환경이 미비하고 MSC의 의미론을 명세하고 있는 PA_c이 이해하기 어렵다는 것도 알아보았다. 이러한 의미론을 CCS의 관점에서 바라보면 쉽게 이해될 수 있다는 것을 살펴보았으며 MSC를 이러한 CCS기반의 프로세스 알지브라인 ACSR로 변환을 위한 방법론을 제시하였다. 이를 위해 ACSR의 특성 및 동작적 의미들을 분석하여 MSC와 동일한 동작을 하는 명세를 구성할 수 있도록 하였다. ACSR로의 전환한 다양한 예를 살펴보았으며 MSC내에 존재하는 오류 ACSR로 명세된 모델에 대해 검증을 하는 versa를 이용하여 찾아낼 수 있다는 것을 보였다.

현재는 MSC를 ACSR형태로 변형이 가능하지만, 앞으로 중간적인 표현형태를 보다 일반화 시킴으로써 다른 프로세스 대수 및 모델 체킹 도구로의 전환을 가능하게 하는 것이 과제로 남아있다.

참고 서적

[1] J. Y. Choi, I. S. Lee and H. Ben-Abdallah. A process algebraic method for the specification and analysis of real-time systems. *Formal Methods for Real-Time Computing*, 1996.

[2] D. Clarke. VERSA: Verification, Execution and Rewrite System for ACSR. *Real-Time Group Report*, University of Pennsylvania, 1996.

[3] ITU-T. Message Sequence Chart(MSC). ITU-T, Geneva, 1994.

[4] ITU-T. Recommendation Z.120 - Annex B: Message Sequence Chart (MSC). ITU-T, Geneva, 1995.

[5] R. Milner. *Communication and Concurrency*, Prentice Hall International(UK) Ltd.,1989.

[6] E. Rudolph, J. Grabowski and P. Graubmann. Tutorial on Message Sequence Charts(MSC'96). *Computer Networks and ISDN Systems -SDL and MSC*, 28(12), JUN, 1996.