UML 2.0 기반의 Generic ADL 정의
(Generic ADL Definition based on UML2.0)

노 성 환 † 김 경 래 †† 전 태 웅 ††† 윤 석 진 ††††
(Sunghwan Roh) (Kyungrae Kim) (Taewoong Jeon) (Seokjin Yoon)

요 약 소프트웨어 시스템의 상위 수준 모델인 소프트웨어 아키텍처는 아키텍처 기술 언어(ADL)를 사용하여 표현된다. 하지만 ACME와 같은 대부분의 ADL들은 별도의 표기형식으로 배워야 하는 부담이 있기 때문에 아키텍처를 명세하는 언어로서 정착되지 못하였다. 반면 UML은 범용 모델링 언어로써 소프트웨어 개발의 전 과정에 일관된 표기형식과 폭넓은 지원도구들을 제공하고 있으므로 소프트웨어 개발을 위한 사실상의 표준 언어로 자리잡았다. 그러나 UML은 소프트웨어 아키텍처를 표현하도록 설계된 것은 아니기 때문에 UML을 사용하여 아키텍처를 표현하기 위해서는 UML을 확장, 변경하여야 한다. 지금까지 아키텍처 모델링에 UML을 이용하기 위한 많은 연구가 진행되어 왔다. 하지만 지금까지의 이러한 연구의 대부분은 아키텍처의 핵심 개념들의 표현이 미흡했던 UML1.x에 기반하고 있으며 곧 발표될 UML2.0에서는 이전 버전에서 미흡했던 아키텍처 모델링에 유용한 개념들이 많이 추가되었다.

본 논문에서는 UML2.0에 기반한 아키텍처 모델링 언어를 정의하였다. UML2.0을 확장하여 Generic ADL을 정의하였으며 정의된 아키텍처 모델링 언어는 식당 예약 시스템을 모델링 하는데 사용되었다.

키워드 : 소프트웨어 아키텍처, 아키텍처 기술 언어, 아키텍처 모델링 언어, UML2.0 프로파일, UML2.0 메타모델

Abstract Software architecture, which is the high level model of a software system, should be specified with ADLs (Architecture Description Languages) for its clarity and preciseness. Most of ADLs such as ACME, however, have not come into extensive use in industries since ADL users should learn a distinct notation specific to architecture. 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. UML, being a general modeling language, does not provide all concepts that are important to architecture description. UML should be extended in order to precisely model architecture. A number of researches on architecture modeling based on UML have been progressed. All of them, however, are based on the UML1.x. UML2.0 embraces much more concepts that are important to architecture modeling than UML1.x.

In this paper, we defined an architecture modeling language based on UML2.0. We defined Generic ADL by extending UML2.0 and applied the defined Generic ADL to a restaurant reservation system.

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

1. 서 론

소프트웨어 아키텍처는 크고 복잡한 시스템을 모델링하는데 있어서 매우 중요한 추상화 수준이다. 소프트웨어 아키텍처를 통해 시스템의 주요 구성 요소들과 그들의 상호 작용을 이해할 수 있게 된다. 이러한 소프트웨어 아키텍처는 아키텍처 기술 언어(ADL: Architecture Description Language)를 사용하여 기술되어야 정확하고 엄밀한 아키텍처 모델링과 아키텍처에 기반한 시스템의 분석, 정제, 검증이 가능하다. 이에 따라 아키텍처를 명시적이고 정확하게 기술할 수 있는 아키텍처 기술 언어와 ADL 지원 환경에 대한 연구가 활발히 진행되어 왔으며 현재 다양한 ADL들과 지원도구들이 소개되어

· 본 연구는 한국전자통신연구원의 지원으로 수행되었음
† 정 회 원 : 삼성전자 반도체총괄 Soc연구소 책임연구원
sunghwan.roh@samsung.com
†† 정 회 원 : LS산전중앙연구소 연구원
krkim1@lsis.biz
††† 종신회원 : 고려대학교 컴퓨터정보학과 교수
jeon@korea.ac.kr
†††† 정 회 원 : 한국전자통신연구원 임베디드 소프트웨어기술 연구단 연구원
sjyoon@etri.re.kr
논문접수 : 2005년 3월 29일
심사완료 : 2005년 12월 13일

있다.

현재까지 나와있는 ADL들을 유형 별로 요약하면, 1) 시맨틱 모델 기반 ADL, 2) 스타일 기반 ADL, 3) 도메인 기반 ADL, 4) ADL 인터체인지 언어, 5) ADL 메타언어들로 구분할 수 있다. 시맨틱 모델 기반 ADL은 아키텍처의 구조적 또는 행위적 성질들을 정형 시맨틱 모델에 의거하여 엄밀하게 기술하는 표현 수단을 제공한다. Wright[1], Rapide[2], Darwin[3] 등이 이에 속한다. 스타일 기반 ADL은 특정한 아키텍처 스타일을 따르는 아키텍처의 기술을 지원하거나, 아키텍처 스타일을 사용자가 정의할 수 있는 표현 수단을 제공한다. Unicon[4], Aesop[5], C2SADL[6] 등이 이에 속한다. 도메인 기반 ADL은 특정 응용 도메인에 속한 아키텍처들의 기술에 적합한 언어로서 해당 도메인에서 중요한 아키텍처 측면들을 잘 기술할 수 있다. 예를 들면, ROOM[7]과 MetaH[8]은 실시간 또는 내장 시스템들의 아키텍처들을 기술하는데 적합한 ADL들이다. ADL 인터체인지 언어는 다수의 ADL들을 병용하여 사용할 수 있도록 서로 다른 ADL로 표현된 아키텍처 기술(architecture description)들의 상호 변환과 공유를 지원하며 ACME[9]가 이에 속한다. ADL 메타언어는 ADL을 사용자가 정의, 변경, 확장할 수 있는 표현 수단을 제공하기 위한 언어로서 XML 스키마를 기반으로 한 xADL[10]이 이에 속한다.

이러한 ADL들은 일반적으로 두 가지 종류로 나누어 볼 수 있다. 그 중 하나는 IDL 기반으로 구성되어 있어 소프트웨어 시스템의 구축 단계를 지원하거나, 다른 하나는 정형적 표기를 사용함에 따라 명세 수준에서 아키텍처의 기능적 혹은 비기능적 속성에 대한 분석 및 검증이 가능한 경우이다[11]. 하지만 어느 경우에도 소프트웨어 시스템의 개발에 필요한 요소들 중에서 특정한 관점만을 고려하고 있는 것이 대부분이며[12], 실제 소프트웨어 개발에 필요한 요소들을 제대로 통합하지 못하고 있다.

여러 ADL들의 기본적인 핵심 개념들을 추려내고 이들을 통합하고자 하는 목적으로 만들어진 차세대 ADL로서 ACME[9]가 있다. ACME에서는 많은 ADL들이 공통적으로 가지고 있는, 아키텍처 기술에 있어 기본이 되는 개념들을 몇 가지로 정리하였다. 하지만 ACME는 이에 통합되는 각각의ADL을 별도의 표기형식으로 배워야 하는 부담이 있기 때문에 아키텍처를 표현하는 언어로서 널리 자리잡지 못하고 있다.

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 아키텍처를 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 아키텍처의 개념을 표현하도록 문법적 또는 의미적으로 설계된 것은 아니기 때문에 UML1.x를 사용하여 아키텍처를 표현하기 위해서는 UML1.x를 확장 또는 변경하여야 한다. 곧 발표될 UML2.0[13]에서는 이전 버전에서 미흡했던 아키텍처 모델링에 유용한 개념들이 많이 추가되었다. 그러나 여전히 UML2.0으로 명시적인 표현이 어려운 아키텍처의 핵심 개념들이 존재한다. 또한 UML2.0은 아키텍처 기술에 불필요하거나 관련이 적은 다른 모델링 요소들도 많이 포함하고 있다.

본 논문은 UML2.0에서 아키텍처 표현에 여전히 미진한 점들을 개선하기 위해 UML2.0의 아키텍처 기술(architecture description) 관련 부분을 특화(구체화), 확장한 연구이다. 본 논문에서는 먼저 아키텍처 모델링이 갖추어야 할 기본적인 개념들을 살펴보고, 이러한 개념들이 UML2.0에서 어느 정도 반영되었는가를 알아본다. 그리고 아키텍처 스타일에 독립적이고 모든 아키텍처 모델이 갖는 공통적 개념들의 어휘를 제공하는 Generic ADL을 UML의 확장 메커니즘을 이용하여 UML2.0 프로파일로 정의한다. Generic ADL은 그 자체만으로 아키텍처 모델링이 가능한 수준을 유지하도록 일반적인 아키텍처 모델링을 위한 핵심 개념들을 모두 포괄하도록 정의한다. 아키텍처의 개념을 충분히 표현할 수 있도록 UML2.0의 구조체를 확장하여 아키텍처의 주요 구성 요소인 컴포넌트와 커넥터, 그리고 형세(configuration) 등을 정의한다. UML2.0 메타모델과 일관성을 유지하도록 하면서, UML2.0을 확장하는데 필요한 제약 조건을 OCL을 이용하여 정형 명세하고, 아키텍처 모델링 언어 프로파일의 메타모델을 정의한다. 정의된 아키텍처 모델링 언어는 식당 예약 시스템을 모델링 하는데 사용되었다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문의 연구와 관련된 기존의 관련 연구들을 살펴본다. 3장에서는 UML2.0을 기반으로 Generic ADL를 정의하는 과정을 설명한다. 4장에서는 식당 예약 시스템 아키텍처를 사례 연구로써 모델링한다. 5장에서는 본 논문의 결론 및 향후 연구를 설명한다.

2. 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.

논문 [14]에서는 ACME로부터 UML1.x로 매핑하는 방법을 연구하였다. 이 논문은 UML1.x의 구조물을 이용하여 아키텍처의 구조물을 표현하는 여러 방법들 간의 장단점을 비교하였다. 그러나 UML1.x의 구조물들을 변경 없이(as is 방법) 아키텍처 구조물로 매핑하였기 때문에 아키텍처에 필요한 개념들이 충분히 표현되지 못하였다.

논문 [9]에서는 ADL의 주요 개념인 컴포넌트, 커넥터, 형세와 아키텍처 관점(view point)를 표현하기 위하여 UML을 어떻게 확장할 수 있는지에 대하여 논하였다. 이 논문에서는 일반적인 아키텍처의 개념들을 UML 프로파일로 표현할 수 있음을 보였으며 커넥터의 표현을 위해 UML collaboration을 확장하였다.

논문 [11]은 여러 ADL에 존재하는 공통적인 개념들을 UML 프로파일로 정의하고, 이를 확장하여 특정 ADL이나 기존 미들웨어 아키텍처를 모델링할 수 있는 언어를 정의하였다. 또한 이 논문에서는 아키텍처 커넥터가 갖는 다양한 의미를 표현하기 위해 추상 커넥터(abstract connector)와 구체 커넥터(concrete connector)의 개념을 분리하였다. 추상 커넥터는 UML association 메타클래스를 확장하여 정의되며, 구체 커넥터는 아키텍처 컴포넌트와 추상 커넥터를 다중 상속함으로써 정의된다.

논문 [15]에서는 도메인 종속적인(domain-specific) 아키텍처 프로파일을 정의하는 방법을 제안하였다. 이 논문에서는 아키텍처 개념을 나타내는 UML 프로파일을 정의한 후 이를 확장하여 특정 도메인의 아키텍처를 정의하였으며, 아키텍처의 프로파일들을 배열(arrange)하는 방법과 해석(interpret)하는 방법을 제공하였다.

위의 [14,9,11,15] 논문들은 아키텍처 개념을 표현하는 UML 프로파일을 다양한 방법으로 정의하고 있다. 그러나 이 논문들은 모두 아키텍처 개념이 부족한 UML1.x를 기반으로 하여 아키텍처를 표현하였다. UML1.x와 달리 아키텍처의 주요 개념들을 명시적으로 지원하는 컴포넌트, 커넥터, 포트 등의 여러 구조물들이 UML2.0에서 새롭게 추가되었다. 논문 [16,17]는 이러한 UML2.0을 이용하여 아키텍처 프로파일을 정의하고 있다.

논문 [16]에서는 UML2.0의 새로운 아키텍처 모델링 구조물들을 lightweight 확장 방법과 OCL을 이용하여 ACME의 아키텍처 구성 요소에 매핑하였다. ACME에서는 포트를 인터페이스의 일종으로 다루고 있지만, UML2.0에서는 포트와 인터페이스를 별개의 개념으로 구분하였다. 따라서 이 논문에서는 UML2.0인터페이스를 아키텍처 개념으로 표현하기 위하여 확장할 때 불필요한 제약이 추가되었다.

논문 [17]에서는 ACME를 UML2.0로 매핑하는 방법을 연구하였다. UML2.0의 여러 구조물들을 이용하여 아키텍처 구조물들을 표현하고 이용된 각 UML2.0 구조물의 장단점을 비교하였다. 그러나 UML2.0의 구조물들을 변경 없이 아키텍처 구조물로 매핑하였기 때문에 아키텍처의 개념을 정확하게 표현하지는 못하였다.

위 논문 [16,17]에서는 UML2.0컴포넌트 또는 클래스를 확장하여 아키텍처의 커넥터 개념을 표현하였다. 이러한 방법은 커넥터와 컴포넌트 간의 동일한 표기로 인하여 시각적 명확성(visual clarity)이 떨어진다. 그리고 아키텍처의 개념을 표현하기 위하여 스테레오타입들을 정의하는 과정에서 나타나는 제약 조건들과 UML2.0 메타모델의 제약 조건들 간의 관계가 명확하지 않다.

논문 [18-20]에서는 lightweight 방법으로 UML을 확장하여 Generic 아키텍처 모델링 언어를 정의하기 위한 본 논문의 이전 연구들을 설명하였다. 논문 [18]에서는 UML2.0에서의 아키텍처 모델링 개념들과 이를 바탕으로 확장된 아키텍처 모델링 언어의 개략적인 정의 과정을 보여주며 논문 [19,20]에서는 UML2.0으로부터 확장된 구조적인 아키텍처 모델링 언어를 보다 상세히 설명한다. 본 논문에서는 기존 연구 논문인 [18-20]를 바탕으로 UML2.0을 확장하는데 필요한 제약 조건을 OCL을 이용하여 정형 명세하였다. 또한 UML2.0 메타모델과 일관성을 유지하도록 하면서 아키텍처 모델링 언어 프로파일의 메타모델을 정의하였다.

3. UML2.0 기반의 Generic 아키텍처 모델링 언어

UML은 범용 모델링 언어로서 아키텍처 기술에 완전히 특화된 개념을 제공하지는 않는다. 그러므로 UML을 기반으로 아키텍처를 정확하게 모델링하기 위해서는 UML을 이에 맞게 확장할 필요가 있다. UML2.0에서는 이전 버전에서 미흡했던 아키텍처 모델링에 유용한 개념들이 많이 추가되었지만, 여전히 UML로 명시적인 표현이 어려운 아키텍처의 핵심 개념들이 존재한다. UML은 또한 아키텍처 기술에 불필요하거나 관련이 적은 다

른 모델링 요소들도 많이 포함하고 있다. 이에 따라 UML을 아키텍처 모델링에 적합하게 확장하는 과정에서 아키텍처 상의 개념과 어울리지 않는 모델링 요소들에 대해서는 표현의 제약을 가할 필요가 있다.

UML로 아키텍처를 표현하고자 할 때, 여러 가지 부분에서 UML의 모델링 개념들이 잘 지원하지 못하는 아키텍처 상의 개념들이 존재함으로 인한 어려움이 따른다[14,21,22]. 특히 [22]에서는 아키텍처의 정의에 따라 아키텍처 기술이 갖추어야 할 요건들을 명시하고 이들에 부합하게 객체지향 표기를 이용할 때 발생하는 의미적 차이를 설명하고 있다. 다음은 UML에서 의미적 차이를 갖는 아키텍처의 주요 개념들이다.

- 합성(composition)과 연결(connection)

  UML의 컴포넌트 다이어그램은 구현 수준의 물리적 컴포넌트들로 구성된 아키텍처 모델로서, 논리적 아키텍처의 표현에는 적합하지 않다. 반면 UML의 클래스 다이어그램은 실행 시 시스템의 논리적 구조를 클래스 또는 패키지 단위의 논리적 컴포넌트들과 이들 사이의 의존적 또는 의미적 관계들로 보여준다. 하지만 일반적인 ADL에서 컴포넌트는 그 형태와 의미에 있어서 클래스 이외의 다른 유형의 classifier로 타입을 정의해야 할 경우가 많이 있다. 그리고 ACME와 같은 ADL에서의 컴포넌트 인스턴스는 인스턴스의 정의 시 새로운 속성이나 행위적인 의미를 부가하여 가질 수 있는 것으로서 UML과 같은 객체지향 표현에서의 인스턴스화의 개념과는 다르다. 또한 이러한 컴포넌트 인스턴스 간의 연결 역시 실행 시의 개념으로서 classifier간의 의미적 연결을 나타내는 association과는 개념이 다르다. 상호작용의 성격에 따라 필요한 트랜잭션(transaction)을 정의하는 프로토콜이나 내부 구조의 표현도 ADL의 커넥터에서는 가능하지만UML의 *association*으로는 나타낼 수 없다.

- 추상화(abstraction)

  아키텍처는 이를 구성하는 요소들이 보다 구체화된 구성 요소들 또는 서브시스템들로 정제될 수 있어야 한다. 이러한 성질은 top-down 방식의 시스템 개발에서뿐만 아니라 프로덕트 라인(*product line*)에서도 필요한 개념이다. 그리고 기존의 아키텍처로부터 새로운 아키텍처를 응용이나 실행환경에 맞게 정의할 수 있어야 한다. 더 나아가 아키텍처에 새로운 컴포넌트 인스턴스가 대체되거나 더해지고, 혹은 연결이 동적으로 변경될 수 있어야 한다. 이는 내부의 구조와 속성, 제약조건을 갖는 아키텍처와 그 구성 요소가 classifier로서 정의되어야 함을 의미한다. 이를 위해 사용될 수 있는 객체지향 개념으로서 상속(inheritance), 합성(composition), 다형성(polymorphism)이 있다. 그러나 이들만으로는 아키텍처의 추상화를 표현하기에는 미흡하다.

- 상호작용 지점(interaction point)과 캡슐화(encapsulation)

  아키텍처 상의 컴포넌트 인스턴스들이 각각 서로 독립적으로 사용되도록 정의하기 위해서는 상호작용 지점을 선언해야 한다. 아키텍처 상의 인스턴스들은 서로의 내부적인 처리나 구조에 무관하게 상호작용 지점의 인터페이스에 정의된 정보만으로 상호작용할 수 있어야 한다. 또한 컴포넌트 기반 개발에서 중요한 대체 가능성(substitutibility)을 허용하기 위해서도 이것은 중요한 개념이다. 대체 가능성을 보장하기 위해서 컴포넌트 인스턴스가 제공하는 기능 외에, 요구되는 기능에 대한 정보까지 기술할 수 있어야 하며 또한 행위적인 정보도 포함해야 한다. [23]에서는 이러한 성질을 연결 가능성(connectability)과 정확성(correctness)로서 명시적으로 나타내고 있다. 또한 컴포넌트 인스턴스의 동일한 서비스가 여러 가지 상호작용에 참여할 수 있기 때문에 상호작용 지점이 classifier로서 정의되어야 하며, 인스턴스화가 가능해야 한다. 하나의 컴포넌트에 필요한 개수만큼 인스턴스화가 가능한 인터페이스 개념을 포트(port)라고 부른다. UML1.x에서는 이러한 것들을 지원하지 못한다. 하지만 UML2.0에서는 포트 개념이 새롭게 추가되었다.

- 확장성(scalability)

  아키텍처를 서로 다른 상세 수준의 단위로 표현할 수 있어야 한다. 예를 들면, 한 컴포넌트는 보다 작은 단위의 컴포넌트들과 그것들의 연결 관계로 구성되는 내부 아키텍처를 가질 수 있다. 즉, 아키텍처는 순환적 합성 구조(recursive composition)을 가질 수 있어야 한다. 이것은 객체지향 표기에서의 합성(composition) 개념과는 다른 것이다. ADL에서의 컴포넌트 간의 합성은 그러한 합성 구조가 정의된 상위 컴포넌트에 국한된 의미를 갖는다. 이와 반면, UML1.x의 객체지향 표기에서는 어떤 classifier 간의 관계가 있을 때 이는 그러한 관계가 정의된 상황(context)과는 무관하게 반드시 만족되어야 하는 전역 조건이다. 하지만 UML2.0에서는 특정 classifier의 내부에 국한된 합성 구조의 표현을 지원하기 위한 개념인structured classifier가 추가되었다.

따라서 UML을 이용하여 아키텍처를 표현하고자 할 때에는 아키텍처 표현과 UML의 모델링 개념들 사이에 이러한 차이가 존재함을 인식하고 이에 대한 대안을 제시해야 한다. 우선 아키텍처 기술을 위해 사용될 각각의 아키텍처 상의 개념들에 대응하는, 의미적으로 가장 가까운 UML 구성물들을 추려내어야 한다. 즉, 아키텍처의 개념을 이에 대응하는 UML 메타클래스에 새롭게

정의된 스테레오타입(stereotype)으로 표현하였을 때, 그 의미적 차이를 최소화할 수 있어야 한다[14]. 이와 동시에 그러한 UML 기반의 ADL로 기술된 아키텍처가 이해하기 쉽도록 가시성 높은 표기 형식으로 표현되어야 한다. 더불어 아키텍처 상의 중요한 개념을 모두 나타낼 수 있어야 한다. 그리고, 각각의 개념에 맞도록 추려낸 UML 구성물들을 확장한다. 마지막으로 아키텍처 설계 상에 반영될 구조적, 행위적 제약조건들을 정의할 수 있어야 한다[15]. 본 장에서는 UML2.0을 아키텍처 기술 언어로 확장, 정의한 프로파일과 메타모델에 대하여 설명한다.

3.1 아키텍처 기술을 위한 프로파일의 계층적 구조

아키텍처 상의 개념들은 스타일에 독립적인 공통 요소와 스타일에 따라 차별되는 요소를 모두 갖는다. 이러한 요소들을 모두 ADL의 기본 어휘, 즉 메타모델로 정의하는 것은 언어의 구조가 너무 복잡해지게 되고 특정 스타일에 의존도가 높아진다. 스타일에 독립적인 요소만을 메타모델로 정의하고 스타일에 차별적인 요소들은 모델 수준에서 아키텍처 개발자가 모두 표현하는 것도 한 방법이다. 하지만 이 경우 문제 스타일에 고유한 개념이 아키텍처 모델에 정확하게 반영되지 못하거나 모델링에 소요되는 노력이 너무 커질 수 있다.

본 연구의 접근 방식은 그림 1에서와 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 아키텍처 모델로 구축하는 것이다. 먼저, 스타일에 독립적인, 모든 아키텍처 모델이 갖는 공통적 개념들의 어휘를 제공하는 Generic ADL을 UML의 확장 메커니즘을 이용하여 UML2.0 프로파일로 정의한다. Generic ADL은 그 자체만으로 아키텍처 모델링이 가능한 수준을 유지하도록 일반적인 아키텍처 모델링을 위한 핵심 개념들을 모두 포괄하도록 정의한다. 그런 후 이를 확장하여 스타일에 종속적인 아키텍처 상의 개념을 표현하는 SS-ADL을 정의한다. SS-ADL은 스타일에 종속적인 아키텍처 설계 요소들의 어휘 사전을 제공한다. 이를 적용하여 프레임웍 아키텍처(framework architecture)를 정의할 수 있다. 이것은 특정 스타일에 공통적으로 나타날 수 있는 설계 요소간 제약조건과 시스템 구성을 표현한 것으로 문제 스타일의 아키텍처 설계 지식을 표현한 것이라 할 수 있다. 해당 스타일에 속한 특정 목표 시스템의 아키텍처는 SS-ADL을 사용하여 설계된다. 그리고 그 과정에서 프레임워크 아키텍처 모델을 참조 또는 확장함으로써 특정 응용 소프트웨어의 아키텍처를 모델링할 수 있다.

3.2 컴포넌트

그림 2는 UML2.0의 주요 메타클래스들과 이로부터 확장되어 정의된 Generic ADL의 스테레오타입들 중에서 arch component에 관련된 스테레오타입들을 보여준다. 그림 2에서 투명한 사각형은 UML2.0의 메타클래스며 어두운 사각형은 스테레오타입 클래스이다.

이전 버전의 UML에서의 컴포넌트는 논리적인 실행 단위로서의 의미와 내부 구조 및 상호작용 지점을 정의

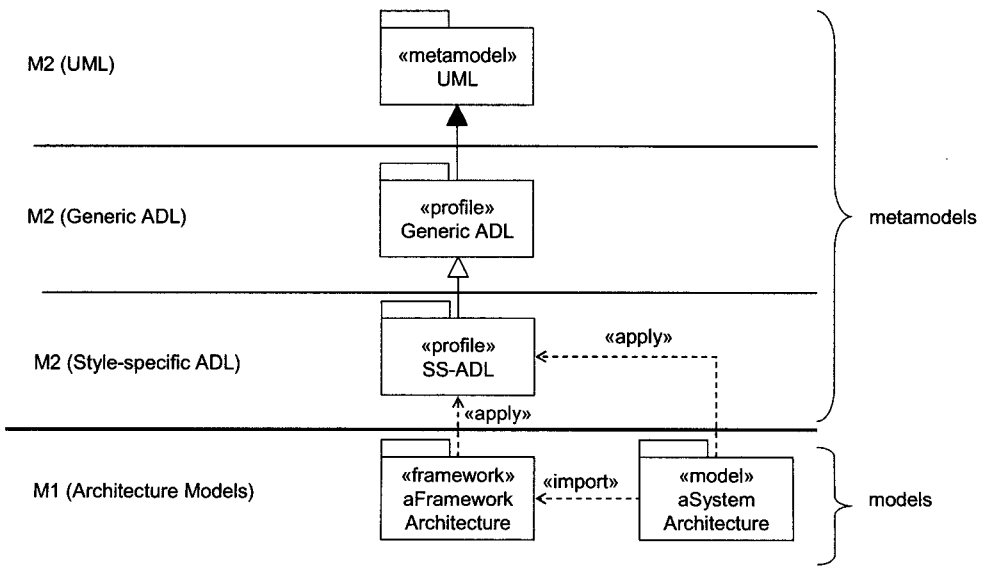

그림 1 UML2.0 기반 ADL의 메타모델 아키텍처

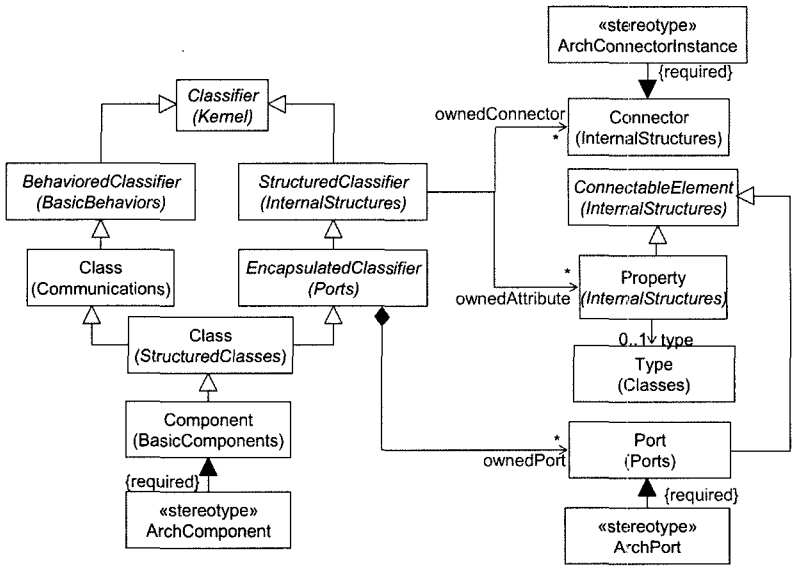

그림 2 Generic ADL의 컴포넌트

할 수 없었던 문제가 있었다. 그러나 UML2.0에서는 이러한 개념을 모두 포괄할 수 있도록 컴포넌트의 정의가 확장되었다. UML2.0 컴포넌트는 그림 2에서 보는 바와 같이 classifier로서 인스턴스화가 가능하다. Structured classifier로서 내부 구조의 표현이 가능하다. 즉, 자신이 소유하거나 참조하는 classifier의 인스턴스를 property로써, 커넥터를 자신의 feature로, 그리고 이들의 연결 형태로 구성되는 내부 구조를 가질 수 있음으로써 확장성 있는 표현이 가능하게 되었다. 또한 컴포넌트는 encapsulated classifier로서 포트를 가짐으로써 내부 구조의 캡슐화의 개념을 갖게 되었다. Behaviored classifier로서 자체의 행위 명세를 가질 수 있으며 인터페이스에 reception을 정의할 수 있도록 하여, 비동기적인 메시지 송수신이 가능한 개체로서 정의되었다.

복잡한 UML2.0 컴포넌트는 그림 3에서처럼 clasifier로서 feature를 갖는 면과 컴포넌트로서 connectable element 와 connector를 갖는 면으로 나누어 볼 수 있다. 따라서 UML2.0 컴포넌트는 아키텍쳐 상의 독립적으로 실행 가능한 합성 단위로서의 컴포넌트를 정의하는데 충분한 모델링 요소로 볼 수 있다. 그러나 UML2.0 classifier로부터 컴포넌트를 점증적으로 정의하는 과정에서 컴포넌트가 갖는 feature 중에 아키텍쳐 상의 컴포넌트가 갖는 특성과 직접적으로 관계되지 않는 것들이 있다. 예를 들면 UML2.0 컴포넌트는 자신의 내부 구조를 이루는 인스턴스들의 타입에 제약이 없으나 순환적 합성(recursive composition) 패턴을 따르는 아키텍쳐 수준의 모델에서는 내부 구성 요소로서의 인스턴스의 타입을 (서브)컴포넌트로 제한할 필요가 있다. 그림 2에서 component 메타클래스의 스테레오타입으로 표현된 arch component는 이러한 제약 사항이 추가된 Generic ADL의 컴포넌트이다.

UML2.0 component는 classifier, structured classifier, 그리고 encapsulated classifier 등으로부터 상속을 받아 만들어진다. component는 classifier로써 feature를 가지며, structured classifier로써 property와 connector를 갖고, encapsulated classifier로써 port를 갖게 되어 복잡한 구조를 형성하게 된다. 그림 3의 UML2.0 component가 가지는 복잡한 구조는 두 가지 측면에서 살펴볼 수 있다. 첫 번째 측면은 component가 갖는 feature로써의 측면이고 두 번째 측면은 connectable element와 connector로써의 측면이다. Component가 갖는 property, port 그리고 connector 클래스는 모두 component가 갖는 feature이다. 이 중 property와 port는 connectable element로써 connector를 통해 서로 연결된다.

그림 4는 그림 2의 스테레오타입들 간의 메타모델 관계를 보여준다. 그림 4에서 arch component의 owned arch port는 arch component가 갖는 외부 port를 의미하며 encapsulated classifier의 owned port로부터 재정의 된다. UML2.0 component는 owned port를 갖지 않을 수 있으나, arch component는 arch port를 최소 1개 이상 가져야 한다. Owned property는 arch component가 갖는 내부의 arch component instance들과 함께 비구조적(extra-structural) 보조 정보를 나타낸다

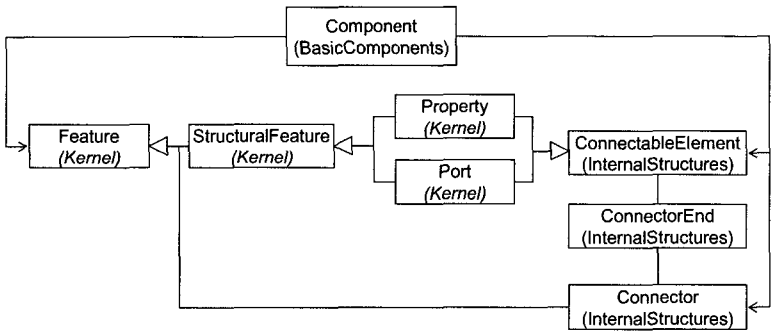

그림 3 UML2.0의 컴포넌트

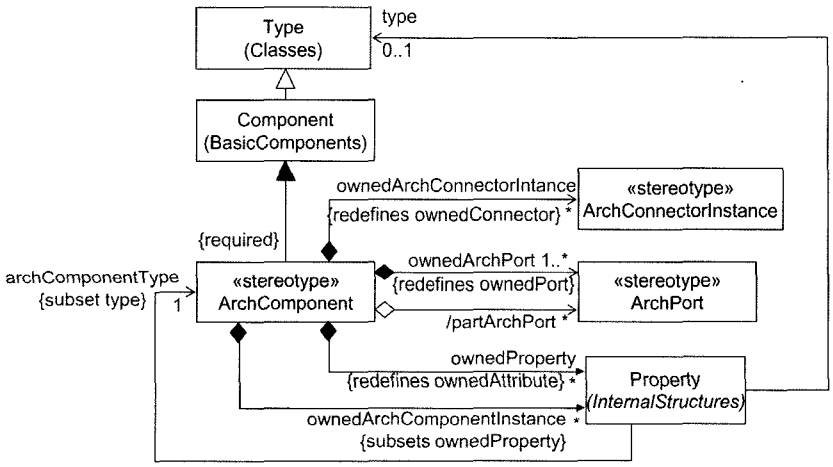

그림 4 Generic ADL의 컴포넌트의 메타모델

[9]. Arch component의 비구조적인 정보는 시스템의 특징, 요구되는 분석의 종류, 사용되는 도구(tool), 상세(detail)의 수준 등을 나타낸다[9]. Arch component가 참조하는 property가 비구조적인 보조정보를 나타낼 경우에 property의 type은 UML2.0 type이며 property가 component instance를 나타낼 경우에 property의 type은 arch component이며 UML2.0 type의 subset이다. Owned arch component instance는 arch component가 갖는 내부의 component instance들을 나타내며 1개의 arch component type을 갖는다. Arch component의 owned arch connector instance는 arch component가 갖는 내부의 connector instance들을 나타내며 structured classifier의 owned connector로부터 재정의된다.

- Arch component의 part arch port는 arch component가 갖는 내부의 port를 의미하며 내부의 owned arch component instance들이 갖는 owned arch port들로부터 유도(derived)된다.

```
derive self.partArchPort
= self.ownedArchComponentIntance.archCompo-
  nentType.ownedArchPort
```

- Arch component의 내부 arch component instance와 내부 arch connector instance는 서로 간에 part arch port를 통하여 연결된다.

```
inv self.ownedArchComponentIntance.
  archComponentType.ownedArchPort
  = self.ownedArchConnectorInstance
        .archConnectorType
  .archConnectorRole.archRoleBinding
```

그림 5는 arch component 스테레오타입을 사용하여 표현된 customer 컴포넌트 예를 보여준다. 그림 5의 customer 컴포넌트는 외부에 pC 포트를 갖고 내부에 InnerCustomer1과 InnerCustomer2의 arch component instance를 갖는다. Customer 컴포넌트는 pC와 InnerCustomer1를 연결하는 delegation connector instance와, InnerCustomer1과 InnerCustomer2를 연결하는 assembly connector instance를 가지고 있다. pC 포트는 iAgent provided interface와 iCustomer required

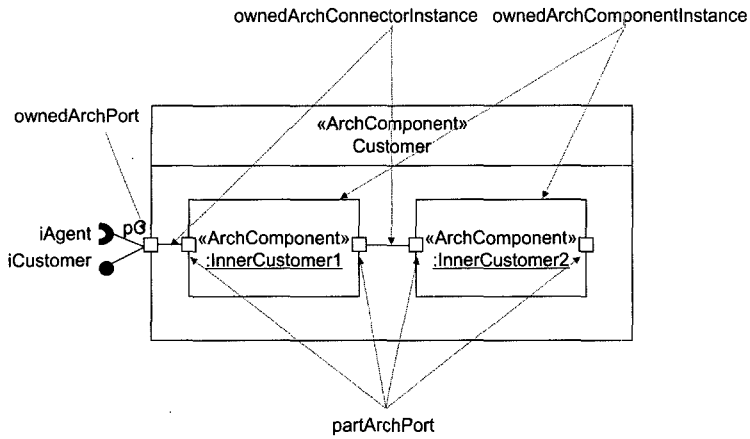

그림 5 Generic ADL의 컴포넌트의 예

interface를 갖는다.

3.3 커넥터

UML2.0의 collaboration은 structured classifer이면서 behaviored classifier로서 협동하는 개체들의 집합의 구조적 측면과 행위적 측면을 함께 표현할 수 있다. Classifier로서의 collaboration이 갖는 property인 connectable element는 협동에 관여하는 개체들의 역할을 의미하며 포트, parameter, 그리고 다른 classifier의 인스턴스들로 나타낼 수 있다. Collaboration은 또한 협동하는 개체들을 연결하는 connector들을 자신의 feature로서 가질 수 있다. Collaboration이 적용된 상황(context)을 collaboration occurrence로서 표현한다. 따라서 그림 6에서와 같이 UML2.0 collaboration을 베이스 메타클래스로 하여 Generic ADL에서 합성패턴을 나타내는 모델링 요소인 arch composition을 정의한다. Arch composition에서 합성 패턴에 참여하는 role들의 의미를 제약하여 arch role로서 정의한다. Arch role은 role binding에 의해 arch port에만 대응되도록 제약한다.

Arch composition은 특정 기능을 수행하기 위하여 함께 일하는 Generic ADL 요소들의 구조를 표현하기 위하여 사용된다.

- UML collaboration이 가질 수 있는 connectable element는 포트, prototype, parameter가 있다. 그러나 그림 6과 같이 Arch role의 base 타입은 connectable element들 중에서 port만 가능하다.

context ArchRole

inv self.base.oclIsTypeOf(Port)

그림 7, 8은 Generic ADL의 arch composition, arch

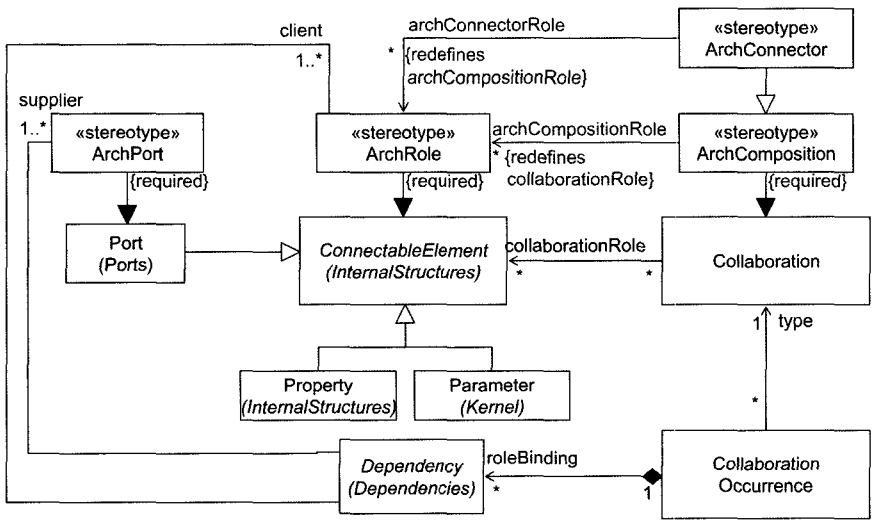

그림 6 Generic ADL의 커넥터

connector, arch role, 그리고 이과 관련된 UML2.0 메타클래스들을 보여준다. 그림 7에서 collaboration은 connector를 feature로써 가지고 있다. Collaboration을 확장하며 정의된 arch composition의 owned arch connector instance는 owned connector로부터 재정의되며 arch composition이 갖는 connector instance를 의미한다. 그림 7의 Arch connector는 Generic ADL에서 connector의 type을 나타낸다. Arch connector는 arch composition으로부터 상속을 받아 만들어지며 arch connector role은 arch composition role을 재정의하여 만들어진다. 그림 7의 Arch connector의 owned arch connector instance는 arch composition의 owned arch connector instance로부터 재정의된다. Arch connector는 Arch composition과 달리 내부에 connector instance를 1개만 가질 수 있다.

- 그림 8에서 arch connector의 arch role은 arch connector instance의 arch port에 binding 된다.

```
context ArchConnectorInstance
inv self. archConnInstBinding
    = self.archConnectorType.archConnec-
      torRole.archRoleBinding
```

그림 9는 arch connector의 한 예로서 reservation connector을 두 가지 다른 표현으로 보여준다. Reservation connector에서 클라이언트와 서버는 arch connector가 갖는 arch role에 해당하고, customer와 service provider는 각 arch role의 type을 나타낸다. Reservation arch connector는 1개의 arch connector instance를 가지며, 이를 통하여 클라이언트와 서버가 서로 연결된다.

그림 10은 arch composition의 한 예로써 restaurant reservation의 구조를 보여준다. Restaurant reservation은 reservation arch connector의 인스턴스인 rsvRequest와 rsvMaking을 가지며, 이 두 개의 커넥터 인스턴스는 guest와 agent, 그리고 agent와 restaurant component 사이를 연결한다. 그림 10의 각 arch component는 arch port를 한 개 또는 두 개 가지고 있으며, 이 arch port들은 reservation connector의 arch role 인 클라이언트 또는 서버와 바인딩(binding) 관계를 갖는다.

3.4 커넥터 인스턴스

커넥터는 컴포넌트 간 상호작용을 매개하는 역할의 개념이다. 커넥터의 기본적인 역할은 컴포넌트 간 데이타의 전송(communication)과 제어의 전송(coordination)이다. 커넥터가 가질 수 있는 또 다른 역할은 상호작용

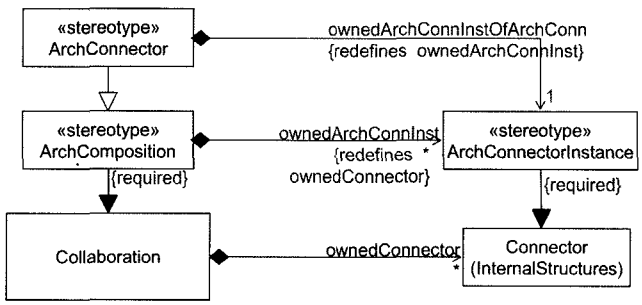

그림 7 Generic ADL의 커넥터의 메타모델

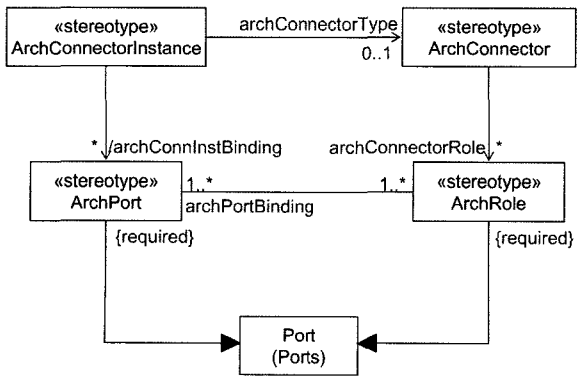

그림 8 Generic ADL의 롤

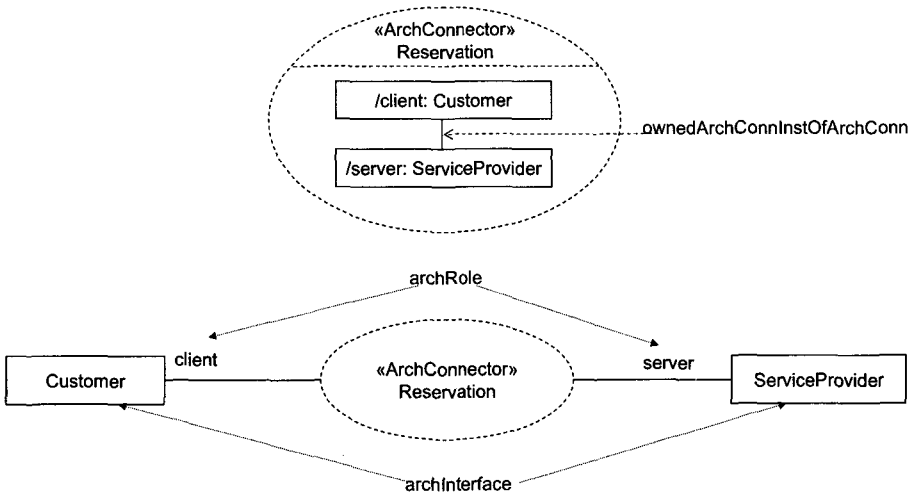

그림 9 Generic ADL의 커넥터의 예

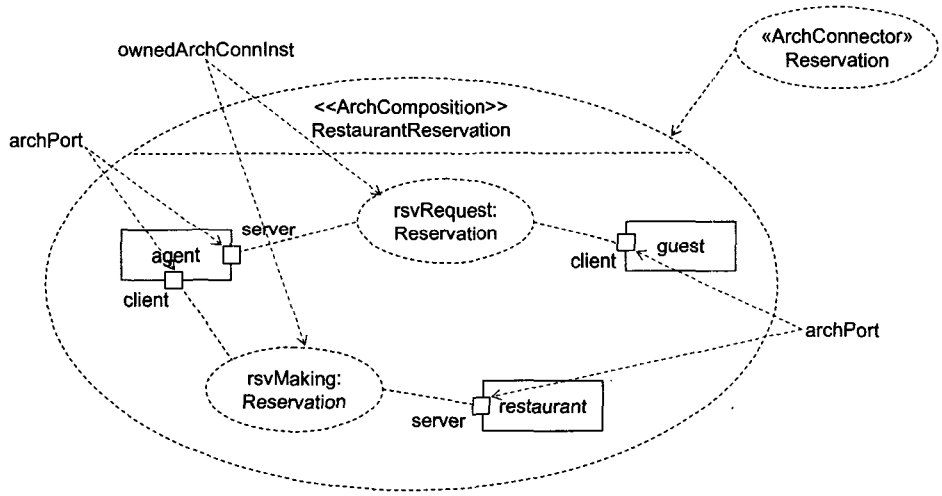

그림 10 Generic ADL의 합성의 예

할 컴포넌트들의 인터페이스가 서로 일치하지 않더라도 상호작용이 가능하도록 상호작용할 컴포넌트의 인터페이스나 프로토콜을 상대방 컴포넌트에 맞게 변환(conversion)하는 일이다. 커넥터는 또한 로드 분산(load balancing), 스케줄링, 동기화 등과 같은 상호작용의 원활함(facilitation)을 높여주는 서비스를 제공할 수 있다. 컴포넌트 간 상호작용의 가장 기본적인 방식은 프로시져 호출이나 공유 데이타 접근이다. 미들웨어 기반 개발에서는 컴포넌트 간 상호작용이 메시지 전송(message passing)이나 RPC를 통해 이루어진다[24]. 이와 같은 요건들과 함께 커넥터 모델링에 고려되어야 할 사항은 아키텍쳐 모델의 가시성이 보장되어야 한다는 것이다. 예를 들어 커넥터를 연결 기능을 갖는 컴포넌트로 보아, 컴포넌트와 동일한 표기를 사용하여 커넥터를 표현할 경우 컴포넌트와 커넥터의 구분이 모호해지는데 이러한 것은 바람직하지 않다[14].

UML2.0 커넥터는 독립적인 개체가 아닌 feature의 일종이며, typed element로서 자신의 타입을 가질 수 있는 인스턴스 수준의 개념이다. UML2.0 커넥터는 자신이 연결하는 connectable element에 실제 대응하는 인스턴스의 타입이 되는 classifier 간에 association이 존재할 때, 그 association을 자신의 타입으로 갖는다. 그림 11의 메타모델에서 connector는 두 개 이상의 connector end를 가지며 이것은 각각 connector가 연결하는 connectable element를 참조한다. 또한 connector end가 defining end로 참조하는 property는 해당 con-

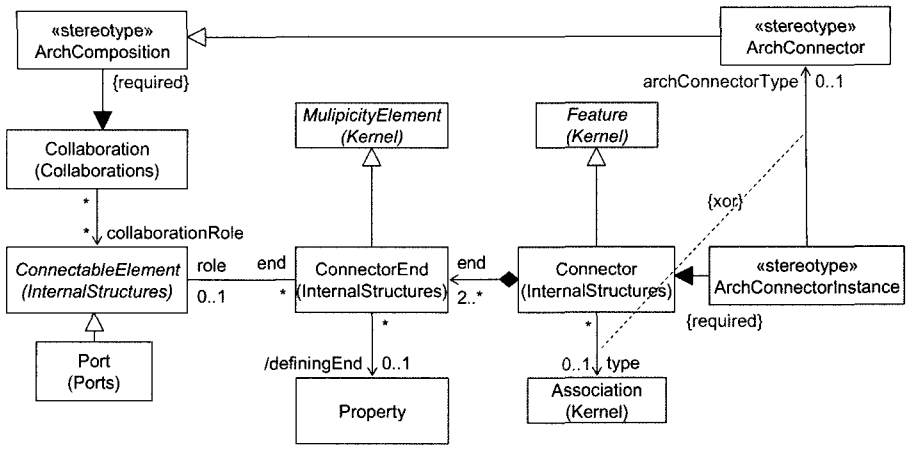

그림 11 Generic ADL의 커넥터 인스턴스

nectable element에 대응하는 association end에 정의된 property를 나타낸다. 커넥터는 delegation과 assembly의 두가지 종류로 나누어진다. Delegation 커넥터는 어떤 컴포넌트의 외부 포트와 그 컴포넌트 내부에 속한 part 컴포넌트의 외부 포트를 연결하는 경우로서 시그널을 내부, 혹은 외부로 그대로 전달하는 의미를 갖는다. Assembly 커넥터는 서로 독립적으로 구성된 컴포넌트들을 연결하는 경우이다.

위와 같이 UML2.0의 커넥터는 타입이 아닌 인스턴스 수준의 개념이다. 이에 따라 본 논문의 Generic ADL에서는 그림 11과 같이 타입으로서의 커넥터를 UML2.0 Collaboration을 확장한 arch connector스테레오타입으로 정의하였다. 그리고 UML2.0 커넥터는 커넥터 인스턴스를 표현하기 위한 arch connector instance스테레오타입을 정의하는데 사용하였다. 이렇게 정의된 arch connector instance는 자신에 연결되는 connectable element의 유형이 port로 한정되며 arch connetor를 타입으로 가질 수 있다.

그림 11은 arch connector instance의 type이 될 수 있는 arch connector와 association 간의 관계를 보여준다. Arch connector instance의 type은 arch connector 또는 association이 될 수 있으나 2개의 type이 동시에 될 수는 없다.

Arch connector를 타입으로 갖지 않고 사용된arch connector instance는 UML2.0의 connector와 동일한 의미를 갖는다. 커넥터가 delegation 용도로 사용되었거나 두개 포트 사이에 required, provided 인터페이스의 단순 연결로 사용되는 경우가 이에 해당한다. 그러나 커넥터가 복잡한 프로토콜에 의거한 상호작용, 상호작용에 참여하는 컴포넌트의 역할 (또는 타입) 명시, 상호작용 변환 기능, 또는 빈번하게 나타나는 연결 패턴의 의미를 포함하는 경우 UML2.0의 connector 개념만으로는 표현이 어렵다. Arch connector는 커넥터가 그러한 복잡한 행위적 의미와 구조를 갖는 경우를 위해, collaboration을 확장하여 정의했던 arch composition를 특화하여 정의한 타입으로서의 커넥터 개념이다. Collaboration에 참여하는 role을 나타내는 connectable element는 UML의 시퀀스 다이어그램과 같은 상호작용 모델을 나타내는데 사용되는 life line에 의해 참조될 수 있다. 따라서 커넥터의 상호작용 프로토콜을 표현하기 위한 개념적인 일관성을 제공할 수 있다. 이러한 상호작용 모델은 일시적 행위구조를 보여주는 것으로 전체적인 행위구조를 나타내는 경우는 protocol statemachine등을 이용하여 나타낼 수 있다. 커넥터를 독립적인 설계 요소로서 선언하고 커넥터의 내부적, 행위적 구조에 대한 구체적 정보 없이 컴포넌트와의 연결의 적합성을 평가하기 위해서는 커넥터의 인터페이스 또한 명시적으로 선언되어야 한다. 이를 위해 collaboration으로서의 arch role에 대응하는 개체는 반드시 컴포넌트의 포트이어야 한다. 따라서 arch role은 항상 arch port에 대응한다. 이와 같은 타입과 인스턴스 개념의 분리와 커넥터 의미의 복잡도 정도에 따른 분리를 통해 보다 가시적이고 유연한 커넥터 모델링이 가능하다.

3.5 포트

아키텍쳐 상의 컴포넌트가 갖는 상호작용 지점은 대체 가능성(substitutability)과 연결 가능성(connectability)의 측면에서 중요한 의미를 갖는다. 이러한 두 가지 성질이 만족되기 위해서는 컴포넌트가 외부로 제공하는 인터페이스 명세 외에 외부로부터 필요로 하는 인터페이스 명세가 또한 필요하다. 또한 연결되는 컴포넌트 간의 상호작용으로 주어진 작업을 정상적으로 수행하기 위해서는 상호작용 지점이 갖는 행위적 의미 또한

명세가 되어야 한다. 이러한 행위적 의미의 명세로서 사전 조건(pre-condition)과 사후 조건(post-condition)을 명세하는 방법이 있다. Arch behavior port는 포트의 행위 명세를 *protocol statemahine*으로 표현할 수 있다.

UML2.0에서는 이러한 성격의 상호작용 지점을 나타내는 개념으로서 포트를 새롭게 정의하고 있다. 그림 12에서와 같이 UML2.0 포트는 커넥터에 의해 연결될 수 있는 개념으로서 connectable element의 하나이며, *encapsulated classifier*의 *structural feature*의 하나로 정의된다. 또한 포트는 자신이 갖는 provided interface와 required interface에 의해 타입이 정해지는 typed element이다. 이러한 포트로부터 아키텍쳐 설계에 사용하기 위한 arch port를 정의한다. 한편 connector가 컴포넌트 간 연결이 아니라 컴포넌트의 내부 구조를 표현하는데 사용되는 경우 connector의 connector end에 연결되는 것은 해당 컴포넌트의 property로서의 *connectable element*의 타입에 따르는 포트가 된다. Generic ADL 프로파일의 port는 arch port만 가능하다. Generic ADL에서 connectable element는 port만 가능하며 arch port만 사용될 수 있으므로 connector end의 arch role은 role로부터 재정의 된다.

3.6 인터페이스

UML2.0에서 포트가 갖는 인터페이스는 기본적으로 외부에 공개되는 attribute와 operation을 정의할 수 있어 메소드 호출과 같은 형식의 상호작용을 지원할 수 있지만 이것 외에 비동기적으로 시그널을 처리할 수 있

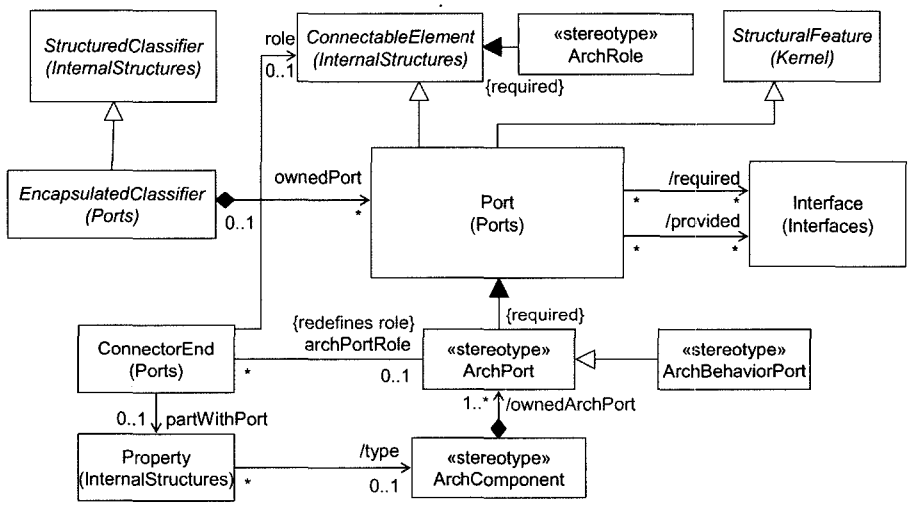

그림 12 Generic ADL의 포트

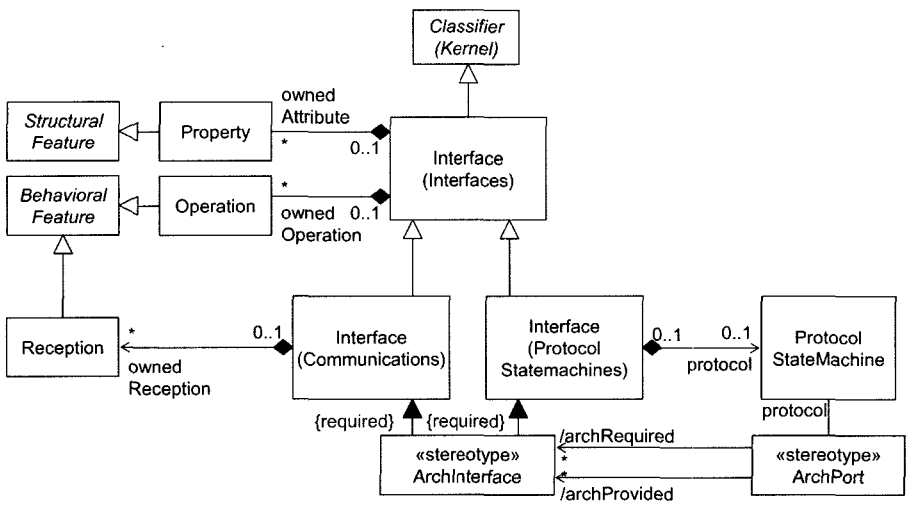

그림 13 Generic ADL의 인터페이스

도록 reception의 개념이 추가되었다. 또한 인터페이스가 갖는 특성의 행위적 의미에 대한 명세가 가능하도록 protocol statemachine을 가질 수 있도록 확장되었다. 이것은 statemachine의 일종으로서 기존의 pre, post-condition의 명세는 물론 호출이나 메시지 처리의 순서까지 명세할 수 있어 인터페이스의 행위를 보다 풍부하게 표현할 수 있다. 본 논문에서 제안하는 Generic ADL에서의 arch port가 갖는 인터페이스도 UML2.0 인터페이스의 이러한 특성을 모두 이용할 수 있도록 확장하여 arch interface로 정의한다. Arch port가 갖는 인터페이스는 arch interface만 가능하도록 의미를 한정한다. Arch port가 갖는 행위적 의미는 이러한 arch interface들의 protocol statemachine의 합성으로 표현될 수 있다. Generic ADL에서 interface는 arch interface만 사용될 수 있으며 arch port의 arch required와 arch provided는 port의 required와 provided로부터 재정의 된다.

3.7 제안된 Generic ADL의 메타모델

지금까지 UML2.0의 메타클래스들을 기반으로 하여 Generic ADL의 모델링 요소들을 어떻게 정의할 수 있는지 기술하였다. 그림 14는 그렇게 정의된 Generic ADL의 모델링 요소들을 나타내는 스테레오타입들로 구성된 UML2.0 프로파일을 보여준다.

그림 15는 앞에서 기술한 Generic ADL의 메타모델이다. 그림 15에서 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 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

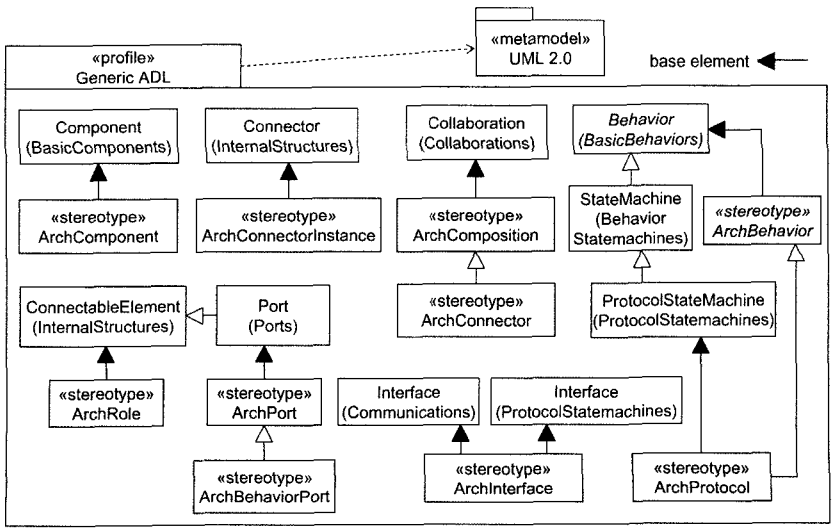

그림 14 Generic ADL에 대한 UML2.0 프로파일

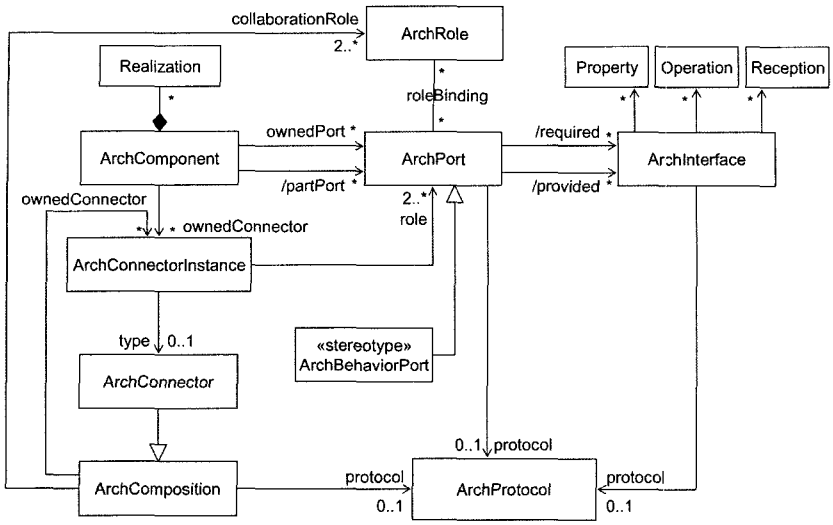

그림 15 Generic ADL의 메타모델

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.

4. 사례 연구

본 논문에서 정의된 아키텍처 모델링 언어 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 아키텍처 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 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.

4.1 아키텍처 컴포넌트의 정의

그림 16은 ArchComponent인 Reservation Agent Component의 Ports를 정의하는 예이다. ArchPort는 ArchInterface를 갖으며 ArchInterface는 Provided ArchInterface와 Required ArchInterface로 나뉜다. 따라서 ArchInterface는 ArchPort를 통해서 표현된다.

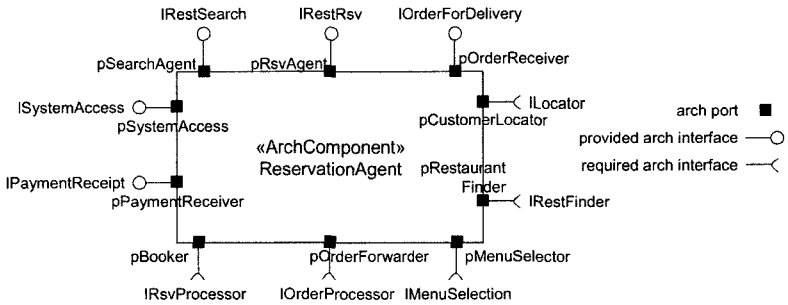

그림 16 ReservationAgent 컴포넌트의 포트 (Black-box view)

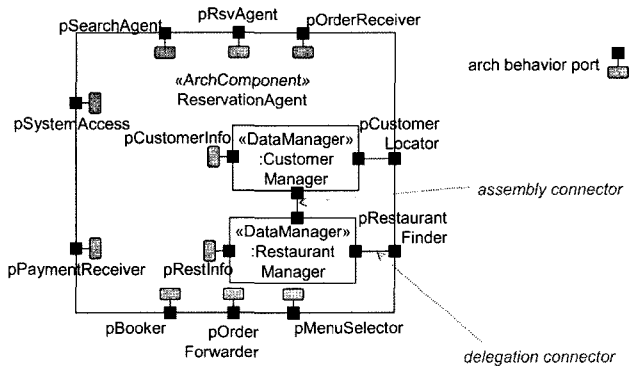

그림 17 ReservationAgent 컴포넌트의 내부 구조 (White-box view)

ArchPorts는 ArchComponent의 내부구조를 구성하는 부분들의 연결점을 나타내는 Ports이다. 위의 ArchInterface 정의에서 Implements 관계를 갖는 ArchInterface는 Provided ArchInterface이고 Uses 관계를 갖는 ArchInterface는 Required ArchInterface 이다.

그림 17은 ArchComponent의 내부구조를 구성하는 부분들과의 ArchPorts와의 연결 보여주고 있다. ReservationAgent Component는 CustomerManager와 RestaurantManager인 두 개의 내부 구조를 가지고 있다. pSearchAgent, pRsvAgent, pOrderReceiver, pMenuSelector, pOrderForwarder, pBooker, pPaymentReceiver, pSystemAccess는 ArchBehaviorPort로써 컴포넌트 자체적인 처리하는 Port로 컴포넌트 내부 부분과 연결되지 않는다. 컴포넌트의 내부구조를 구성하는 부분도 내부 Port를 가지고 있으며 외부 Port와 Delegation Connector로 연결되고 내부 Port와 Assembly Connector로 연결된다.

4.2 아키텍쳐 커넥터의 정의

그림 18은 식당 예약 시스템에서 사용되는 ArchConnectors를 정의한다. POISearch 커넥터는 Customer가

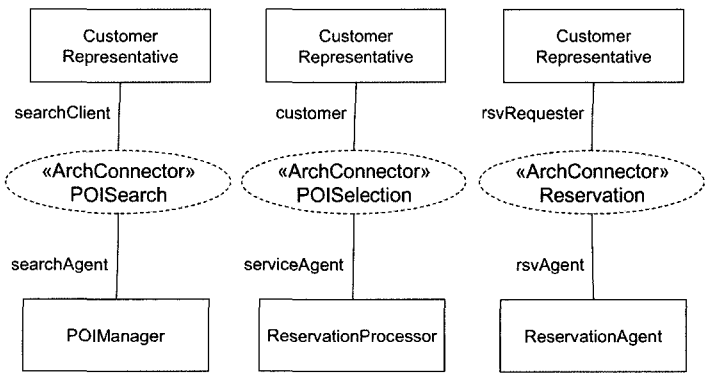

그림 18 커넥터의 정의

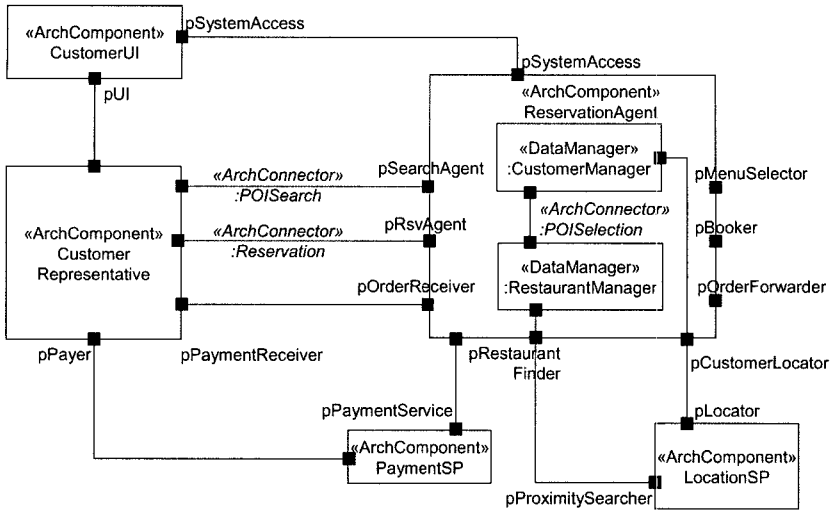

그림 19 식당 예약 시스템의 커넥터 인스턴스

Agent의 POIManager와 관심있는 위치정보인 POI (Position-Of-Interest)를 찾는 커넥터로 연결된다. POISelection 커넥터는 찾은 POI(Position-Of-Interest)를 선택하는 커넥터이다. Reservation 커넥터는 찾은 정보를 가지고 예약을 수행하는 커넥터이다.

그림 19는 위에서 정의된 ArchConnector가 식당 예약 시스템의 커넥터 인스턴스에서 어떻게 표현되는지를 나타내고 있다.

4.3 아키텍처의 정의

그림 20은 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 21, 22의 식당 찾기와 음식 배달 주문이다.

그림 21은 식당을 찾기를 나타내는 아키텍처의 합성 구조를 보여준다. 이 인스턴스 수준의 합성 구조는 그림 20의 classifier 수준의 합성 구조로부터 인스턴화된다. 이 합성 구조는 식당 찾기 use case가 초기화될 때 동적으로 합성된다.

그림 22는 음식 배달 주문을 위한 C2 아키텍처 합성 구조이다. 그림 21의 합성 구조에 비하여 식당의 정보를 갖는 Restaurant Representative 컴포넌트와 이를 연결하는 Reservation Making 커넥터, 그리고 음식값의 지불을 처리하는 PaymentSP 컴포넌트가 추가로 인스턴

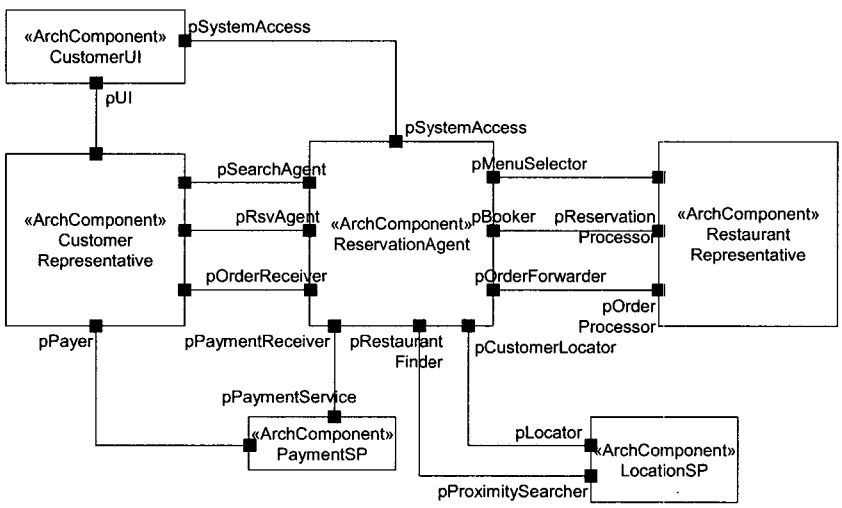

그림 20 식당 예약을 위한 classifier 수준의 전체 합성 구조

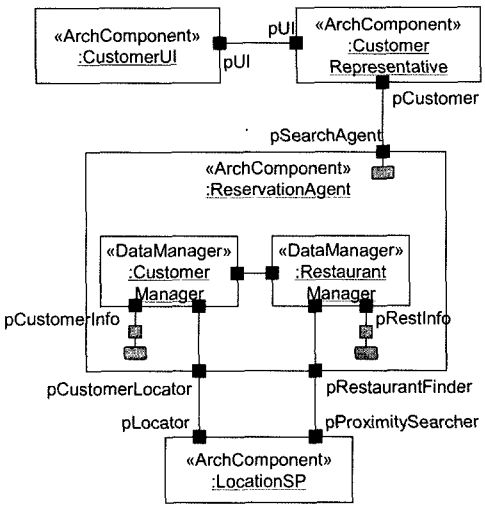

그림 21 식당 찾기를 위한 인스턴스 수준의 합성 구조

스화 되어 연결되었다.

5. 결론 및 향후 연구

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

- UML2.0을 lightweight 방법으로 확장하여 아키텍처 모델링 언어를 정의하였으므로 기존 UML2.0 도구의 지원을 받을 수 있다.
- UML2.0 메타모델과의 일관성을 충분히 고려하여 아키텍처 모델링 언어를 정의하였으므로 비구조적 속성의 표현이 가능하고 아키텍처 인터페이스와 아키텍처 포트가 분리되어 표현되었다.
- 아키텍처 컴포넌트는 UML2.0 컴포넌트를 확장하여 정의되었으며 논리적인 아키텍처 컴포넌트 수준의 개념을 표현할 수 있다.
- 아키텍처 컴포넌트는 아키텍처 포트에 의해 내부 구조(internal structure)가 캡슐화된다.
- 아키텍처 컴포넌트는 사용자가 정의할 수 있는 아키텍처 커넥터를 통해 서로 연결된다.
- 아키텍처 컴포넌트의 하위 컴포넌트(subcomponent)들은 모두 아키텍처 컴포넌트들로 제한되는 순환 합성 구조(recursive composite)의 아키텍처 컴포넌트를 표현할 수 있다.
- 아키텍처 커넥터의 타입은UML2.0 collaboration을 확장하여 정의되었으며 아키텍처 커넥터 수준의 개념을 표현할 수 있다.
- 아키텍처 커넥터의 인스턴스는 UML2.0 커넥터를 확장하여 정의되었다.
- 아키텍처 커넥터에 연결되는 개체(connectable element)를 아키텍처 포트로 제한할 수 있다.
- 아키텍처 커넥터의 시각적 명확성이 높다
- 아키텍처 커넥터를 사용함으로써 연결의 형태에 따라

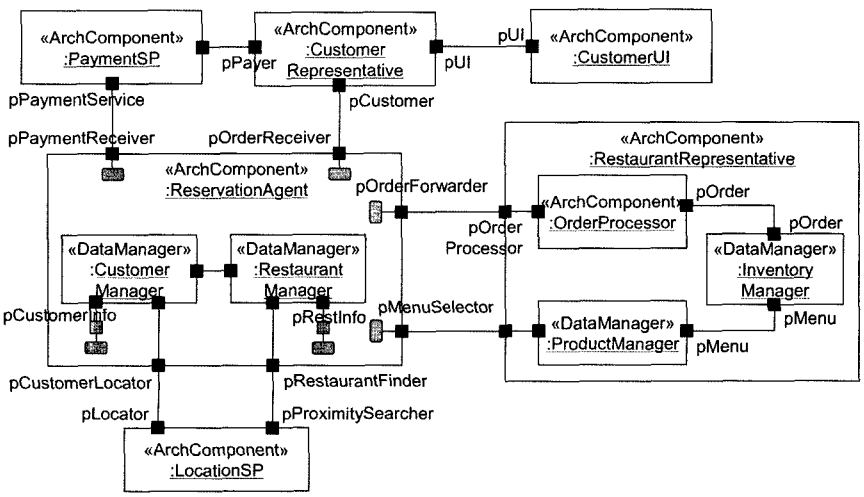

그림 22 음식 배달을 위한 인스턴스 수준의 합성 구조

반복적으로 나타나는 연결 패턴을 정의하거나 계층적으로 상세화된 연결 구조를 표현할 수 있다.

- 아키텍처 커넥터는 복잡한 프로토콜을 내부에 가질 수 있다.
- 아키텍쳐 커넥터의 상호작용이 단순한 오퍼레이션 호출이나 이벤트 전송일 경우에 아키텍쳐 커넥터 인스턴스는 UML association을 타입으로 가질 수 있다.

본 논문의 연구 결과를 바탕으로 향후 필요한 연구는 다음과 같다.

- 본 논문에서 정의된 아키텍쳐 모델링 언어의 메타모델은 주로 아키텍쳐의 구조적인 면을 다루고 있으며 아키텍쳐의 행위를 명세하기에는 충분하지 않을 수 있다. 아키텍쳐의 행위적 측면은 구조적인 명세와의 연관성을 유지하도록 명세 되어야 하며 기존 UML 메타모델 구조물과의 일관성을 보장해야만 기존 도구를 이용한 검증과 분석이 가능하다.
- 본 논문에서 정의된 아키텍쳐 모델링 언어를 사용하여 기술된 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 아키텍쳐 커넥터가 이러한 다양한 연결 방식들을 충분히 표현할 수 있는지에 대한 검증이 필요하다.
- 소프트웨어는 개발 도중이나 개발이 완료된 후에도 지속적으로 변경, 개선된다. 이에 따라 소프트웨어 아키텍처도 변경된다. 소프트웨어 아키텍처는 이를 구성하는 컴포넌트들과 커넥터들의 개별적인 변경은 물론, 이들이 합성된 시스템 형세의 변경이 필요할 수 있다. 따라서 정확하고 효율적인 아키텍처 변경을 지원하기 위하여 변경된 아키텍처 모델의 부분적, 점증적 분석 방법 등이 향후 연구 분야가 될 수 있다.

참 고 문 헌

[1] R. Allen and D. Garlan, "Beyond Definition/Use: Architectural Interconnection," Proceedings of Workshop on Interface Definition Languages, Portland, Oregon, USA, January 1994.

[2] Luckham, D. C., Augustin, L. M., Kenney, J. J., Veera, J., Bryan, D., and Mann, W., "Specification and analysis of system architecture using Rapide," IEEE Transactions on Software Engineering, Special Issue on Software Architecture, 21(4), pp. 336-355, April 1995.

[3] J. Magee and J. Kramer, "Dynamic Structure in Software Architectures," Proceedings of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering, Oct. 1996, San Francisco, CA, USA, pp. 3-14.

[4] M. Shaw, R. DeLine, D. V. Klein, T. L. Ross, D. M. Young, and G. Zelesnik, "Abstractions for Software Architecture and tools to Support Them," IEEE Transactions on Software Engineering, Vol. 21, No. 4, April 1995, pp. 314-335.

[5] Garlan, D., Allen, R., and Ockerbloom, J., "Exploiting style in architectural design environments," In Proceedings of SIGSOFT'94, The Second ACM SIGSOFT Symposium on the Foundations of Software Engineering, ACM Press, pp. 179-185, December 1994.

[6] 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.

[7] B. Selic, G. Gullekson, and P. T. Ward, Real-Time Object-Oriented Modeling, John Wiley & Sons, 1994.

[8] S. Vestal, MetaH Programmer's Manual, Version 1.09, Technical Report, Honeywell Technology Center, April 1996.

[9] David Garlan, Robert T. Monroe, and David Wile, "Acme: Architectural Description of Component-Based Systems," *Foundations of Component-Based Systems*, pp. 47-68, Cambridge University Press, 2000.

[10] E. M. Dashofy, A. van der Hoek, and R. N. Taylor, "A Highly-Extensible, XML-Based Architecture Description Language," Proceedings of the IEEE/IFIP Conference on Software Architectures (WICSA-2001), Amsterdam, Netherlands, 2001, pp. 103-112.

[11] A. Zarras, V. Issarny, C. Kloukinas, and V. K. Kguyen, "Towards a Base UML Profile for Architecture Description," *Proceedings of ICSE 2001 Workshop for Describing Software Architecture with UML*, pp. 22-26. Toronto, Ontario, Canada, IEEE Computer Society, May 2001.

[12] 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.

[13] UML 2.0 Superstructure, 3rd Revision, OMG document ad/03-04-01, Object Management Group, 2003, www.omg.org/cgi-bin/doc?ad/03-04-01.

[14] 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.
[15] P. Selonen and J. Xu, "Validating UML Models Against Architectural Profiles," *Proceedings of ESE/FSE2003*, pp. 58-67, September 1-5, 2003.
[16] Miguel Goulao, and Fernando Brito e Abreu, "Bridging the gap between Acme and UML2.0 for CBD," *Workshop at ESEC/FSE 2003*, September 2003.
[17] 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.
[18] 김경래, 노성환, 전태웅, "UML2.0에 기반한 아키텍처 모델링 언어", *Joint Workshop on Software Engineering Technology (KSEJW-2004)*, August 26-27, 2004, pp. 125-134.
[19] 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.
[20] 노성환, 김경래, 전태웅, 승현우, "아키텍처 모델링을 위한 UML2.0 프로파일", *한국정보과학회 2004 가을 학술발표논문집 II*, pp. 412-414.
[21] R.T. Monroe, A. Kompanek, R. Melton, and D. Garlan, "Architectural styles, design patterns, and objects," *IEEE Software 14(1)*, pp. 43-52, 1997.
[22] Thomas Weigert, David Garlan, John Knapman, Birger Møller-Pedersen, and Bran Selic, "Modeling of Architectures with UML," *UML 2000,* LNCS 1939, pp. 556-569, 2000.
[23] D. C. Luckham, J. Vera, and S. Meldal, "Key Concepts in Architecture Definition Languages," *Foundations of Component-Based Systems*, pp. 23-45, Cambridge University Press, 2000.
[24] N. R. Mehta, N. Medvidovic, and S. Phadke, "Towards a Taxonomy of Software Connectors," *Proceedings of International Conference Software Engineering (ICSE-2000)*, pp. 178-187, 2000.

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. 관심분야는 소프트웨어 아키텍처, 소프트웨어 테스팅, 소프트웨어 프레임웍, 객체지향 방법론

김 경 래

1996년 3월~2003년 2월 고려대학교 컴퓨터정보학과 학사. 2003년 3월~2005년 2월 고려대학교 전산학과 석사. 2005년 6월~현재 LS산전 중앙연구소 연구원. 관심분야는 소프트웨어 아키텍처, 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. 관심분야는 소프트웨어 아키텍처, 소프트웨어 테스팅, 객체지향 방법론, 컴포넌트 개발

윤 석 진

1988년 3월~1992년 8월 중앙대학교 컴퓨터공학과 졸업(학사). 1992년 9월~1994년 8월 중앙대학교 컴퓨터공학과 졸업(공학석사). 1994년 10월~현 한국전자통신연구원 선임 연구원. 관심분야는 객체지향 방법론, CASE, 컴포넌트, 재사용