C2 아키텍쳐 스타일을 위한 UML2.0 프로파일의 확장

(Extending UML2.0 Profile of the C2 Architecture Style)

노 성 환 † 전 태 웅 †† 승 현 우 †††
(Sunghwan Roh) (Taewoong Jeon) (Hyonwoo Seung)

요 약 소프트웨어 아키텍쳐는 소프트웨어 시스템의 상위 수준 모델이다. 이러한 소프트웨어 아키텍쳐는 명료하고 엄밀한 표현을 위해 아키텍쳐 기술 언어(ADL)를 사용하여 기술된다. 하지만 대부분의 ADL들은 C2 스타일 ADL(C2SADL)처럼 소프트웨어 시스템의 개발에 필요한 요소들 중에서 특정한 관점만을 고려하여 정의되었거나 별도의 표기형식으로 배워야 하는 부담이 있기 때문에 아키텍쳐를 명세하는 언어로서 정착되지 못하였다. 반면 UML은 범용 모델링 언어로서 소프트웨어 개발의 전 과정에 일관된 표기형식과 폭넓은 지원도구들을 제공하고 있으므로 소프트웨어 개발을 위한 사실상의 표준 언어로 자리잡았다. 그러나 UML은 소프트웨어 아키텍쳐를 표현하도록 설계된 것은 아니기 때문에 UML을 사용하여 아키텍쳐를 표현하기 위해서는 UML을 확장, 변경하여야 한다. 본 논문에서는 UML2.0에 기반한 C2 아키텍쳐 모델링 언어 프로파일을 정의하였다. 정의된 C2 아키텍쳐 모델링 언어는 식당 예약 시스템을 모델링 하는데 사용되었다.

키워드 : 소프트웨어 아키텍쳐, 아키텍쳐 기술 언어, 아키텍쳐 모델링 언어, UML2.0 프로파일, UML2.0 메타모델, ACME, C2 스타일

Abstract Software architecture is the high level model of a software system. It should be specified with ADLs (Architecture Description Languages) for its clarity and preciseness. Most ADLs such as C2SADL have not come into extensive use in industries since ADL users should learn a distinct notation specific to architecture, and ADLs do not address all stakes of the development process that is becoming diversified everyday. On the other hand, UML is a de facto standard general modeling language for software developments. UML provides a consistent notation and various supporting tools during the whole software development cycle. But, UML is a general modeling language and does not provide all concepts that are important to architecture description. UML should be extended in order to precisely model architecture. In this paper, we defined a C2 architecture modeling language as a UML2.0 profile. We applied the defined C2 architecture modeling language to the modeling of a restaurant reservation system.

Key words : Software architecture, ADL: Architecture Description Language, architecture modeling language, UML2.0 profile, UML2.0 metamodel, ACME, C2 style

1. 서 론

소프트웨어 아키텍쳐는 상위 추상화 수준에서 소프트웨어 시스템을 구성하는 컴포넌트들, 이들의 상호연결 및 상호작용 관계들(커넥터), 컴포넌트와 커넥터 인스턴스들이 형성하는 형세(configuration), 그리고 컴포넌트와 커넥터들의 합성 규칙 등으로 구성된다. 이러한 소프트웨어 아키텍쳐는 아키텍쳐 기술 언어(ADL: Architecture Description Language)를 사용하여 기술되어야 정확하고 엄밀한 아키텍쳐 모델링과 아키텍쳐에 기반한 시스템의 분석, 정제, 검증이 가능하다. 이에 따라 아키텍쳐를 명시적이고 정확하게 기술할 수 있는 아키텍쳐 기술 언어와 ADL 지원 환경에 대한 연구가 외국에서 활발히 진행되어 왔으며 현재 다양한 ADL들과 지원도

· 본 연구는 유비쿼터스컴퓨팅 및 네트워크 원천기술개발 사업단 지원으로 수행되었음
† 정 회 원 : 삼성전자 반도체총괄 SOC연구소 연구원
sunghwan.roh@samsung.com
†† 종신회원 : 고려대학교 컴퓨터정보학과 교수
jeon@korea.ac.kr
††† 정 회 원 : 서울여자대학교 컴퓨터학부 교수
hwseung@swu.ac.kr
논문접수 : 2005년 3월 29일
심사완료 : 2005년 11월 7일

구들이 소개되어 있다. 하지만 대부분의 ADL은 소프트웨어 시스템의 개발에 필요한 요소들 중에서 특정한 관점만을 고려하고 있으며, 실제 소프트웨어 개발에 필요한 요소들을 통합하지 못하고 있다. 예를 들어 C2와 같은 아키텍쳐 스타일(style-specific) ADL들의 경우에 ADL이 지원하는 특정 아키텍쳐 스타일을 따르는 아키텍쳐의 기술에는 적합하지만 이와 다른 스타일을 따르는 아키텍쳐 또는 여러 스타일들이 병용된 아키텍쳐의 모델링에는 부적합하다. 즉 스타일에 대한 범용성이 낮다.

한편 UML1.x는 요구 분석, 시스템 설계, 시스템 개발 등의 소프트웨어 개발 과정에서 얻어지는 다양한 소프트웨어 산출물들을 표현하는 표준 모델링 언어가 되었다. 따라서 응용 프로그램의 소프트웨어 아키텍쳐를 표현하는데 있어서 UML1.x를 사용하려는 시도가 많이 있어 왔다. UML을 사용함으로써 소프트웨어 개발 기간 동안 일관된 모델을 유지할 수 있고 기존의 도구들의 지원을 받을 수 있는 장점이 있다. 그러나 UML1.x은 소프트웨어 아키텍쳐의 개념을 표현하도록 문법적 또는 의미적으로 설계된 것은 아니기 때문에 UML1.x를 사용하여 아키텍쳐를 표현하기 위해서는 UML1.x를 확장 또는 변경하여야 한다. 곧 발표될 UML2.0에서는 이전 버전에서 미흡했던 아키텍쳐 모델링에 유용한 개념들이 많이 추가되었다. 그러나 여전히 UML2.0으로 명시적인 표현이 어려운 아키텍쳐의 핵심 개념들이 존재한다. 또한 UML2.0은 아키텍쳐 기술에 불필요하거나 관련이 적은 다른 모델링 요소들도 많이 포함하고 있다.

본 논문에서는 이전 연구에서 정의된 Generic ADL-UML 프로파일을, 문헌에 구체적으로 잘 정의되어 있고 응용사례 연구가 풍부한 대표적인 아키텍쳐 스타일 중의 하나인 C2 스타일의 아키텍쳐를 기술하는 ADL로 어떻게 확장할 수 있는가를 살펴 본다. UML2.0 메타모델과 일관성을 유지하도록 하면서, UML2.0을 확장하는데 필요한 제약 조건을 OCL을 이용하여 정형 명세하여 C2 아키텍쳐 모델링 언어 프로파일의 메타모델을 정의한다. 그리고 정의된 아키텍쳐 프로파일을 사용하여 식당 예약 시스템의 아키텍쳐를 설계한다.

본 논문의 구성은 다음과 같다. 2장에서는 본 연구의 배경 지식을 설명한다. 3장에서는 본 논문의 연구와 관련된 기존의 관련 연구들을 살펴본다. 4장에서는 Generic ADL을 확장하여 C2 프로파일을 정의하는 과정을 설명한다. 5장에서는 식당 예약 시스템 아키텍쳐를 사례 연구로써 모델링한다. 6장에서는 본 논문의 결론 및 향후 연구를 설명한다.

2. 배경 지식

2.1 C2 아키텍쳐 스타일

C2 스타일[1]은 UCI(University of California in Irvine)에서 GUI를 갖는 응용 소프트웨어 개발을 지원하기 위하여 설계한 아키텍쳐 스타일이며 그 밖의 다른 타입의 응용 소프트웨어 개발도 잘 지원한다. 다른 아키텍쳐 스타일과 마찬가지로 C2 스타일은 컴포넌트들과 커넥터들로 구성된다. 그림 1은 이러한 C2 스타일의 아키텍쳐의 예를 보여준다.

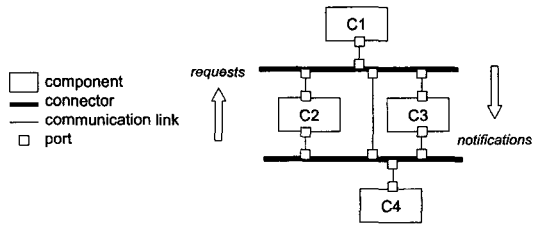

그림 1 C2 아키텍쳐 스타일

C2 스타일의 컴포넌트는 두 개(top, bottom)의 포트를 통해 들어오는 메시지(incoming message)에 대해 내부적으로 선언된 메소드를 호출하거나 top 또는 bottom 포트를 통해 나갈 메시지(outgoing message)를 생성한다. 컴포넌트가 처리 또는 생성하는 메시지는 top 포트를 통해 나가거나 bottom 포트를 통해 들어오는 요청(request) 메시지와 top 포트를 통해 들어오거나 bottom 포트를 통해 나가는 공고(notification) 메시지로 구분된다. C2 스타일의 모든 컴포넌트들은 메시지 교환을 통해 상호작용을 한다. C2 스타일의 커넥터는 컴포넌트들 간에 교환되는 메시지를 라우팅(routing), 브로드캐스팅(broadcasting), 또는 메시지 필터링(message filtering)하는 역할을 한다.

C2 스타일에서 컴포넌트와 커넥터에 대한 규칙과 이들의 합성 규칙은 다음과 같다.

- C2 규칙 1번: C2 컴포넌트는 하나의 top 포트와 하나의 bottom 포트를 갖는다(그림 2).

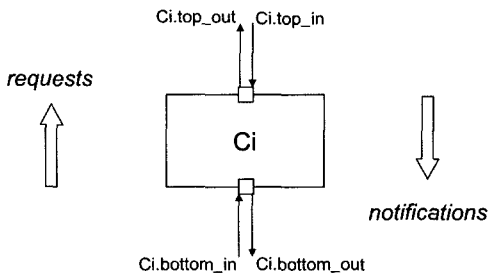

그림 2 C2 컴포넌트

- C2 규칙 2번: C2 커넥터는 연결된 컴포넌트 또는 커

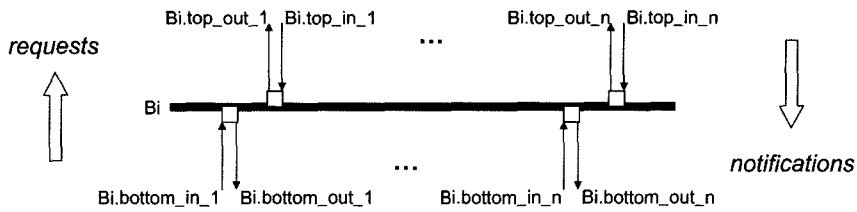

그림 3 C2 커넥터

넥터에 따라서 여러 개의 top과 bottom 포트를 가질 수 있다(그림 3).

- C2 규칙 3번: C2 컴포넌트의 top 포트는 커넥터의 bottom 포트와 연결된다.
- C2 규칙 4번: C2 컴포넌트의 bottom 포트는 커넥터의 top 포트와 연결된다.
- C2 규칙 5번: 두 개의 C2 커넥터가 연결될 때는 한 커넥터의 bottom 포트로부터 다른 커넥터의 top 포트로 연결되거나, 또는 한 커넥터의 top 포트로부터 다른 커넥터의 bottom 포트로 연결된다.

2.2 아키텍쳐 기술을 위한 프로파일의 계층적 구조

본 연구의 접근 방식은 그림 4에서와 같이 스타일에 독립적인 요소와 스타일에 고유한 핵심 요소를 각각 Generic ADL과 SS-ADL(Style-Specific ADL)로 정의하고, 정의된 SS-ADL을 사용하여 해당 스타일이 적용된 응용 도메인에서 재사용 가능한 요소들을 프레임워크 아키텍쳐 모델로 구축하는 것이다[2]. 먼저, 스타일에 독립적인, 모든 아키텍쳐 모델이 갖는 공통적 개념들의 어휘를 제공하는 Generic ADL을 UML의 확장 메커니즘을 이용하여 UML2.0 프로파일로 정의한다. Generic ADL은 그 자체만으로 아키텍쳐 모델링이 가능한 수준을 유지하도록 일반적인 아키텍쳐 모델링을 위한 핵심 개념들을 모두 포괄하도록 정의한다. 그런 후 이를 확장하여 스타일에 종속적인 아키텍쳐 상의 개념을 표현하는 SS-ADL을 정의한다. 해당 스타일에 속한 특정 목표 시스템의 아키텍쳐는 SS-ADL을 사용하여 설계된다.

2.3 Generic ADL의 메타모델

이전 연구인 [2]에서는 UML2.0을 확장하여 Generic ADL을 정의하였다. 그림 5는 UML2.0의 메타클래스들을 기반으로 하여 정의된 Generic ADL의 메타모델이다. 그림 5에서 흰색 바탕으로 표시된 모델 요소들은 UML2.0의 메타클래스이고 어두운 바탕색은 UML2.0 메타클래스의 스테레오타입으로 정의되어 Generic ADL에 새롭게 추가된 모델 요소들이다. Arch component는 자신이 소유한 외부 포트(ownded port)로서 arch port를 갖는다. 그리고 part port는 ArchComponent의 내부 구조를 구성하는 part들의 연결점을 나타내는 포트들이다. 컴포넌트의 내부 구조를 표현하는 경우 arch connector instance는 connector end의 part with port로써 part port를 참조한다. 즉, 컴포넌트의 내부 구조는

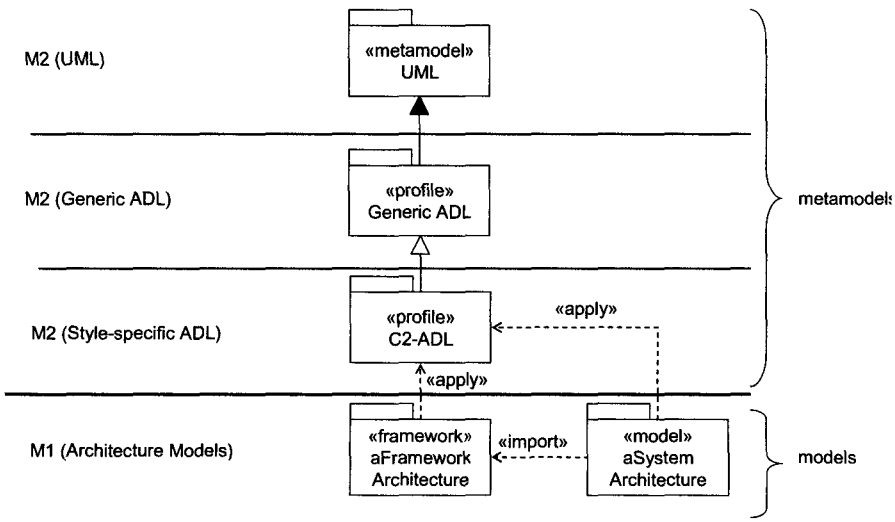

그림 4 UML2.0 기반 ADL의 메타모델 아키텍쳐

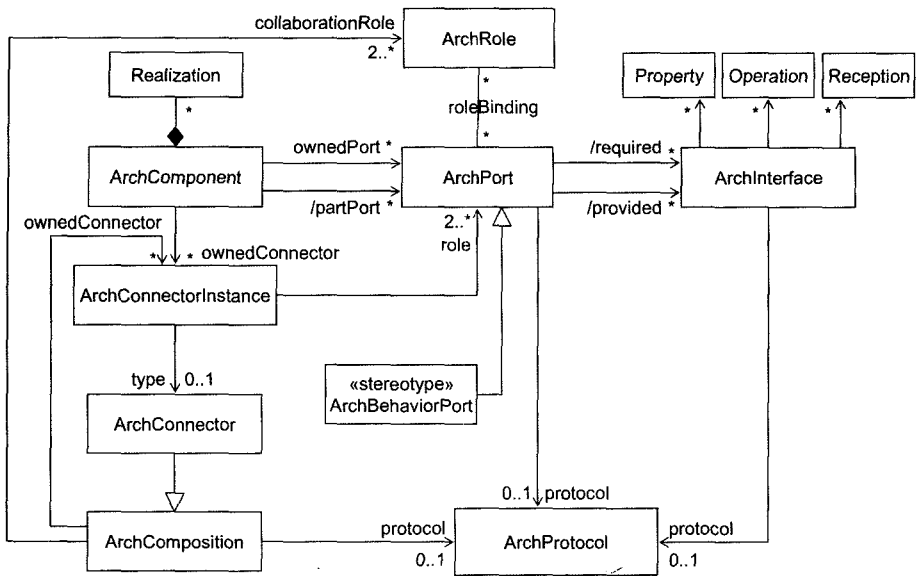

그림 5 Generic ADL의 메타모델

포트들의 연결 관계로 정의된다. Arch port는 자신이 속한 컴포넌트가 제공하는 행위적 특성(provided interface)과 필요로 하는 행위적 특성(required interface)을 나타내기 위해 arch interface를 참조한다. Arch interface는 property, operation, reception을 feature로 가짐으로써 동기적, 비동기적인 상호작용을 모두 표현할 수 있다. Arch interface는 protocol statemachine을 통해 행위적인 계약관계를 명세할 수 있으며 arch port는 자신이 참조하는 required, provided arch interface의 protocol statemachine의 합성으로 포트 수준에서의 계약관계를 명세할 수 있다.

Collaboration의 스테레오타입으로 arch composition을 정의한다. 이것은 컴포넌트 간의 합성 패턴을 나타내는 것으로 collaboration의 role에 참여하는 개체를 arch port로서 한정함으로써 arch role을 정의하고 이러한 arch role을 갖는 collaboration으로서 arch composition이 정의된다. Arch role은 collaboration occurrence의 role binding 관계를 이용하여 arch port로 대응된다. 또한 합성 관계를 나타내는데 있어 커넥터를 arch connector instance로 한정한다. 또한 arch component는 behaviored classfier로서 행위적 명세를 가질 수 있고 statemachine이나 activity 모델 등을 통해 행위를 표현할 수 있다. Arch connector는 arch composition을 특화하여 정의되는 것으로 Arch connectorIntance의 타입을 정의한다. 또한 protocol statemachine을 이용하여 상호작용 프로토콜을 명세할 수 있다. Arch connectorInstance는 connector의 스테레오타입으로 자신의 타입으로 collaboration의 스테레오타입인 arch connector를 참조할 수 있다. 그리고 자신의 인터페이스로서 arch port 개체를 참조할 수 있는데, 이것은 arch connector instance의 타입인 arch connector의 arch role에서 role binding을 통해 대응시킨 arch port의 타입에 맞는 포트가 된다.

3. 관련 연구

UML과 같은 표준 언어를 사용하여 기술된 아키텍처는 이해가 쉽고, 일관성 있게 개발되며, 기존 도구들의 지원을 받을 수 있다. UML은 영역 모델링과 구현 모델링에 효과적으로 사용되어 왔기 때문에 UML로 표현된 아키텍처는 보다 쉽게 설계 및 구현으로 상세화(refinement)될 수 있다. 이러한 장점들로 인해 UML을 이용해 아키텍처를 표현하려는 여러 연구들이 진행되어왔다. 이러한 연구들은 크게 다음 세 가지로 분류될 수 있다. 첫 번째는 기존의 UML 구조물들을 변경 없이 그대로(as is) 이용하여 아키텍처 구조물들을 표현하는 방법이다. 두 번째는 UML의 메타모델을 수정하여 UML과 비슷한 형태의 새로운 아키텍처 모델링 언어를 정의하는 heavyweight 방법이다. 세 번째는 UML의 확장 메커니즘을 사용하여 UML 메타모델의 변경 없이 아키텍처 개념을 표현하는 새로운 구조물들을 정의하는 lightweight 방법이다.

[3]에서는 ACME로부터 UML1.x로 매핑하는 방법을 연구하였다. 이 논문은 UML1.x의 구조물을 이용하여 아키텍처의 구조물을 표현하는 여러 방법들 간의 장단

점을 비교하였다. 그러나 UML1.x의 구조물들을 변경 없이(as is 방법) 아키텍쳐 구조물로 매핑하였기 때문에 아키텍쳐에 필요한 개념들이 충분히 표현되지 못하였다.

[4]는 heavyweight 방법을 사용하였으며 UML 메타 모델을 직접 수정하여 새로운 아키텍쳐 모델링 언어를 정의하였다. Heavyweight 방법은 아키텍쳐의 개념을 정확하게 표현하는 새로운 언어를 정의할 수 있는 장점이 있다. 반면에 heavyweight 방법을 통해 새로 정의된 아키텍쳐 모델링 언어는 UML과는 다른 언어이므로 UML을 지원하는 기존 툴을 이용할 수 없는 단점이 있다. 따라서 다수의 다른 논문들에서는 lightweight 방법을 이용하여 아키텍쳐 개념을 표현하고 있다.

[5]에서는 lightweight 방법을 기반으로 UML1.x을 확장하여 C2 아키텍쳐 스타일과의 통합을 시도하였으며, [6]에서는 [5]를 더욱 발전시켜 C2 외에 Wright 아키텍쳐 스타일과의 연결을 제안하였다.

[7]에서는 ADL의 주요 개념인 컴포넌트, 커넥터, 형세와 아키텍쳐 관점(view point)를 표현하기 위하여 UML을 어떻게 확장할 수 있는지에 대하여 논하였다. 이 논문에서는 일반적인 아키텍쳐의 개념들을 UML 프로파일로 표현할 수 있음을 보였으며 커넥터의 표현을 위해 UML collaboration을 확장하였다.

[8]에서는 UML을 변경 없이 사용하는 방법과 확장하여 사용하는 방법을 각각 적용하여 C2 아키텍쳐 스타일의 개념을 표현하였으며 이 두 방법을 비교 분석하였다. 이 논문에서는 UML1.x는 아키텍쳐의 개념을 표현하는데 있어서 부족한 점이 많음을 보이고 있다.

위의[3-8] 논문들은 아키텍쳐 개념을 표현하는 UML 프로파일을 다양한 방법으로 정의하고 있다. 그러나 이 논문들은 모두 아키텍쳐 개념이 부족한 UML1.x를 기반으로 하여 아키텍쳐를 표현하였다. UML1.x와 달리 아키텍쳐의 주요 개념들을 명시적으로 지원하는 컴포넌트, 커넥터, 포트 등의 여러 구조물들이 UML2.0에서 새롭게 추가되었다. [9-11]는 이러한 UML2.0을 이용하여 아키텍쳐 프로파일을 정의하고 있다.

[9]에서는 UML2.0의 새로운 아키텍쳐 모델링 구조물들을 lightweight 확장 방법과 OCL을 이용하여 ACME의 아키텍쳐 구성 요소에 매핑하였다. ACME에서는 포트를 인터페이스의 일종으로 다루고 있지만, UML2.0에서는 포트와 인터페이스를 별개의 개념으로 구분하였다. 따라서 이 논문에서는 UML2.0인터페이스를 아키텍쳐 개념으로 표현하기 위하여 확장할 때 불필요한 제약이 추가되었다.

[10]에서는 C2 아키텍쳐 스타일의 개념을 UML로 정의할 때 UML1.x와 UML2.0간의 표현 능력의 차이를 설명하고 있다. 또한 아키텍쳐 커넥터 개념을 UML2.0 메타 모델을 사용하여 표현하는 것이 어려움을 보여주고 있다.

[11]에서는 ACME를 UML2.0로 매핑하는 방법을 연구하였다. UML2.0의 여러 구조물들을 이용하여 아키텍쳐 구조물들을 표현하고 이용된 각 UML2.0 구조물의 장단점을 비교하였다. 그러나 UML2.0의 구조물들을 변경 없이 아키텍쳐 구조물로 매핑하였기 때문에 아키텍쳐의 개념을 정확하게 표현하지는 못하였다.

위 [9-11]에서는 UML2.0컴포넌트 또는 클래스를 확장하여 아키텍쳐의 커넥터 개념을 표현하였다. 이러한 방법은 커넥터와 컴포넌트 간의 동일한 표기로 인하여 시각적 명확성(visual clarity)이 떨어진다. 그리고 아키텍쳐의 개념을 표현하기 위하여 스테레오타입들을 정의하는 과정에서 나타나는 제약 조건들과 UML2.0 메타모델의 제약 조건들 간의 관계가 명확하지 않다.

4. UML2.0 기반의 C2 스타일 아키텍쳐 모델링 언어

본 장에서는 C2 스타일 ADL을 정의하기 위해 위에서 정의된Generic ADL을 어떻게 확장할 것인가를 설명한다.

4.1 C2 컴포넌트

그림 6과 같이 두 개의 class A, B가 generalization 관계에 있을 때 본 논문에서 정의한 {substituted} 제약조건은 두 가지 의미를 갖는다.

- 첫째로, 특정 프로파일에서는 general class A 대신에 항상 specific class B가 사용되어야 함을 의미한다. 이는 UML 스테레오타입의 {required}와 의미상 유사하다.

```
context A
inv self.oclIsTypeOf(A) implies self.oclIsTypeOf(B)
```

- 둘째로, UML2.0의 {complete, disjoint} 제약 조건의 의미를 포함한다.

```
context B
inv self.generalization.generalizationSet.isCovering
inv self.generalization.generalizationSet.isDisjoint
```

그림 7은 C2 component에 관련된 UML2.0 메타클래스, Generic ADL 스테레오타입, 그리고 C2 스타일ADL 스테레오타입을 보여준다. C2 스타일 ADL의 C2 component 스테레오타입은 Generic ADL의 arch compo-

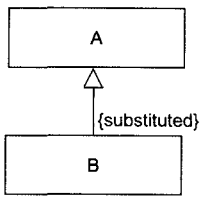

그림 6 {substituted} 제약조건

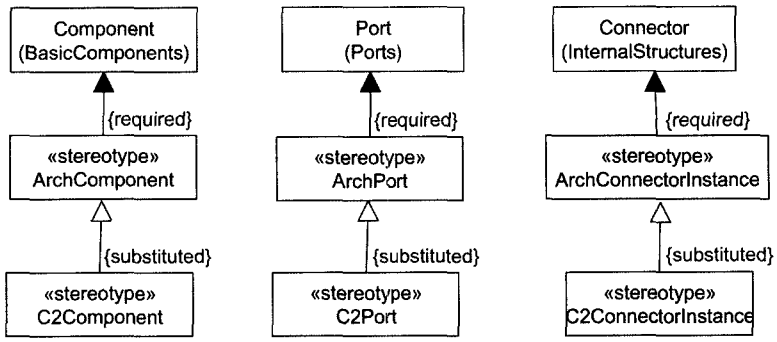

그림 7 C2 컴포넌트, 포트, 커넥터 인스턴스

nent로부터 상속을 받아 만들어진다. 2 *component*와 *arch* component 간의 generalization 관계는 {substituted} 제약 조건을 갖는다. C2 component의 정의와 유사하게 그림 7의 *C2 port*와 *C2 connector instance*가 정의된다.

그림 8은 C2 component와 C2 component instance 및 C2 connector instance 간의 메타모델 관계를 보여준다. *C2 component*의 owned *C2 connector instance*는 C2 component가 갖는 내부의 connector instance들을 나타내며 *arch component*의 owned arch connector로부터 재정의 된다. Owned C2 component instance는 C2 component가 갖는 내부의 component instance를 나타내며 *arch component*의 owned arch component instance로부터 재정의 된다. C2 component의 내부 component instance의 type은 arch component type을 재정의한다.

그림 9는 C2 component와 C2 port 간의 관계를 보여준다. C2Component의 owned C2 port는 component

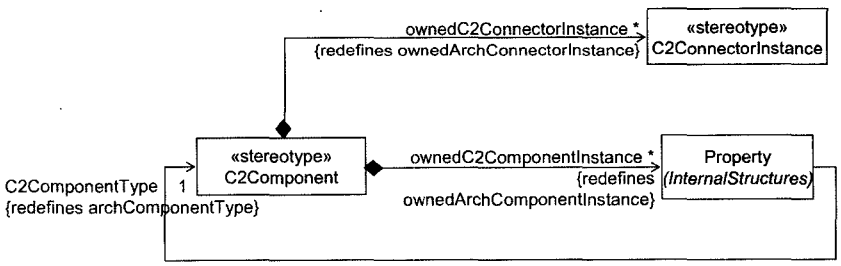

그림 8 C2 컴포넌트의 메타모델

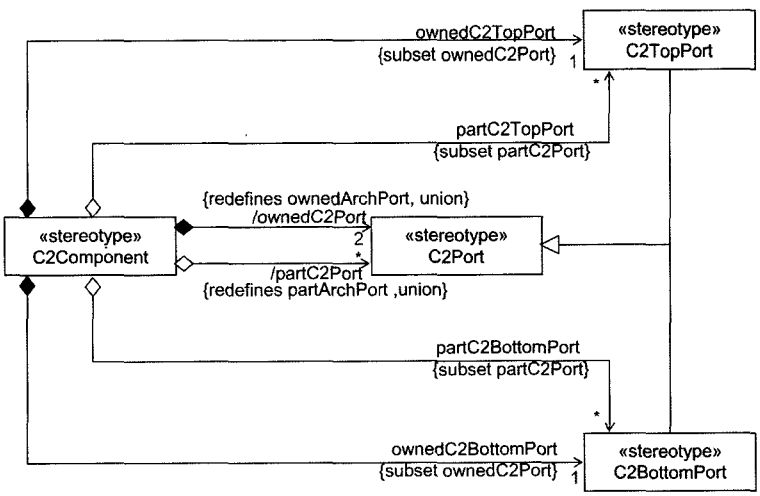

그림 9 C2 컴포넌트와 포트 간의 메타모델

의 외부 port를 의미하며 arch component의 owned arch port를 재정의한다. C2Component의 part C2 port는 component의 내부의 port들을 의미하며 arch component의 part arch port를 재정의한다.

- C2 top port와 C2 bottom port는 C2 port의 하위 클래스로써 정의되며 각각 component의 top과 bottom port를 의미한다. 모든 C2 component는 외부에 top port 한 개와 bottom port 한 개를 가지므로 owned c2 port는 owned c2 top port 1개와 owned c2 bottom port 1개로 구성되는 크기 2의 union을 형성한다. 이것을 통해 위의 C2 규칙 1번을 만족한다.

```
inv self.ownedC2TopPort->size() = 1
inv self.ownedC2BottomPort->size() = 1
inv self.ownedC2Port->size() = 2
inv self.ownedC2Port
= self.ownedC2TopPort->union(self.ownedC2BottomPort)
```

- Part C2 port는 내부 component instance 들의 외부 port 들을 의미하기도 한다. Part C2 top port는 내부 component instance 들의 외부 top port를 의미하며, part C2 bottom port는 내부 component instance 들의 외부 bottom port를 의미한다. 따라서 part C2 port와 part C2 bottom port는 subset으로서 part C2 port의 union을 이루며 서로의 크기는 같다.

```
inv self.partC2Port = self.partC2TopPort->union(self. -
    partC2BottomPort)
inv self.partC2TopPort->size() = self.partC2Bot-
    tomPort->size()
```

그림 10은 C2 component 스테레오타입을 사용하여 표현된 customer component의 예를 보여준다. Customer component는 외부 port로 owned C2 top port와 owned C2 bottom port를 갖고 내부 component instance로 InnerCustomer1과 InnerCustomer2를 갖는다. InnerCustomer1과 InnerCustomer2는 내부 C2 connector instance로 연결된다.

- C2 component의 내부 C2 component instance와 내부 C2 component instance는 part C2 port와 C2 connector role을 통하여 연결된다.

```
inv self.ownedC2ComponentIntance.C2Component-
    Type.ownedC2Port
      = self.ownedC2ConnectorInstance
.C2ConnectorType.C2ConnectorRole.C2RoleBinding
```

4.2 C2 커넥터

그림 11과 12는 C2 architecture와 C2 connector 스테레오타입을 설명한다. C2 role은 C2 architecture 혹은 C2 connector에서 collaboration role을 의미한다. C2 프로파일에서는 C2 architecture와 C2 connector, 그리고 C2 role만이 사용될 수 있다({substituted} 제약조건). C2 architecture 스테레오타입은 C2 스타일 아키텍처의 topology를 나타내며, arch composition으로부터 상속을 받아 정의된다. C2 architecture의 C2 composition role은 arch composition의 arch composition role을 재정의하여 만들어진다.

그림 12에서 arch composition을 확장하며 정의된 C2 architecture의 owned C2 connector instance는 owned arch connector로부터 재정의되며 C2 architecture가 갖는 connector instance를 의미한다. C2 connector는 C2 architecture과 달리 내부에 connector instance를 1개만 가질 수 있다.

4.3 C2 커넥터 인스턴스

그림 13은 C2 connector 와 C2 connector instance 와의 관계를 보여준다. C2 connector는 arch connector를 확장하여 정의되고 C2 connector instance는 arch connector instance를 확장하여 정의된다. C2 프로파일에서는 C2 connector와 C2 connector instance만이 사용될 수 있다({substituted} 제약조건). C2 connector instance가 갖는 C2 connector type은 arch connector type을 재정의한다.

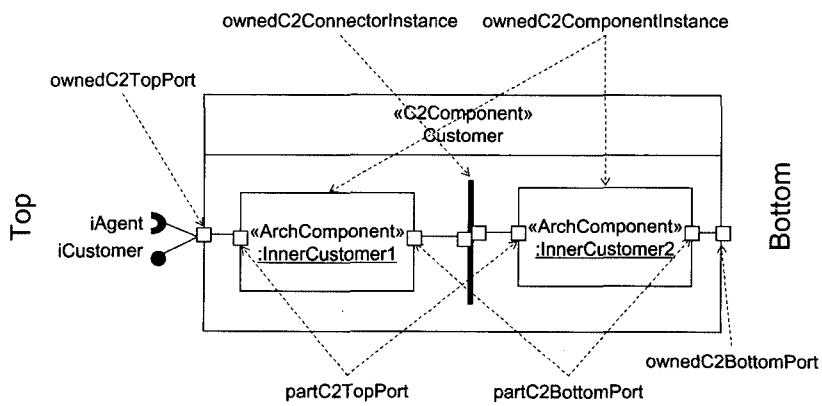

그림 10 C2 컴포넌트의 예

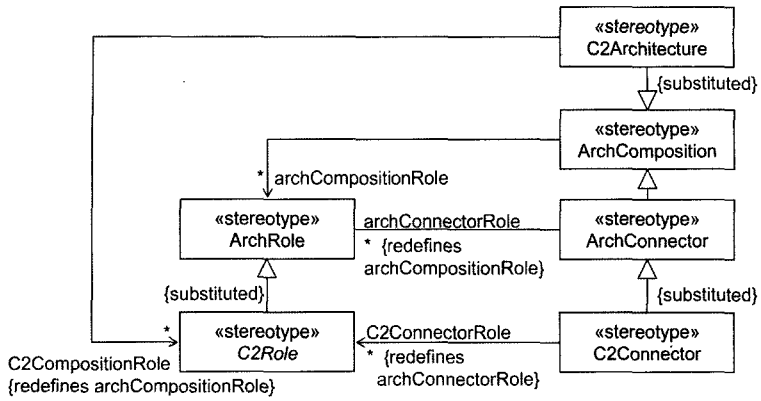

그림 11 C2 커넥터와 아키텍처

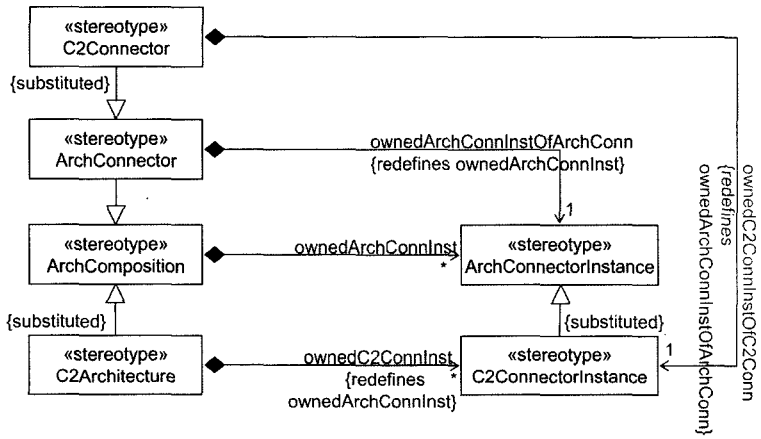

그림 12 C2 커넥터와 커넥터 인스턴스

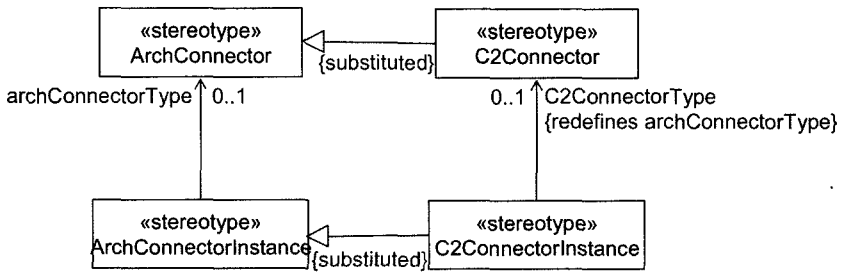

그림 13 C2 커넥터 인스턴스

4.4 C2 룰

그림 14에서 C2 role은 C2 connector가 갖는 port의 역할을 한다.

- C2 connector의 C2 role은 C2 connector instance가 생성될 때 C2 component instance의 C2 port에 binding 되거나 다른 C2 connector instance의 C2 role에 binding 된다. 이때 C2 connector의 C2 role에 binding 될 수 있는 C2 port 나 C2 role의 크기는 0 이상이며 이것은 C2 규칙 2번을 만족한다.

```
context C2ConnectorInstance
inv self.C2ConnInstPortBinding
= self.C2ConnectorType.C2ConnectorRole.C2Port
  Binding
inv self.C2ConnInstRoleBinding
= self.C2ConnectorType.C2ConnectorRole.C2Role
  Binding
inv self.C2ConnInstPortBinding->size()
```

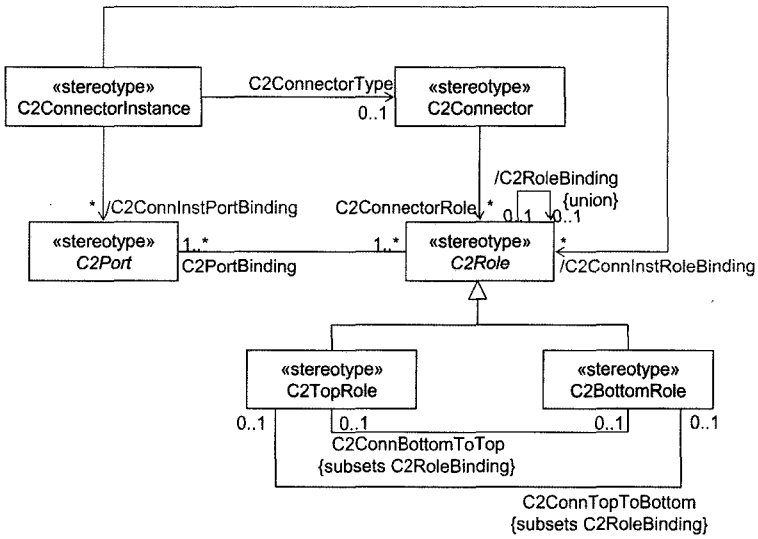

그림 14 C2 롤과 커넥터 인스턴스

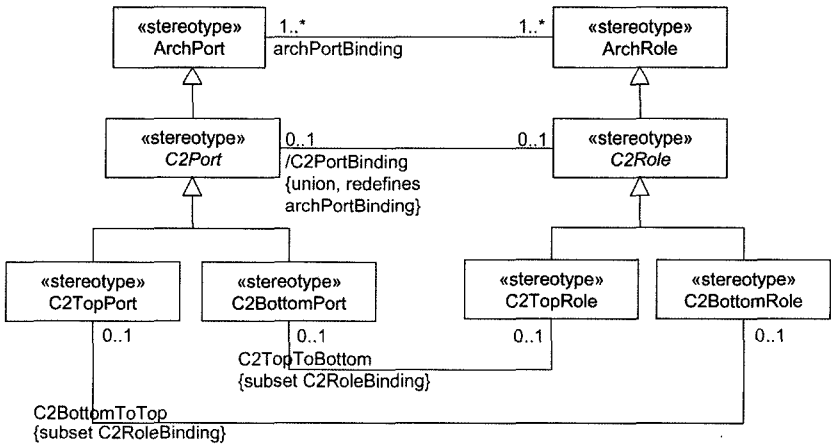

그림 15 C2 롤과 포트

```
+ self.C2ConnInstRoleBinding->size() >= 0
```

C2 top role과 C2 bottom role은 한 connector상의 top 또는 bottom port를 의미한다. C2 role의 C2 role binding은 한 connector의 role에서 연결된 다른 connector의 role을 의미한다. 따라서 C2 bottom role의 C2ConnBottomToTop은 한 connector의 bottom role에서 연결된 다른 connector의 top role을, C2 top role의 C2ConnTopToBottom은 한 connector의 top role에서 연결된 다른 connector의 bottom role을 의미한다. C2ConnBottomToTop와 C2ConnTopToBottom의 union은 C2 role binding을 형성한다. 이것은 C2 규칙 5번을 만족한다.

```
context C2Role
inv self. C2RoleBinding = self.C2ConnBottomToTop
->union(self. C2ConnTopToBottom)
```

- 그림 15는 C2 connector의 C2 role과 C2 component의 C2 port와의 관계를 보여준다. C2 role의 C2 port binding은 arch role의 arch port binding으로부터 재정의된다. C2 bottom to top은 connector의 bottom role에서 component의 top port로 연결되는 binding을 나타낸다. 이것은 C2 규칙 3번을 만족한다. C2 top to bottom은 connector의 top role에서 component의 bottom port로 연결되는 binding을 나타낸다. 이것은 C2 규칙 4번을 만족한다.
- C2 role의 C2 port binding은 C2 bottom role의 C2BottomToTop과 C2 top role의 C2TopToBottom을 union한 것과 같다.

```
context C2Role
```

```
inv self. C2PortBinding = self.C2BottomToTop
    ->union(self.C2TopToBottom)
```

그림 16은 C2 connector의 한 예로서 reservation connector을 보여준다. Reservation connector에서 top와 bottom는 c2 connector가 갖는 c2 role에 해당하고, customer와 service provider는 각 c2 role에 연결되는 port의 type을 나타낸다. Reservation c2 connector는 1개의 c2 connector instance를 갖는다.

그림 17은 c2 아키텍처의 한 예로써 restaurant reservation의 구조를 보여준다. Restaurant reservation은 reservation c2 connector의 instance인 rsvRequest와 rsvMaking을 가지며, 이 두 개의 connector instance는 guest와 agent 그리고 agent와 restaurant c2 component 사이를 연결한다. 그림 17의 각 c2component는 top과 bottom c2 port를 가지고 있으며, 이 c2 port들은 reservation connector의 c2 role 인 top 또는 bottom와 binding 관계를 갖는다.

4.5 C2 인터페이스

그림 18은 C2 interface와 C2 port와의 관계를 나타낸다. C2 top port는 topIn과 topOut interface를 가지며 C2 bottom port는 bottomIn과 bottomOut interface를 갖는다. 이 중 topIn과 bottomOut interface는 C2 notification signal을 처리하며 topOut과 bottomIn interface는 C2 request signal을 처리한다.

5. 사례 연구

본 논문에서 정의된 C2 스타일 아키텍처 프로파일을 사용하여 식당 예약(restaurant reservation) 시스템의 아키텍처 모델을 설계한다. 식당 예약 시스템을 사용하는 고객(customer)의 최종 목적은 원하는 식당을 찾아서 음식을 예약하고 배달 받는 것이다.

식당 예약 시스템을 구성하는 주요 C2 컴포넌트는 CustomerUI, Customer Representative, Reservation Agent, 그리고 Restaurant Representative이다. CustomerUI 컴포넌트는 고객이 시스템 접속할 때 사용하는 사용자 인터페이스(UI: User Interface)를 담당한다. 이 컴포넌트는 내부에 RestSelectionUI 컴포넌트와 MenuSelectionUI 컴포넌트를 포함한다. RestSelectionUI 컴포넌트는 고객이 식당을 선택할 때 사용하는 인터페이스를 담당하며, MenuSelectionUI 컴포넌트는 음식 메뉴를 선택하는 인터페이스를 제공한다. Customer Representative 컴포넌트는 식당 예약 시스템을 사용하는 고객에 대한 정보를 가지고 있다. 이 컴포넌트는 고객의 기호를 나타내는 Preference 컴포넌트, 고객의 고정 주소를 나타내는 Address 컴포넌트, 그리고 고객의 현재 위치를 표현하는 Location 컴포넌트를 포함한다. Reservation

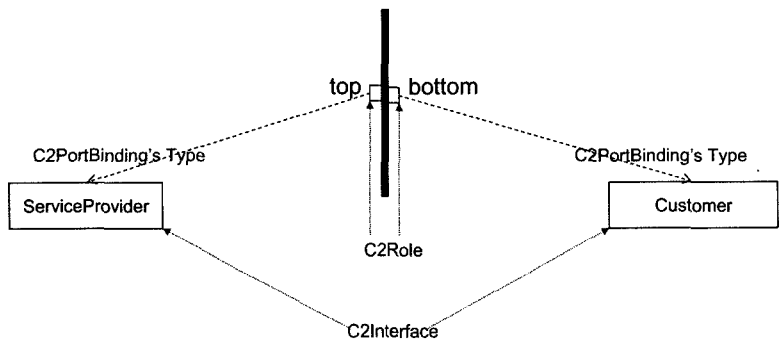

그림 16 C2 커넥터의 예

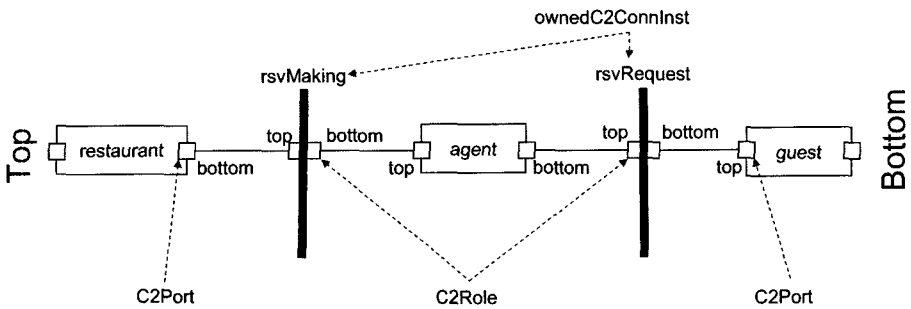

그림 17 C2 아키텍처의 예

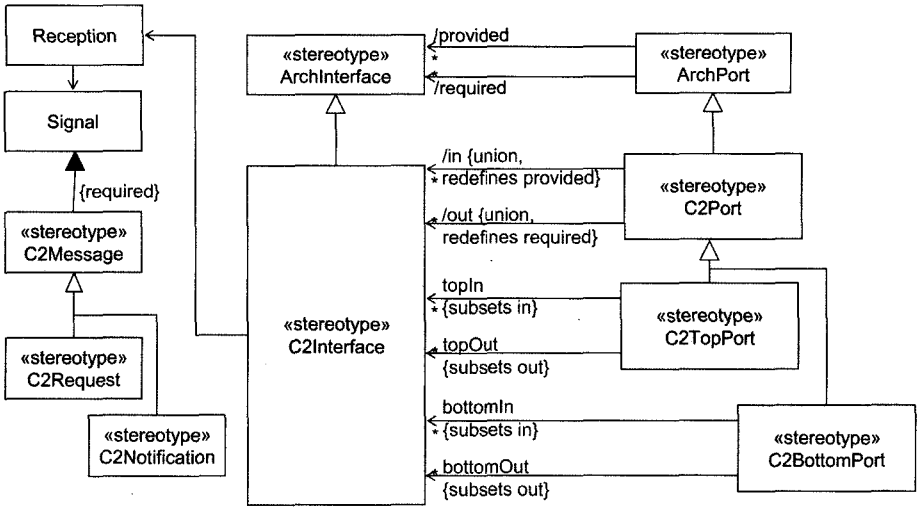

그림 18 C2 인터페이스

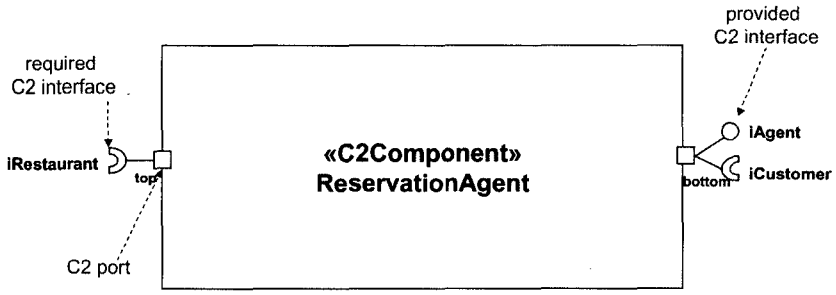

그림 19 Reservation Agent 컴포넌트의 포트

Agent 컴포넌트는 여러 고객들과 식당들 간의 예약 과정을 담당한다. 이 컴포넌트는 내부에 고객들을 관리하는 Customer Manager 컴포넌트와 식당들을 관리하는 Restaurant Manager 컴포넌트를 포함한다. LocationSP 컴포넌트는 고객의 현재 위치에 대한 정보를 알려주는 서비스 제공자(service provider)이다. PaymentSP 컴포넌트는 고객의 음식값 지불을 처리하는 서비스 제공자이다. Restaurant Representative 컴포넌트는 각 식당에 대한 정보를 갖고 있다. Product Manager 컴포넌트는 음식을 나타내고 Inventory Manager 컴포넌트는 음식 목록을 관리한다. Reservation Processor 컴포넌트는 Reservation Agent 컴포넌트가 제공하는 정보를 바탕으로 식당을 예약한다. Order Processor 컴포넌트는 고객의 주문을 받아 배달하는 일을 처리한다. 여러 고객들과 식당들은 Reservation agent를 통해 예약을 하게 되며 각 고객마다 하나의 UI를 갖는다.

5.1 C2 컴포넌트의 정의

그림 19는 Reservation Agent 컴포넌트를 black box 관점으로 보여준다. Reservation Agent 컴포넌트는 C2 provided와 required 인터페이스를 가지며 이 인터페이스들은 C2 포트를 통하여 외부에 드러난다(expose). iAgent 인터페이스는 Reservation Agent 컴포넌트가 외부에 예약 처리 기능을 제공하기 위한 인터페이스이다. iCustomer와 iRestaurant 인터페이스는 Reservation Agent 컴포넌트가 예약 처리를 하기 위해서 외부로부터 필요로 하는 인터페이스들이다. 이러한 인터페이스들은 C2 top 포트와 C2 bottom 포트로서 컴포넌트 외부에 드러난다.

C2 포트는 C2 컴포넌트의 내부 구조를 캡슐화한다. 캡슐화된 Reservation Agent 컴포넌트의 white-box 관점 내부 구조는 그림 20과 같다. C2 컴포넌트의 내부 구조는 C2 서브컴포넌트(subcomponent)와 C2 서브커넥터(subconnector)의 인스턴스들로 구성되며 자신의 외부 C2 포트에 의해서 캡슐화된다. 내부 컴포넌트와 외부 포트는 delegation 커넥터(링크)를 통하여 서로 연결되며, 내부 컴포넌트들은 assembly 커넥터(링크)를 통

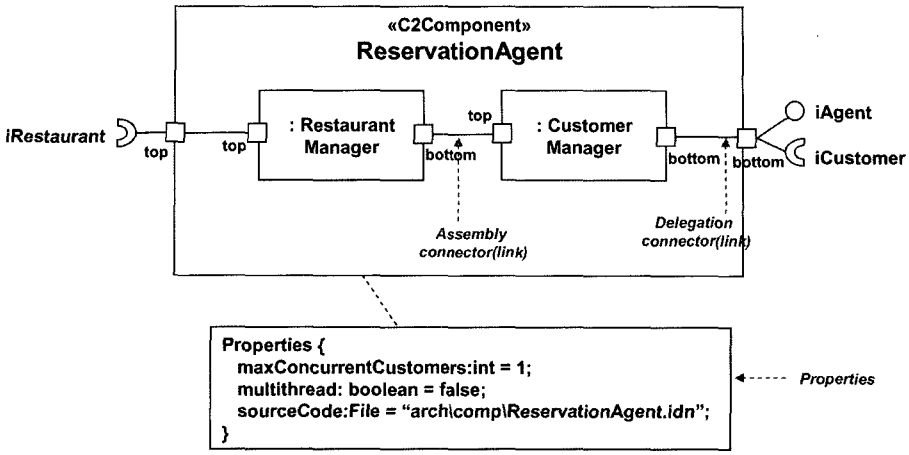

그림 20 Reservation Agent 컴포넌트의 내부 구조

하여 서로 연결된다. C2 컴포넌트들은 비구조적(extra-structural) 정보를 Property를 통하여 표현할 수 있다. 그림 20의 Reservation Agent 컴포넌트의 Property는 Reservation Agent 컴포넌트에 동시에 연결될 수 있는 최대 고객 컴포넌트의 개수와 다중 스레드(multithread) 가능 여부, 그리고 이 컴포넌트가 텍스트 형태로 저장된 파일의 위치를 알려준다.

5.2 C2 커넥터의 정의

그림 21의 Reservation Making 커넥터와 Reservation Request 커넥터는 내부에 식당 예약 과정의 처리를 위한 복잡한 프로토콜을 가지고 있다. 커넥터 내부의 프로토콜은 UML collaboration 다이어그램을 표현될 수 있으며 커넥터의 외부에는 나타나지 않는다.

5.3 C2 아키텍처의 정의

그림 22는 식당 예약 과정에 대한 최상위 수준(top-level) C2 아키텍처를 UML collaboration 다이어그램 형태로 보여준다. 그림 23은 식당 예약 시스템의 전체 합성 구조(composition structure)로써 이러한 classifier 수준의 합성 구조는 인스턴스 수준의 합성 구조로 인스턴스화(instantiation)된다. 인스턴스 수준의 합성 구조의 예는 그림 24, 25의 식당 찾기와 음식 배달 주문이다.

그림 24는 식당을 찾기를 나타내는 C2 아키텍처의 합성 구조를 보여준다. 이 인스턴스 수준의 합성 구조는 그림 23의 classifier 수준의 합성 구조로부터 인스턴화된다. 이 합성 구조는 식당 찾기 use case가 초기화될 때 동적으로 합성된다.

그림 25는 음식 배달 주문을 위한 C2 아키텍처 합성 구조이다. 그림 24의 합성 구조에 비하여 식당의 정보를 갖는 Restaurant Representative 컴포넌트와 이를 연결하는 Reservation Making 커넥터, 그리고 음식값의 지

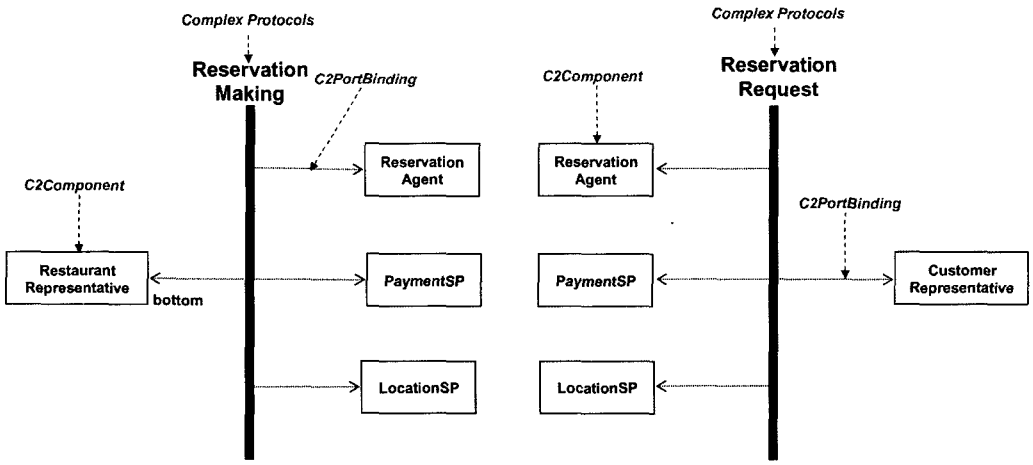

그림 21 Reservation Making 커넥터와 Reservation Request 커넥터

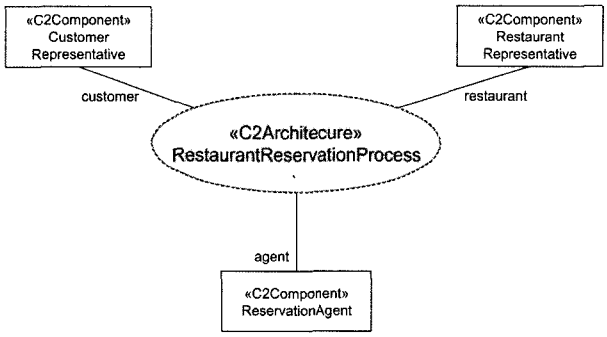

그림 22 식당 예약을 위한 최상위 수준의 C2 아키텍쳐

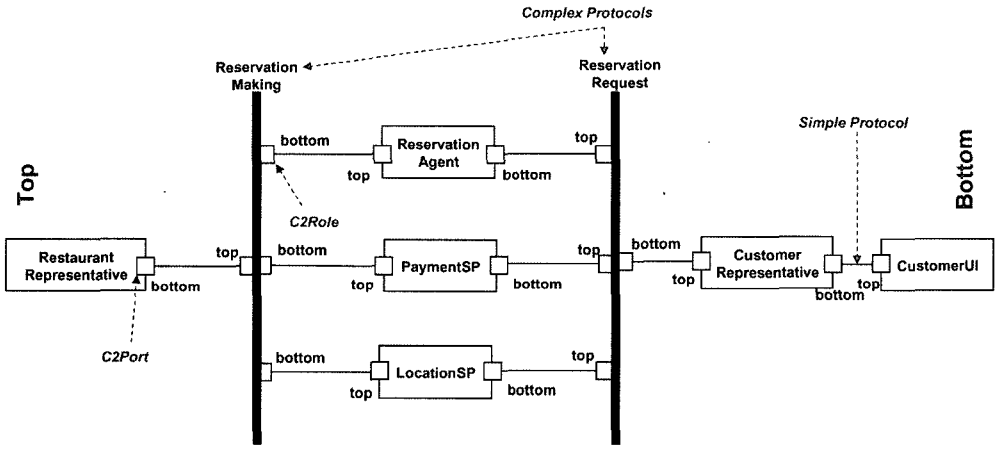

그림 23 식당 예약을 위한 classifier 수준의 전체 합성 구조

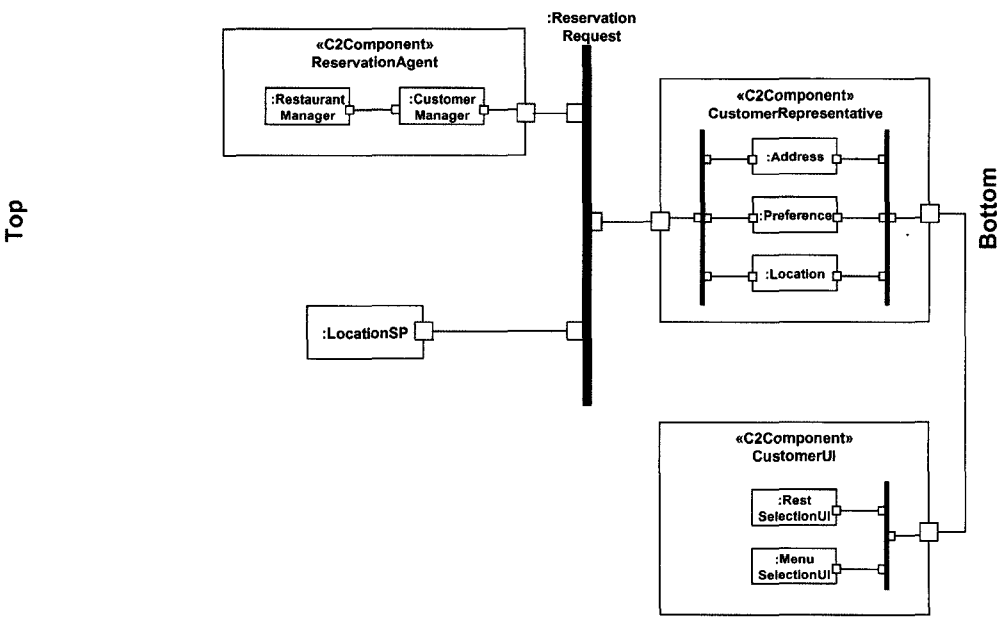

그림 24 식당 찾기를 위한 인스턴스 수준의 합성 구조

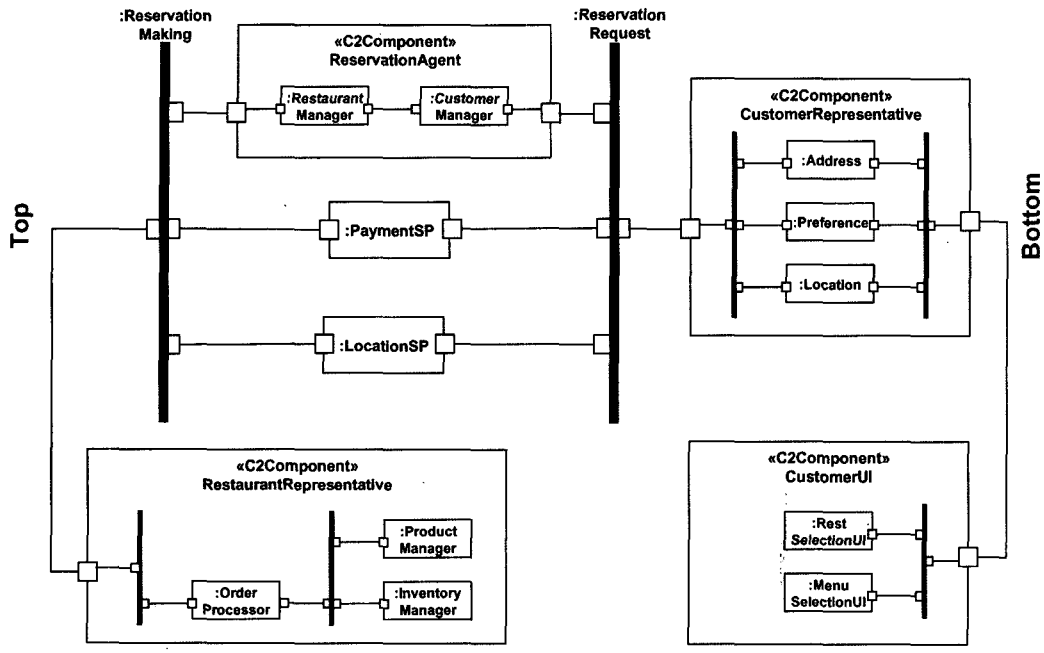

그림 25 음식 배달을 위한 인스턴스 수준의 합성 구조

불을 처리하는 PaymentSP 컴포넌트가 추가로 인스턴스화 되어 연결되었다.

6. 결론 및 향후 연구

UML을 이용하여 아키텍쳐를 모델링 하기 위한 기존 연구들은 대부분 아키텍쳐 개념의 표현이 부족했던 UML1.x을 기반으로 하고 있다. 본 논문에서는 아키텍쳐 모델링 개념이 많이 추가된 UML2.0을 확장하여 C2 아키텍쳐 모델링 언어를 정의하였으며, UML2.0을 확장하는데 필요한 제약 조건을 OCL로 정형 명세하였고, 그리고 C2 아키텍쳐 프로파일의 메타모델을 제시하였다. 정의된 C2 아키텍쳐 프로파일은 식당 예약 시스템의 C2 아키텍쳐를 설계하는데 사용되었다. 본 논문에서 정의된 C2 아키텍쳐 모델링 언어와 C2 아키텍쳐 컴포넌트, 그리고 C2 아키텍쳐 커넥터는 다음과 같은 특징을 갖는다.

- UML2.0을 lightweight 방법으로 확장하여 C2 아키텍쳐 모델링 언어를 정의하였으므로 기존 UML2.0 도구의 지원을 받을 수 있다.
- Generic ADL 프로파일을 정의하고 이를 확장하여 C2 스타일 프로파일을 정의하였다. 이러한 추상화 수준에 따른 계층적인 언어 구조는 MDA 소프트웨어 개발에 적합하다.
- UML2.0 메타모델과의 일관성을 충분히 고려하여 C2 아키텍쳐 모델링 언어를 정의하였으므로 비구조적 속성의 표현이 가능하고 C2 아키텍쳐 인터페이스와 C2 아키텍쳐 포트가 분리되어 표현되었다.
- C2 아키텍쳐 컴포넌트는 C2 아키텍쳐 포트에 의해 내부 구조(internal structure)가 캡슐화된다.
- C2 아키텍쳐 컴포넌트는 사용자가 정의할 수 있는 C2 아키텍쳐 커넥터를 통해 서로 연결된다.
- C2 아키텍쳐 컴포넌트의 하위 컴포넌트(subcomponent)들은 모두 C2 아키텍쳐 컴포넌트들로 제한되는 순환 합성 구조(recursive composite)의 C2 아키텍쳐 컴포넌트를 표현할 수 있다.
- C2 아키텍쳐 커넥터에 연결되는 개체(connectable element)를 C2 아키텍쳐 포트로 제한할 수 있다.
- C2 아키텍쳐 커넥터의 시각적 명확성이 높다
- C2 아키텍쳐 커넥터를 사용함으로써 연결의 형태에 따라 반복적으로 나타나는 연결 패턴을 정의하거나 계층적으로 상세화된 연결 구조를 표현할 수 있다.
- C2 아키텍쳐 커넥터는 복잡한 프로토콜을 내부에 가질 수 있다.

본 논문의 연구 결과를 바탕으로 향후 필요한 연구는 다음과 같다.

- 본 논문에서 정의된 C2 아키텍쳐 모델링 언어의 메타모델은 주로 C2 아키텍쳐의 구조적인 면을 다루고 있으며 C2 아키텍쳐의 행위를 명세하기에는 충분하지 않을 수 있다. C2 아키텍쳐의 행위적 측면은 구조적인 명세와의 연관성을 유지하도록 명세 되어야 하며 기존 UML 메타모델 구조물과의 일관성을 보장해야

만 기존 도구를 이용한 검증과 분석이 가능하다.

- 본 논문에서 정의된 C2 아키텍쳐 모델링 언어를 사용하여 기술된 C2 아키텍쳐 모델로부터 구현 모델로의 정제 과정을 지원하기 위한 연구가 필요하다. 정제 과정은 분할(decomposition)과 서브타이핑(subtyping)의 개념을 이용할 수 있는데 아키텍쳐 상의 개념들에 대해 이러한 방식을 적용했을 때 정제 과정에서 전체 시스템과 단위 수준의 제약사항을 만족해야 한다.
- 소프트웨어 개발 환경이 복잡해짐에 따라 다양한 컴포넌트 간 연결이 나타나고 있다. 본 논문의 C2 아키텍쳐 커넥터가 이러한 다양한 연결 방식들을 충분히 표현할 수 있는지에 대한 검증이 필요하다.
- C2스타일 이외의 다른 스타일이나 또는 영역에 종속적인 아키텍쳐 모델링 언어를 개발하고, 개발된 아키텍쳐 모델링 언어를 사용하여 다양한 영역의 실제 아키텍쳐를 설계, 분석하는 연구가 필요하다.

참 고 문 헌

[1] R.N. Taylor, N. Medvidovic, K.M. Anderson, E.J. Whitehead Jr., J.E. Robbins, K.A. Nies, P. Oreizy, and D.L. Dubrow, "A Component- and Message-Based Architectural Style for GUI Software," *IEEE Transactions on Software Engineering*, Vol. 22, No. 6, June 1996, pp. 390-406.

[2] Sunghwan Roh, Kyungrae Kim, and Taewoong Jeon, "Architecture Modeling Language based on UML2.0," *In the Proceedings of APSEC2004(Asia Pacific Software Engineering Conference) Workshop on Software Architecture and Component Technologies*, IEEE, 2004, pp.663-669.

[3] David Garlan, Andrew J. Kompanek, and Shang-Wen Cheng, "Reconciling the Needs of Architectural Description with Object-Modeling Notations," *Science of Computer Programming Volume 44*, Elsevier Press, pp. 23-49, 2002.

[4] Jorge Enrique, and Perez Martinez, "Heavyweight extensions to the UML metamodel to describe the C3 architectural style," *ACM SIGSOFT Software Engineering Notes Volume 28*, Issue 3, May 2003.

[5] Jason E. Robbins, David F. Redmiles, and David S. Rosenblum, "Integrating C2 with the Unified Modeling Language," *Proceedings of the 1997 California Software Symposium*, Irvine, CA, November 7, 1997.

[6] Jason E. Robbins, Nenad Medvidovic, David F. Redmiles, and David S. Rosenblum, "Integrating Architecture Description Languages with a Standard Design Method," *IEEE*, 1998.

[7] Mohamed Mancona Kande and Alfred Strohmeier, "Towards a UML Profile for Software Architecture Descriptions," *UML2000*, York, UK, October 2-6, 2000, LNCS, pp. 513-527, no. 1939, 2000.

[8] Nenad Medvidovic, David S. Rosenblum, Jason E. Robbins, and David F. Redmiles, "Modeling software architectures in the Unified Modeling Language," *ACM Transactions on Software Engineering and Methodology (TOSEM)* Volume 11, Issue 1, pp 2-57, January 2002.

[9] Miguel Goulao, and Fernando Brito e Abreu, "Bridging the gap between Acme and UML2.0 for CBD," *Workshop at ESEC/FSE 2003*, September 2003.

[10] Jorge Enrique Perez-Martinez and Almudena Sierra-Alonso,UML 1.4 versus UML2.0 as Languages to Describe Software Architectures, *EWSA 2004, LNCS 3047*, pp. 88-102, 2004.

[11] James Ivers, Paul Clements, David Garlan, Robert Nord, Bradley Schmerl, Jaime Rodrigo, and Oviedo Silva, *Documenting Component and Connector Views with UML2.0*, TECHNICAL REPORT CMU/SEI-2004-TR-008 ESC-TR-2004-008, April 2004.

노 성 환

1992년 3월~1997년 8월 고려대학교 컴퓨터정보학과 학사. 1997년 9월~1999년 8월 고려대학교 전산학과 석사. 2000년 3월~2005년 2월 고려대학교 전산학과 박사. 2005년 3월~2005년 9월 고려대학교 자연과학연구소 연구조교수. 2005년 10월~현재 삼성전자 반도체총괄 SOC연구소 책임연구원. 전공분야는 소프트웨어 아키텍처, 소프트웨어 테스팅, 소프트웨어 프레임웍, 객체지향 방법론

전 태 웅

1977년 3월~1981년 2월 서울대학교 계산통계학과 학사. 1981년 3월~1983년 2월 서울대학교 계산통계학과 석사. 1987년 8월~1992년 5월 Illinois Institute of Technology, Dept. Computer Science, Ph. D.. 1983년 3월~1987년 7월 금성통신 연구소 주임연구원. 1992년 9월~1995년 2월 LG산전 연구소 책임연구원. 1995년 3월~현재 고려대학교 컴퓨터정보학과 교수. 전공분야는 소프트웨어 아키텍처, 소프트웨어 테스팅, 객체지향 방법론, 컴포넌트 기반 개발

승 현 우

1977년 3월~1981년 2월 서강대학교 영문학과 학사. 1985년 9월~1988년 5월 Illinois Institute of Technology, Dept. Computer Science, 석사. 1988년 9월~1991년 12월 Illinois Institute of Technology, Dept. Computer Science, 박사. 1992년 8월~1994년 2월 한국통신 소프트웨어연구소 데이타베이스연구실 선임연구원. 1994년 3월~현재 서울여자대학교 정보통신대학 컴퓨터학부 교수. 전공분야는 데이타베이스, 데이타마이닝, 데이타웨어하우징