시간제약 속성을 지원하는 상호작용 컨트랙트

엄충용° 신정민 류성열
숭실대학교 대학원 컴퓨터학과
{umcy°, shinjm}@selab.ssu.ac.kr, syrhew@computing.ssu.ac.kr

An Interaction Contract supporting Time-Constraint Properties

ChungYong Eom° JungMin Shin SungYul Rhew
Dept. of Computer Science, Soongsil University

요 약

컴포넌트의 합성을 통하여 신뢰성 있는 시스템을 구축하려면 명시된 품질 속성을 만족해야 한다. 이를 위해서는 잘 정의된 상호작용을 바탕으로 하여 비기능적 속성이 명세되어야 한다. 그러나 기존의 컴포넌트 명세 방법들은 컴포넌트의 기능 속성에 주로 초점을 맞추어 왔으며 비기능적 속성에 대한 지원이 미약하다. 본 논문에서는 비기능적 속성들을 명세하는 방법에 초점을 둔다. 구체적으로, 비기능적 속성들 중에서 시간제약 속성을 상호작용 컨트랙트에 정의하고 UML 다이어그램으로 표현하는 방법을 제시한다. 정형화된 구조를 가진 컨트랙트에서 시간제약 속성들은 보다 명확한 의미를 가지며, 다양한 형태의 컴포넌트 합성에서 검증이 가능하다.

1. 서 론

컴포넌트의 합성을 통한 소프트웨어 시스템을 구축하는 컴포넌트-기반 개발 방법론이 제시되고 있다. 이러한 접근방법은 소프트웨어 시스템을 구축할 때 규격화되고 독립적인 빌딩블록들을 조립하는 데 초점을 둔다. 이러한 컴포넌트들의 사용은 비용과 노력을 크게 감소시킬 뿐만 아니라 검증된 컴포넌트의 사용으로 시스템의 신뢰성을 높일 수 있다[1].

신뢰성 있는 컴포넌트 시스템의 구축을 위해서는 아키텍처에 명시된 품질 속성을 만족하는 것이 중요하다. 이를 위해서는 컴포넌트 사이에 상호작용이 잘 정의되어 있어야 하고, 이를 바탕으로 비기능적 속성이 명세되어야 한다[4]. 하지만 기존의 컴포넌트 명세 방법에서는 컴포넌트가 가진 기능적 속성을 명세하는 데 주로 초점을 맞추어 왔다. 이것은 시스템의 기능을 나타내기에는 유용하지만, 요청한 서비스를 올바르게 수행한다는 보장을 하지 않는다[3]. 시스템의 품질 속성을 정확히 얻어내기 위해서 컴포넌트의 비기능적 속성을 잘 정의되고 검증가능한 형태로 명세에 반영할 수 있어야 한다.

이에 본 논문에서는 비기능적 속성들을 명세하는 방법에 초점을 둔다. 구체적으로 비기능적 속성들 중 시간제약 속성을 상호작용 컨트랙트에 정의하고, UML 다이어그램으로 이를 표현하는 방법을 제시한다. 정형화된 구조를 가진 컨트랙트에서 시간제약 속성들은 보다 명확한 의미를 가지게 되며, 다양한 합성 형태에서 검증이 가능하다.

2장 관련 연구에서는 컨트랙트 및 비기능적 속성의 명세 방법을 분석하고 요약한다. 3장에서는 비기능 속성을 지원하기 위한 컨트랙트를 제안하고 UML 표현 방안을 제시한다. 4장에서는 제시한 컨트랙트가 다양한 합성 형태에 적용될 수 있음을 보인다.

2. 관련 연구

2.1 컨트랙트

컨트랙트(contract) 란 둘 이상의 당사자 사이에서 성립되는 규약(obligation)으로서, 컴포넌트가 가져야할 기능 혹은 상호작용 컨텍스트를 명세한 것이다[3]. 기존의 컴포넌트를 명세하는 방법으로는 인터페이스 혹은 IDL 등의 명세 언어가 주로 사용되고 있다. 하지만 컴포넌트 합성(composition)에서는 컴포넌트의 기능적 컨텍스트와 더불어 상호작용 컨텍스트 정보까지 명세하여야 한다. 그러므로 단지 컴포넌트 기능을 명세하는 컴포넌트 컨트랙트뿐만 아니라 합성에 참여하는 컴포넌트 간의 상호작용 컨트랙트가 필요하다.

잘 정의된 컨트랙트가 가지는 장점은 첫째, 정형화된 표현으로 의미를 명확하게 기술하여 검증할 수 있고, 둘째, UML 다이어그램과 명세를 분리함으로써 모델의 복잡성을 줄일 수 있다는 점이다.

현재 컴포넌트의 기능에 관한 명세 연구는 많이 있지만 상호작용에 관한 명세 연구는 미흡하다. 다양한 컴포넌트 상호작용 및 합성 형태를 지원하는 합성 컨트랙트[4]가 제안되었으나 단점은 비기능적 속성이 정의되지 않았다. 그러므로 본 논문에서는 시간제약 속성을 정의한 상호작용 컨트랙트를 제시한다.

2.2 비기능적 속성의 분류 및 명세

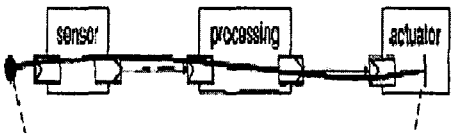
컴포넌트가 가져야 할 비기능적 속성들은 다음과 같이 분류될 수 있다[1].

- 생산성 (Productivity)
- 품질 (Quality)
- 제품 안정성 (Product stability)
- 재사용 (Reuse)
- 설계 및 코딩 (Design and coding)
- 성능 (Performance)

이 중에서 고려해야 할 중요한 속성은 성능과 관련된 시간제약 속성이다. 그것은 컴포넌트 상호작용에 밀접하게 관련된 속성 중 하나이며 정량화하기 쉽다는 점에서 본 논문에서 명세 대상으로 선정하였다.

컴포넌트 합성 시 일어나는 상호작용에 대해 시간제약 속성을 반영하기 위해서는 컴포넌트 사이에 발생하는 행위를 분석한 다음, 이를 기반으로 시간제약 속성을 부여하는 방법을 사용할 수 있다. 이를 위해 Use Case Maps(UCMs) 라는 모델을 사용한다[5,6]. UCMs 는 시스템에서 나타나는 행위들을 경로(path)로 가시화하여 표현하며 이는 유스케이스 시나리오에 대응된다. UCMs의 장점은 시나리오의 복잡한 기술이 없이도 복잡하고 커다란 시스템의 행위를 분석하는 데 유용하며, 요구사항 명세 및 설계 사이의 gap을 연결하는 역할을 한다.

컴포넌트 간의 상호작용에서 발생하는 경로는 이벤트의 흐름으로 표시하고, 이를 트랜잭션이라고 부른다. 트랜잭션의 시간제약 속성은 주기와 마감시간, 시작시간으로 표현된다. 또한 경로 사이에서 분기 혹은 통합이 발생할 수 있는데 이를 AND fork, AND join, OR fork, OR join 으로 분류한다. 또한 경로 사이의 차수에 따라 1:N, N:1, N:M 등으로 분류한다[7,8].



<그림 1> 비기능적 속성을 명세한 컴포넌트 상호작용 경로

<그림 1>은 sensor, processing, actuator 컴포넌트로 구성된 간단한 제어 어플리케이션을 나타낸다. 여기에서 하나의 행위를 나타내는 트랜잭션, 즉 sensor를 읽고, 데이터를 처리하고, actuator를 제어하는 경로는, 모두 10ms 이내에 완료되어야 한다. 그러므로 주기 T는 10ms이고 마감시간 D는 주기와 같다. 시작 시간 T0의 값은 0ms이다. 즉 이 활동은 어플리케이션이 시작할 때 즉시 시작되어야 함을 의미한다.

본 논문에서는 UCMs를 기반으로 한 컴포넌트 상호작용의 비기능적 속성 명세 방법을 상호작용 컨트랙트에 반영하고, 이를 UML로 표현하는 방안을 제시한다.

3. 시간제약 속성을 명세한 상호작용 컨트랙트

3.1 컨트랙트의 구조

시간제약 속성을 반영한 상호작용 컨트랙트를 제안한다. UCMs에서 사용하는 개념을 사용하여, 합성에 참여하는 컴포넌트 사이에 발생하는 상호작용을 트랜잭션으로 정의하였다. 또한 각각의 트랜잭션에 대하여 시간제약 속성을 부여하기 위해 주기, 시작시간, 마감시간을 상호작용 컨트랙트에 명세하였다. <그림 2>는 본 논문에서 제안한 컨트랙트의 구조를 묘사한다.

```
INTERACTION CONTRACT contract_name
IMPORT import_component_name, ..

{
   import_component_name
       (* REQUIRE *)   service_name(), .,
       (* PROVIDE *)   service_name(), ..
} +
{
   TRANSACTION transaction_name
      (* MECHANISM *)
         transaction_sequence
      (* CONSTRAINT *)
         T = period,  T0 = start_time, D = deadline,
         ..
} +
```

<그림 2> 상호작용 컨트랙트 구조

interaction constract 부분에서는 컨트랙트의 이름을 정의하고, import 에서는 합성에 참여하는 컴포넌트들의 리스트를 나열한다. 그 다음으로, 각 컴포넌트의 서비스를 요구(require) 또는 제공(provide) 서비스로 분류하여 기술한다. transaction 에서는 컨트랙트에 정의되어질 상호작용을 정의하는 부분으로, 유일하게 식별되는 이름을 할당한다. mechanism 부분에서는, 트랜잭션이 발생하는 메커니즘을 기술한다. 기본적으로 트랜잭션의 시퀀스, 즉 이벤트가 호출되는 순서가 기술된다. 또한 경로 상에서 발생하는 분기 및 통합, 그리고 차수를 고려할 때 다음과 같이 기술되어질 수 있다.

normal sequence	event1 -> event2 -> ..
OR fork	event1 -> event2 | event3
OR join	event1 | event2 -> event3
AND fork	event1 -> event2 & event3
AND join	event1 & event2 -> event3
N:M	event1 & event2 & .. -> event3 & ..

<표 1> 트랜잭션 메커니즘의 표현

OR의 경우 경로가 선택적(alternative)인 반면, AND의 경우 모든 경로에 대해 동기적(synchronized)이다.

그 다음에 constraint 부분에서는 비기능적 속성, 즉 본 논문에서는 시간제약 속성인 주기(T), 시작시간(T0), 마감시간(D)을 명세한다. 이를 통해 컴포넌트 합성시 발생하는 각 트랜잭션에 대해 명세된 시간제약 속성에 따라 처리되어야 함을 의미하고 있다.

제시된 상호작용 명세는 기존의 컴포넌트에 대한 행위적 상호작용 패턴뿐만 아니라 각 트랜잭션에 따른 시간 제약 속성까지도 명세함으로써 컴포넌트 합성시 발생하는 이벤트에 대한 비기능적 속성을 표현하는 것이 가능하다.

3.2 컨트랙트의 UML 표현

이 장에서는 제시한 컨트랙트가 UML에서 표현될 수 있는 방법을 제시한다. 예를 들어 <그림 1>에서의 상호작용 경로를 고려할 때, 제안된 상호작용 컨트랙트로 <그림 3>과 같이 명세될 수 있다. 즉 control이라는 상호작용 컨트랙트에 참여하는 컴포넌트들은 sensor, processing, actuator 이며, 이들간의 상호작용 경로는 data_process 라는 이름의 트랜잭션으로 정의한다. 이벤트가 전달되는 순서는 input() -> process() -> output() 이고, 여기에 비기능적 속성을 명세한다.

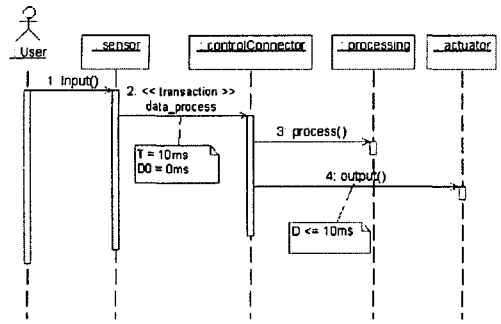
```
INTERACTION CONTRACT control
IMPORT sensor, processing, actuator

.. (중략)..

TRANSACTION data_process
   (* MECHANISM *)
      sensor.input() →
      processing.process() →
      actuator.output()
   (* CONSTRAINT *)
         T  = 0.1
         T0 = 0
         D  = 0.1
```

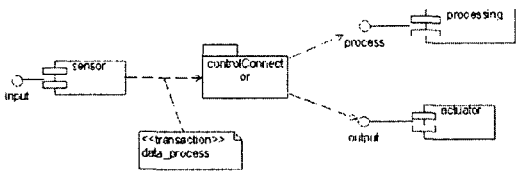
<그림 3> control 상호작용 컨트랙트의 예

상호작용 컨트랙트에서 명세된 모든 트랜잭션은 그것을 관리하는 연결자(connector)로 구현되어질 수 있다. 만일 컴포넌트를 클래스의 개념으로 간주할 때 상호작용에 정의된 모든 트랜잭션은 각각 시퀀스 다이어그램으로 표현될 수 있다. 이 때 시간제약 속성은 <그림 4>에서 묘사되는 바와 같이 주해될 수 있다. 시간제약 속성들 중에서 주기 T와 시작시간 T0 는 시작점에 명세되어야 하고, 마감시간 D는 시퀀스의 종료점에 명세되어야 한다.



<그림 4> data_process 트랜잭션을 나타낸 시퀀스 다이어그램

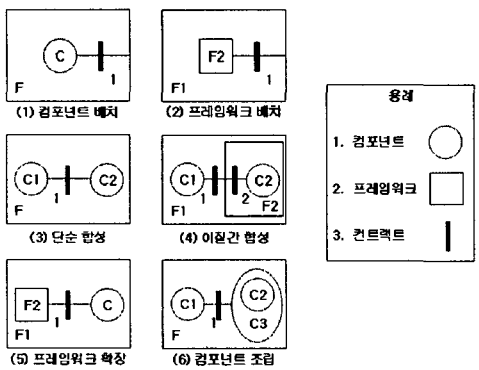
또한 이를 컴포넌트 다이어그램으로 표현할 경우 <그림 5> 와 같이 나타낼 수 있다.



<그림 5> data_process 트랜잭션을 나타낸 컴포넌트 다이어그램

4. 다양한 합성 형태의 지원

제시한 상호작용 컨트랙트가 다양한 합성 형태에 적용될 수 있음을 검증한다. 컴포넌트 합성에 참여하는 요소로는 컴포넌트, 프레임워크가 있으며, 이들간의 결합을 통해 나타나는 합성 형태는 <그림 6>과 같이 6가지로 분류된다[3]. 상호작용 컨트랙트는 그 자체로 컴포넌트의 컨텍스트 정보를 표현한 컴포넌트 컨트랙트(contract)의 기능을 수행하면서, 또한 합성시 필요한 상호작용 패턴과 비기능적 속성을 표현할 수 있다. 그러므로 <그림 6> 의 다양한 합성 형태를 모두 처리할 수 있다.



<그림 6> 컴포넌트 합성 형태

1) 컴포넌트 배치
컴포넌트는 그것이 구성되거나 실행되기 전 프레임워크 안에 배치되어야 한다. 상호작용 컨트랙트는 컴포넌트가 구현되어 프레임워크에 배치될 때 필요한 컴포넌트의 컨텍스트 정보를 제공하기 때문에 배치 컨트랙트와 동일한 역할을 수행한다.

2) 프레임워크 배치
Szyperski 의 다중 아키텍처 개념에 따라 프레임워크는 또다른 프레임워크 안에 배치되는 형태이다. 이때 컴포넌트 및 상호작용 컨트랙트로 서로 구성되는 프레임워크는 그 자체로 하나의 컴포넌트로 볼 수 있기 때문에, 프레임워크 배치는 컴포넌트 배치 형태와 유사하다.

3) 단순 합성
같은 프레임워크 안에 배치된 컴포넌트는 서로 합성이 가능하다. 기존의 컨트랙트와는 달리 상호작용 컨트랙트를 사용할 때 두 컴포넌트를 연결시키면서 발생하는 상호작용 패턴을 관리할 수 있다.

4) 이질간 합성
다중 프레임워크에 대한 지원은 <그림 6>의 (4) 에서 보여지듯 프레임워크 간, 계층적 혹은 수평적 프레임워크 사이의 컴포넌트 합성을 내포한다. 기존의 컨트랙트에서는 일반적인 컴포넌트 모델에 걸치는 상호작용을 지원하는 컨트랙트에 추가적으로 연결 계약이 필요하다. 그러나 상호작용 컨트랙트를 사용할 경우 프레임워크는 내부에 컴포넌트와 상호작용 컨트랙트를 포함한 복합 컴포넌트로 볼 수 있으며 컴포넌트 어셈블리와 같은 형태의 합성이 되기 때문에 프레임워크를 위한 별도의 컨트랙트가 필요 없다.

5) 프레임워크 확장
프레임워크는 컴포넌트처럼 취급되며 또다른 컴포넌트와 함께 구성된다. 이 때 프레임워크 확장 또한 단순 합성과 같은 형태를 갖게 된다.

6) 컴포넌트 어셈블리
컴포넌트-기반 시스템은 컴포넌트의 조합으로 구성된다. 이 때 컴포넌트 어셈블리는 하나의 복합 컴포넌트가 다른 컴포넌트와 서로 합성되는 단순 합성과 유사한 형태를 갖게 된다.

5. 결 론

본 논문에서는 비기능적 특성을 지원하는 상호작용 컨트랙트를 제안하였다. 기존 컴포넌트 합성 기법에서는 기능적인 속성만을 고려하고, 비기능적 속성의 지원은 한계를 갖고 있다. 그러나 그러나 본 논문에서 제시한 컴포넌트 상호작용 컨트랙트를 이용할 때, 각각의 컴포넌트 합성시 발생할 수 있는 상호작용과 그의 시간제약 속성을 명세할 수 있으므로 보다 높은 신뢰성을 보장할 수 있다.

향후 상호작용 컨트랙트의 구현 및 비기능적 속성의 검증 방법에 대한 연구가 더 필요하다.

참 고 문 헌

[1] George T. Heineman, William T. Councill, *Component-Based Software Engineering*, Addison Wesley, pp. 42-43, 2001
[2] Felix Bachman, Ien Bass, Charles Buhman, Santiago Commella-Dorda, Fred Long, John Robert, Robert Seacord, Jurt Wallnau, "Volume II : Technical Concepts of Component-Based Software Engineering,", CMU/SEI-2000-RT-008, 2000
[3] A. Beugnard, J-M Jezequel, and D.Watkins. "Making Components Contract Aware," IEEE Computer, July, 1999
[4] 백경원, 박성은, 이정태, 류기열, "컴포넌트 결합 명세서에 기반한 컴포넌트 결합 모델", 정보처리학회논문지 D 제8-D권 제6호, 2001
[5] U.Rastofer and F.Bellosa, "Component-Based Software Engineering for Distributed Embedded Real-time Systems", IEE Proc.-Softw. Vol. 148, No. 3, 2001
[6] CORNWELL P.D, "Reusable component enginerring for hard real-time systems", Ph.D dissertation, Department of Computer Science, University of York, UK, 1998
[7] Daniel Amyot, "Use Case Maps Quick Tutorial, version 1.0", SITE, University of Ottawa, 1999
[8] Danial Amyot and Gunter Mussbacher, "On the Extension of UML with Use Case Maps Concepts", SITE, University of Ottawa, 1999