DOI: 10.3745/KIPSTB.2008.15-B.5.465

유비쿼터스 컴퓨팅을 위한 온톨로지 기반의 서비스 기술 및 오버로딩 기법

이 미 연[†] · 이 정 원[††] · 박 승 수[†††] · 조 위 덕[††††]

요 약

이질성, 이동성, 가변성 등의 특징을 갖는 유비쿼터스 컴퓨팅 환경에서 사용자의 의도에 맞도록 자율적이고 동적인 서비스를 제공하기 위해서는, 실시간의 상황을 고려하여 목적 달성에 필요한 서비스들을 합성할 수 있는 기법과, 이를 가능케 하는 효과적인 서비스 기술 및 관리 방법이 전제되어야 한다. 본 연구에서는, 도메인 내의 서비스를 추출하여 서비스 온톨로지로 구조화하는 메커니즘을 제안한다. 추출된 서비스는 제안한 서비스 규격에 따라 기술되고 온톨로지 내에서 계층적인 구조를 이루게 된다. 이를 근간으로 사용자는 다양한 추상화 레벨의 서비스를 사용하여 목표를 기술할 수 있고, 서비스 오버로딩 기법을 통해 실행 시에 가장 적합한 서비스가 선택된다. 또한, 요청한 서비스가 유효하지 않은 경우에도 서비스 온톨로지를 참조하여 대체 서비스를 찾을 수 있는 합성 방법을 제안한다. 구축한 서비스 온톨로지에 대한 실험 결과, 비구조화된 서비스 리스트를 사용하는 것보다, 서비스 온톨로지를 사용함으로써 사용자의 목적 달성 성공률을 높일 수 있을 뿐만 아니라 서비스의 바인딩 시간도 감소시킬 수 있음을 보였다.

키워드 : 서비스 기술, 서비스 합성, 서비스 온톨로지, 유비쿼터스 컴퓨팅

Ontology-based Service Description and Overloading Method for Ubiquitous Computing

Meeyeon Lee[†] · Jung-Won Lee[††] · Seung Soo Park[†††] · We Duke Cho[††††]

ABSTRACT

To provide autonomous and dynamic services for users in a ubiquitous environment where heterogeneity, mobility and variability are main characteristics, an efficient service description/structuring mechanism and a service composition method are essential. Service composition can consider context in real-time and compose appropriate services. In this research, we propose a mechanism for extracting services from a specific domain and structuring them into hierarchical service ontology. Each service is described using the proposed service specification. Based on this service ontology, users can represent their goals using various abstraction levels of services, and then our service overloading method enables to invoke the most appropriate service at the execution time. Moreover, we present a method which can discovery an alternative service by referencing the service ontology, when the requested service is not available. The experimental result shows that our service ontology could improve the success probability of users' goals and reduce service binding time compare to using just an unstructured list of services.

Keywords : Service Description, Service Composition, Service Ontology, Ubiquitous Computing

1. 서 론

유비쿼터스 컴퓨팅의 중요한 이슈 중 하나는 이질적이고 이동성을 지닌 다양한 기기들의 협력적인 실행을 통해 사용자의 복잡한 목적 달성을 지원하는 것이다. 즉, 실시간에 가변적인 상황(context)을 고려하여 사용자가 의도한 바에 맞는 적절한 서비스를 연속적으로 제공할 수 있어야 한다. 이를 위해서는 환경 내의 다양한 기기(device)의 '기능(operation)'을 기술하는 기법뿐만 아니라, 서비스 제공 시의 사용자의 위치, 서비스의 유효성등과 같은 상황에 따라 적절한 서비스를 발견하고 합성할 수 있는 메커니즘이 필요하다.

서비스 합성(service composition)이란 기존의 서비스 또는 어플리케이션들을 재사용하여 새로운 복잡한 서비스를 생성하는 기법이다[1, 2]. 이는 정적과 동적 기법으로 분류되는데, 실행할 서비스와 워크플로우를 미리 정의하는 정적

※ 본 연구는 21세기 프론티어 연구개발사업의 일환으로 추진되고 있는 지식경제부의 유비쿼터스컴퓨팅및네트워크원천기반기술개발사업의 08B3-S3-10M 과제로 지원된 것임.
† 준 회 원 : 이화여자대학교 컴퓨터정보통신공학과 박사과정
†† 종신회원 : 아주대학교 전자공학부 조교수
††† 정 회 원 : 이화여자대학교 컴퓨터정보통신공학과 교수
†††† 종신회원 : 아주대학교 전자공학부 교수
논문접수 : 2008년 5월 15일
수 정 일 : 2008년 6월 13일
심사완료 : 2008년 7월 29일

서비스 합성은 유비쿼터스 컴퓨팅 실현에 부적합하므로, 합성될 서비스들이 실행 시간에 결정되는 동적 서비스 합성을 고려할 수 있어야 한다.

대표적인 기존 연구들 중에서, PICO[3, 4]는 각 서비스를 유향 그래프로 표현하고, 특정 환경 내에서 제공될 수 있는 서비스들을 하나의 그래프로 통합하여 유지/관리한다. 이 단일 그래프가 사용자의 복잡한 요구를 지원하는 기초가 된다. 실시간에 요청되는 사용자의 태스크도 유향 그래프로 표현하고, 서비스 그래프와 일치하는지 비교한다. 두 그래프간의 일대일 일치가 없으면, PICO는 서비스 가용성(availability)을 향상시키기 위해 복합 서비스를 구성하는데, 서비스 그래프에 존재하는 기본 서비스들의 입력과 출력 파라미터를 비교하여 원하는 입력으로부터 출력까지의 경로를 찾는 방식으로 이루어진다. 이 방법은 모든 서비스들에 대한 그래프를 미리 정의해야 하지만, 사용자의 태스크를 달성하기 위해 실시간에 기본 서비스들을 합성할 수 있게 함으로써 동적 특성(dynamism)을 실현한다고 할 수 있다. 하지만, PICO의 서비스 합성 기법은 사용자 태스크의 초기 입력과 최종 출력만을 고려하고 있으므로, 서비스를 제공하는 중간 과정에서 발생할 수 있는 다른 영향(효과)을 고려하지 못한다는 한계가 있다. 즉, 입력 A로부터 출력 B를 끌어낼 수 있는 서비스들의 조합을 만들 뿐, 그 과정에서 파생되는 효과나 다른 영향들을 고려하지 못하고 있다. 반면에, 우리가 제안하는 서비스 조합 및 대체 기법은 동일한 효과를 발생시킬 수 있는 서비스를 찾아줄 수 있도록 구조화된 서비스 온톨로지를 사용함으로써 위의 단점을 해결할 수 있다.

Aura[5]에서는, 각 서비스는 'supplier'에 의해 제공되며 자원을 소비한다고 정의한다. 또한, 사용자의 작업을 환경의 변화에 상관없이 지속적으로 지원하기 위해, 기기의 가용성을 고려한다. 즉, 사용자의 위치, 사용 가능한 자원 등의 변화에 따라 해당 서비스를 제공할 supplier를 선택하여 사용자의 작업 환경을 재구성한다. Aura는 주어진 환경에서 사용자의 작업에 필요한 서비스들을 제공할 수 있는 supplier들 중에서 사용자의 선호도를 최대한 만족시키고 각 서비스의 질을 극대화할 수 있는 최선의 조합을 계산하여 선택한다. 하지만, 서비스, supplier, 자원을 의미적으로 연결하거나 체계적으로 관리하는 구조를 제안하지는 못하고 있다.

Gaia[6]에서는, 'application'이라는 용어를 사용하고, 1개의 application은 5개의 컴포넌트(Adapter, Model, Controller, Presentation, Coordinator)로 구성한다. Gaia는 'application adaptation' 메커니즘을 통해 공간 이동, 기기의 실행 실패와 같은 환경의 변화에 구애 받지 않고 사용자 작업의 지속적인 실행을 지원한다.

그러나, Aura, Gaia, DIANE[7], Anamika[8], ICrafter[9] 등을 비롯한 대부분의 기존 연구들은 '서비스'와 '기기'를 명확하게 분리된 개념으로 다루지 않고 있으며, '기기' 측면에 더 초점을 맞추고 있다. 따라서, '서비스'보다는 그 서비스를 제공할 수 있는 '기기'의 분류/구조화/대체를 통해 동적 지원과 가용성 향상을 이루려고 한다. 즉, 이 연구들은 '특정 서비스의 지속적인 지원을 위해 환경 내에서 유효한 기기를 대체'하고, 우리의 방법론은 '특정 효과를 지속적으로 발생시키기 위해 서비스를 대체'한다.

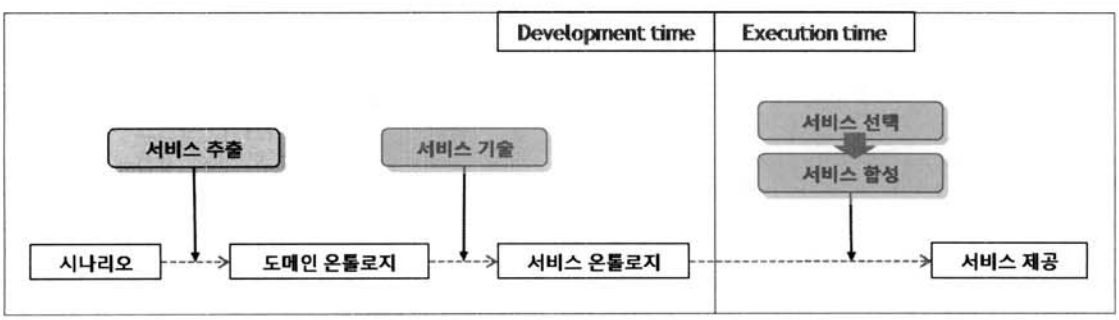
본 논문의 구성은 다음과 같다. 2장에서는, 도메인 시나리오로부터 서비스를 추출하여 제안한 규격에 맞게 기술하고, 온톨로지 형태로 구축하기 위한 프로세스를 제시한다. 3장에서는 서비스 온톨로지를 이용하여 동적 서비스 합성을 실현할 수 있는 서비스 오버로딩과 대체 서비스 발견 기법을 설명한다. 4장에서는 서비스 온톨로지의 효율성을 실험 결과를 통해 증명한다. 마지막으로 5장에서 결론을 맺고 향후 연구방향을 제시한다.

2. 서비스 기술 방법

본 연구에서는 유비쿼터스 환경 내에 존재하는 기기(device)들이 제공하는 기능(operation)을 추상화하여 '서비스'라는 개념으로 정의하였다. 즉, 서비스는 제공 가능한 기기, 위치, 실행 조건 등과 같은 기능에 대한 정보를 추상화하여 기술하고 계층적으로 구조화되는 단위로서, 사용자의 목적 달성을 위해 실행되고 유비쿼터스 환경의 이질성과 이동성을 지원할 수 있는 기초가 된다. (그림 1)은 서비스를 기술하고 기술된 서비스를 합성하여 사용하기까지의 단계를 나타내고 있는데, 크게 서비스 기술과 합성의 두 부분으로 나눌 수 있다.

2.1 서비스 추출

온톨로지 구축의 기본적인 과정은 크게 7단계를 거치게 된다. 우선, (1) 모델링할 도메인을 정하고, 온톨로지의 사용



(그림 1) 서비스 기술 및 합성의 단계

목적을 명확히 정립한다. (2) 온톨로지의 지식 공유와 재사용성이라는 근본 원칙에 따라 기존에 구축되어 있는 사례가 있는지도 고려해야 한다. (3) 해당 도메인에 등장할 수 있는 용어들(terms)을 직관적으로 열거해본 후에, (4) 일반적인 개념들은 클래스(Classes)로 표현하고 그들을 계층 구조로 정의한다. (5) 클래스로 정의된 것들의 속성이나 그들간의 관계를 나타내는 용어들이 대부분 남게 되는데, 속성(Properties)으로 정의할 수 있다. (6) 명확한 개념 표현을 위해 정의한 속성을 사용해서 각 클래스에 대한 제한 사항을 기술한다. 마지막으로, (7) 클래스에 대한 실제 사례(instances)를 추가한다[10].

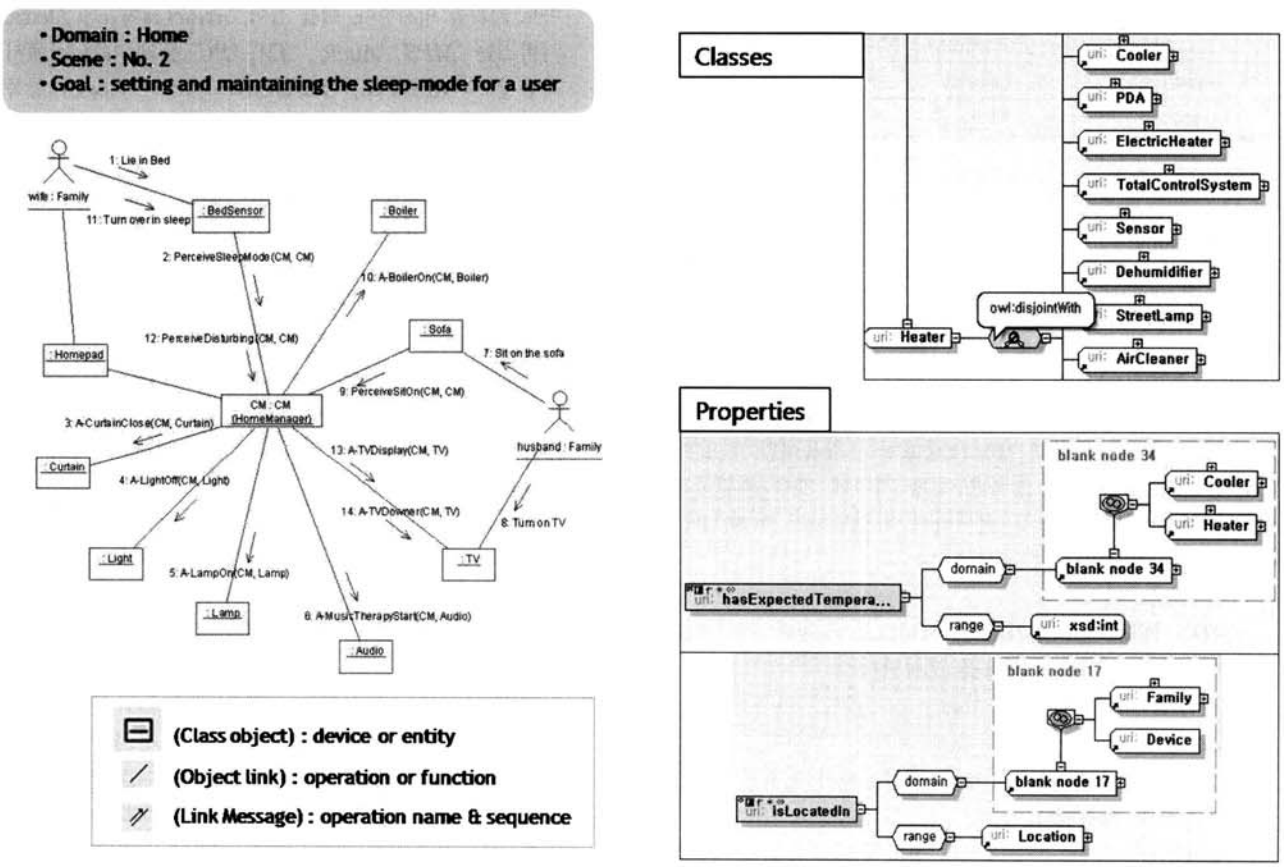
온톨로지 구축 과정의 3, 4, 5단계와 유사하게, 서비스를 기술하는데 있어서 가장 중요한 기초가 되는 기기와 그들의 기능에 대한 정보는 도메인에 대한 텍스트 시나리오로부터 추출할 수 있다. 따라서, 서비스 기술의 준비 단계에서 각 기기와 그들 간의 정확한 관계 파악을 위해 UML의 협동 다이어그램(collaboration diagram)으로 시나리오를 모델링한다. (그림 2(a))는 홈 도메인에서 발생할 수 있는 상황(situation)에 대한 다이어그램으로, 사용자의 요구에 맞는 수면 환경 조성과 유지를 목적으로 한다.

모델링 과정을 통해 생성된 각 시나리오에 대한 다이어그램으로부터 객체와 기능의 이름을 목록화하고, 그들을 바탕으로 시나리오에 직접적으로 등장하지는 않지만 도메인 내에 포함될 수 있는 것들까지 추가한다. 예를 들어, (그림 2(a))의 다이어그램으로부터 '*Curtain*', '*Light*', '*TV*'와 같은 기기들과 '*wife*', '*husband*'와 같은 사용자, '*CurtainClose*', '*LightOn*', '*BoilerOn*'과 같은 기능들을 추출할 수 있다. 뿐만 아니라, 모델에는 포함되어 있지 않지만 '*CurtainClose*'라는 기능으로부터 '*CurtainOpen*'과 같은 기능을, '*LightOn*'으로부터 '*LightOff*'와 같은 기능을 추가할 수 있고, 그 밖에 홈 도메인 내에 존재할만한 '*Air-Conditioner*', '*Audio*' 등의 기기들도 직관적으로 추가할 수 있다.

서비스를 기술하고 온톨로지화 하기 전에, 해당 도메인에 대한 몇 개의 모델로부터 추출된 개체와 그들 간의 관계는 도메인 온톨로지로 표현할 수 있다. 서비스 온톨로지의 기반이 되는 도메인 온톨로지는 W3C의 표준 온톨로지 언어인 OWL[11]을 사용하여 구축된다. 각 개체는 Classes로, 개체의 속성이나 관계는 Properties로 정의한다. (그림 2(b))는 홈 도메인에 대한 온톨로지의 일부분을 나타내고 있다.

2.2 서비스 기술

이전의 서비스 추출 단계를 통해 해당 도메인 내의 기기들에 대한 기능 목록과 도메인 온톨로지를 얻을 수 있다. 하지만, 아직까지 기능의 이름들만 목록화했을 뿐이고 각



(a) Collaboration diagram **(b) A part of domain ontology**

(그림 2) 서비스 추출 단계: 시나리오 모델링 및 도메인 온톨로지 구축

서비스의 특성이나 실행에 필요한 정보를 기술하거나 특정 구조를 표현하지는 못한다. 본 연구에서는 특정 기기들의 기능을 추상적인 개념인 '서비스'로 정의할 수 있도록 서비스 기술 규격(description specification)을 정립하고, 기술된 서비스들을 체계적으로 구조화하기 위해 온톨로지화 하는 방법을 제안한다.

2.2.1 서비스 기술 규격

서비스를 표현하기 위한 규격은 <표 1>과 같이 크게 9가지 항목을 포함한다.

〈표 1〉 서비스 기술 규격

SID	ServiceName
S.ScenarioNumber.ServiceNumber	string

SceneName	AbsLevel
integer	{0, 1, 2}

Child	Object
opt(SID_i, SID_j, ⋯, SID_n) || none	{obj_1, obj_2, ⋯, .obj_n}

TerminatingService	Precondition
	Status
none || SID_k	obj_k.*status* || none

Precondition		
Location	Priority	Input
?location	?p	{in_1, in_2, ⋯, in_n}

Effect		
Environmental/ Functional	Device	Output
Effect Parameter	obj_k.*status* || *value* || N/A	?outputdata

'SID', 'ServiceName', 'SceneName'은 각 서비스를 참조하기 위해 사용될 수 있고, 'AbsLevel'과 'Child'는 서비스들을 구조화하는데 있어서 가장 중요한 항목이다. 또한, 'Precondition'과 'Effect' 항목은 실행 시간에 가장 적합하고 유효한 서비스를 동적으로 선택하여 바인딩하기 위해 고려되어야 할 사항들이다.

- SID: 서비스의 식별자로서, 서비스가 추출된 시나리오의 번호와 서비스의 번호를 포함한다.
- ServiceName: 서비스의 이름.
- SceneName: 서비스가 추출된 상황을 나타내는 번호로서, 각 서비스가 다른 서비스와 협동하여 해결할 수 있는 상황을 의미한다.
- AbsLevel: 서비스는 추상화의 정도에 따라 3가지 레벨로 분류된다. 'AbsLevel=0'은 Atomic 서비스를, 'AbsLevel=1'은 Composite 서비스를, 'AbsLevel=2'는 Abstract 서비스를 의미한다. 이 항목을 통해 서비스들을 계층화할 수 있다.
  ① Atomic 서비스 - 가장 하위 레벨의 단말 서비스로, Child를 가질 수 없다. 서비스 추출 단계에서 목록화한 기능들은 특정 기기의 실제 동작이므로 대부분이 Atomic 서비스에 해당한다.
  ② Composite 서비스 - Atomic과 Abstract 서비스의 중간에 해당하는 서비스들은 모두 이 레벨로 분류된다. 1개 이상의 Atomic 또는 다른 Composite 서비스들로 구성되고, Atomic 서비스와 달리 여러 층을 가질 수 있다.
  ③ Abstract 서비스 - 최상위 레벨의 서비스이다. 유사한 Environmental/Functional Effect를 갖는 1개 이상의 Atomic 또는 Composite 서비스를 포함한다. 즉, 각 Abstract 서비스는 특정 Effect와 연관된 서비스들을 묶어서 하나의 그룹을 형성하는 루트가 된다.
- Child: Composite과 Abstract 서비스는 하위에 1개 이상의 서비스를 포함할 수 있는데, opt(SID_i, SID_j, ..SID_n) 형식으로 표현한다. 'opt'는 Choice (자식 서비스들 중에서 하나만 실행), AnyOrder(자식 서비스들을 순서에 상관없이 모두 실행), Sequence(자식 서비스들을 정의된 순서에 맞게 차례대로 모두 실행) 등과 같은 결합 속성을 나타낸다.
- Object: 해당 서비스를 제공할 수 있는 플랫폼 혹은 스마트 기기를 명시한다. 예를 들어, '*musicOn*'이라는 Atomic 서비스는 '*MP3*', '*Audio*', '*TV*', '*PC*' 등의 기기 상에서 작동될 수 있으므로, '*musicOn*' 서비스의 Object 항목은 {'*MP3*', '*Audio*', '*TV*', '*PC*', ⋯}로 정의할 수 있다.
- TerminatingService: 수행 중인 서비스를 종결시켜야 할 경우에 교체할 서비스를 지정한다. (현재는 기기의 상태를 'OFF'시킬 수 있는 서비스로 가정하고 있다.) 예외적으로, Abstract 서비스의 경우에는 하위에 다양한 기기의 서비스를 포함할 수 있기 때문에 종결 조건이 모호하므로 'none'으로 명시한다.
- Precondition: 서비스를 실행하기 위해 만족되어야 하는 4가지 타입의 전제 조건이다. 실행 시간에 'Precondition'과 'Effect' 항목을 검사하여 가장 적합한 서비스가 선택된다.
  - Status: 서비스를 실행할 기기의 상태를 의미한다. obj_k.*status* 형식으로 표현되는데, '*status*'는 {ON, OFF, OPEN, CLOSE, IDLE, ⋯} 중 하나이다. 예를 들어, 난방 기구의 희망 온도를 올리기 위한 '*HeaterUp*' 서비스는 해당 난방 기구가 미리 켜져 있어야, 즉 'ON' 상태이어야 실행 가능함을 명시해야 한다.
  - Location: 서비스가 제공되어야 할 특정 위치를 나타낸다. 위치 정보는 서비스를 실행할 기기를 선택할 때 가장 중요한 정보 중의 하나이다. 대부분 서비스를 제공받을 사용자의 현재 위치가 되는데, 이러한 정보는 상황 정보를 수집하거나 유비쿼터스 컴퓨팅 환경을 제어하는 모듈로부터 얻게 된다.
  - Priority: 유사한 서비스들 간의 우선 순위를 명시하

여 서비스 선택 시에 참고한다.

- Input: 서비스를 실행할 때 필요한 입력 파라미터이다. 예를 들어, '*HeaterUp*' 서비스의 경우에는 온도 레벨의 상승 정도를 의미하는 '*?temperatureValue*' 파라미터 값이 필요하다.

· Effect: 서비스를 실행한 후에 발생하는 효과를 의미한다.

- Environmental/Functional: 조도, 온도, 소음과 같은 환경적 요소에 대한 영향과 디스플레이, 스트레스 해소 등과 같은 기능적 결과를 명시한다.
- Device: 서비스 실행 후의 기기의 상태 변화를 표기한다. 서비스에 따라 'Precondition'의 'Status' 항목의 형식대로 표기될 수도 있고, 난방 기구의 희망 온도 값을 의미하는 '*?expectedTemperature*'와 같이 기기의 상태 변수값이 변경될 수도 있다.
- Output: 서비스를 실행한 후에 얻게 되는 결과물이다. 예를 들어, '*getLocation*' 서비스는 사용자의 현재 위치인 '*?currentLocation*' 정보를 반환한다.

<표 2>는 온도를 조절할 수 있는 서비스들 중에서 특히 난방 기구와 연관된 서비스들을 규격에 맞게 기술한 예이다. 8개의 서비스 중에, 4번 서비스는 (그림 2(a))의 모델로부터 추출된 "*boilerOn*" 기능을 추상화한 것으로, 규격을 통해 '*ElectricHeaterOn*', '*FanHeaterOn*' 등의 유사 기능들을 모두 포괄할 수 있는 추상화된 개념으로 정의함으로써 '서비스'라고 부를 수 있다. 5, 7, 8번은 직관적으로 추측하여 추가한 것들이고, 2, 3, 6번은 4개의 서비스를 적절하게 그룹화하는 과정에서 추가된 상위 레벨의 서비스들이다. 1번은

〈표 2〉 규격에 따라 기술된 서비스의 예

No	SID	ServiceName	SceneName	AbsLevel	Child	Object
1	S.02.021	manageTemperature	2	2	Choice(S02.022, S02.029, S02.036, S02.014)	heater(boiler, electric-heater, fan-heater)
2	S.02.022	manageHeater	2	1	Choice(S02.023, S02.026)	heater
3	S.02.023	HeaterPower	2	1	Choice(S02.024, S02.025)	heater
4	S.02.024	HeaterOn	2	0	None	heater
5	S.02.025	HeaterOff	2	0	None	heater
6	S.02.026	HeaterControl	2	1	Choice(S02.027, S02.028)	heater
7	S.02.027	HeaterUp	2	0	None	heater
8	S.02.028	HeaterDown	2	0	None	heater

No	Terminating Service	Precondition		
		Status	Location	Priority
1	None	heater	?location	?p
2	S.02.025	heater	?location	?p
3	S.02.025	heater	?location	?p
4	S.02.025	heater.OFF	?location	?p
5	S.02.025	heater.ON	?location	?p
6	S.02.025	heater.ON	?location	?p
7	S.02.025	heater.ON	?location	?p
8	S.02.025	heater.ON	?location	?p

No	Precondition	Effect		
	Input	Environmental/Functional	Device	Output
1	None	E-Temperature	N/A	none
2	() || (?sign, ?temperatureValue) || (?temperatureValue)	E-Temperature	N/A	none
3	None	E-Temperature	N/A	none
4	None	E-TemperatureUP	ON	none
5	None	E-TemperatureDOWN	OFF	none
6	(?sign, ?illuminanceValue) || (?illuminanceValue)	E-Temperature	N/A	none
7	(?illuminanceValue)	E-TemperatureUP	?expectedTemperature += ?temperatureValue	none
8	(?illuminanceValue)	E-TemperatureDOWN	?expectedTemperature -= ?temperatureValue	none

온도 조절 효과를 발생시킬 수 있는 모든 서비스들의 루트인 Abstract 서비스이다.

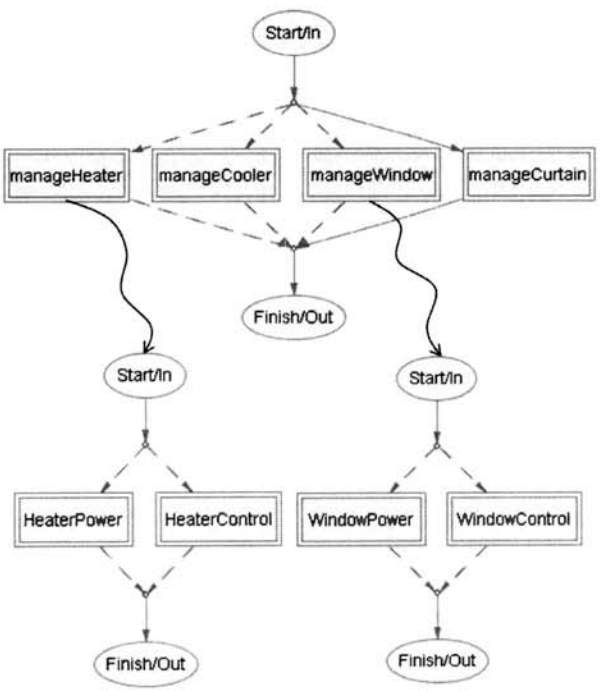
2.2.2 서비스 온톨로지

각각의 서비스는 기술 규격에 따라 W3C 표준인 OWL-S [12] 언어로 온톨로지화한다. OWL-S는 도메인 온톨로지 구축에 사용한 OWL에 기반한 언어이지만, 프로세스의 실행과 조합에 대한 정보를 기술할 수 있도록 지원하기 때문에 서비스를 표현하기에 적합하다. 또한, OWL-S는 WSDL[13], SWSL[14] 등의 다른 서비스 기술 언어에 비해 사용 구문이 간단하다. 서비스 온톨로지는 Protégé[15] 툴의 OWL-S 에디터 플러그인을 사용 하여 구축하였다.

<표 3>과 같이, OWL-S의 표현 규격 중에서 제안한 서비스 기술 규격의 특성과 맞는 항목들을 선택하여 표현하였다. 서비스 구조화에 가장 중요한 'AbsLevel'과 'Child' 항목은 프로세스의 조합을 표기하는 프로세스 모델 부분에서 표현한다. 서비스 선택과 바인딩에 중요한 역할을 하게 되는 'Object', 'Precondition', 'Effect' 항목은 OWL-S의 IOPE(Input/Output/Precondition/Effect) 표현 형식대로 기술할 수 있다.

〈표 3〉 OWL-S와 제안한 서비스 기술 규격의 매핑

Our Service Specification	OWL-S
Abstract/Composite service ('ServiceName' & 'AbsLevel')	*Composite process*
Atomic service ('ServiceName' & 'AbsLevel')	*Atomic process*
'AbsLevel' & 'Child'	*Control constructs*
'Input' ('Precondition')	*hasInput property*
'Status' & 'Location' ('Precondition') & 'Object'	*hasPrecondition property*
'Environmental/Functional' ('Effect')	*hasResult property*
'Output' ('Effect')	*hasOutput property*
The rest	*Comment*



(그림 3) 서비스 온톨로지의 일부분

(그림 3)은 서비스 온톨로지 내에서 <표 2>의 서비스들을 정의한 부분이다.

3. 서비스 합성 기법

사용자는 제공받고자 하는 복잡한 목표 서비스를 도메인 환경 내에 디플로이되어 있는 여러 개의 서비스를 사용하여 기술한다. 그리고 제안된 방법에 따라 구축된 서비스 온톨로지라는 구조를 참조함으로써 실행 시에 동적인 서비스 합성을 실현할 수 있다. 서비스 온톨로지의 강점은 크게 2가지 기법 - 서비스 오버로딩과 대체 서비스 발견 - 을 통해 부각되는데, 동적 서비스 합성의 효과를 증대시킬 수 있다.

3.1 서비스 오버로딩

사용자의 목표 달성을 위해 실제로 제공되어야 하는 서비스는 실행 시의 상황에 따라 결정되므로, 목표 서비스를 기술할 때 필요한 모든 서비스를 미리 예측하여 명확한 단말 서비스를 명시하는 것은 불가능할 뿐만 아니라 무의미하다고 할 수 있다. 하지만, 다양한 레벨의 서비스를 사용하더라도 서비스 온톨로지 내에서는 서비스들이 계층적으로 연결되어 있으므로 구조에 따라 가장 적절한 서비스가 선택될 수 있다.

서비스 오버로딩은 Abstract 또는 Composite 서비스가 명시되어 있다고 하더라도 실행 시간에 가장 적합한 Atomic 서비스를 발견하여 호출할 수 있도록 해주는 기법이다. 오버로딩이라는 기법은 원래 다양한 객체 지향 프로그래밍 언어의 특징으로, 입력과 출력 파라미터를 다르게 설정함으로써 같은 이름을 갖는 함수를 여러 개 생성할 수 있게 하는 방법이다. 이와 유사하게, Abstract 또는 Composite 서비스는 실시간의 상황 정보, 즉 Precondition과 Effect에 따라 하위 서비스들 중에서 적합한 서비스를 실행하게 된다. 예를 들어, <표 2>의 서비스들 중에서 '*manageHeater*' Composite 서비스는 다음과 같은 3가지 경우를 내포하고 있다.

① *manageHeater()*: 입력 파라미터가 없으므로, <표 2>의 서비스들 중에서 3번이 선택되고 다시 4번 또는 5번이 선택될 것이다. 기기가 'OFF' 상태일 때는 '*HeatertOn*'이, 기기가 'ON' 상태라면 '*HeaterOff*'가 선택된다.

② *manageHeater(?temperatureValue)*: 입력 파라미터가 1개이므로, <표 2>에서 직속 Child 서비스들 중에서 6번 서비스인*HeaterControl(?temperatureeValue)*이 선택된다.

③ *manageHeater(?sign, ?temperatureValue)*: 입력 파라미터가 2개이므로, <표 2>에서 직속 Child 서비스

들 중에서 6번 서비스인*HeaterControl(?sign, ?temperatureValue)*이 선택된다.

다시 말해서, *'manageHeater'*라는 동일한 서비스가 명시되어 있어도, 최종적으로 실행되는 서비스는 상황에 따라 달라진다. 마찬가지로, <표 2>의 *'HeaterControl'* Composite 서비스는 다음의 2가지 경우를 내포한다.

① *HeaterControl(?temperatureValue)*: 입력 파라미터가 1개이므로, Child인 7번과 8번 서비스 중에서 선택될 것이다. *'?temperatureValue'*는 절대값으로, 희망 온도값을 의미한다. 즉, 현재의 온도값이 *'?temperatureeValue'* 보다 높으면 *'HeaterDown(?currentTemperature - ?temperatureValue)'*가 선택되고, 낮으면 *'HeaterUp(?temperatureValue - ?currentTemperature)'*가 선택된다.

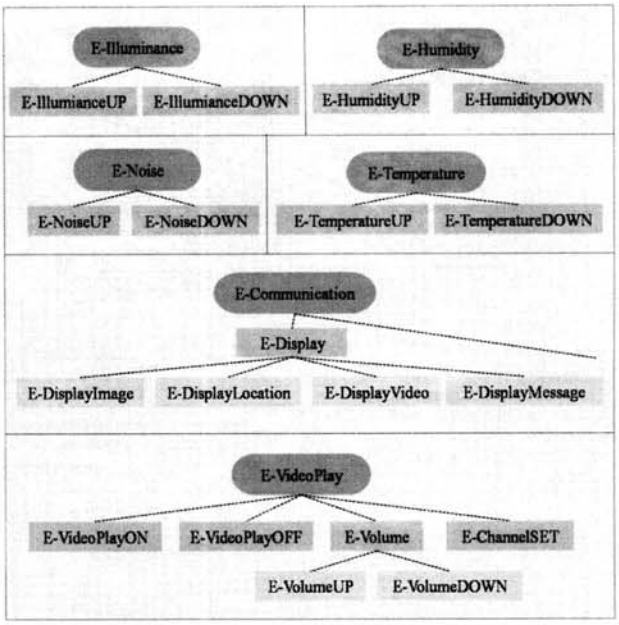
② *HeaterControl(?sign, ?temperatureValue)*: 입력 파라미터가 2개이므로, *'?temperatureValue'*는 상대값이고 온도의 변화량을 의미한다. 즉, *'sign'*에 따라, '+'이면 *'HeaterUp(?temperatureValue)'*이, '-'이면 *'HeaterDown(?temperatureValue)'*이 선택될 것이다.

3.2 대체 서비스 발견

사용자가 원하는 목표 서비스를 성공적으로 제공할 수 있는지의 여부는 협력하도록 명시된 서비스들의 실제 실행 시간 시의 유효성에 달려 있다. 즉, 각 서비스들은 서비스의 규격 중에서 Precondition을 체크하여 유효한 경우에 실제로 바인딩된다. 하지만, 해당 서비스가 유효하지 않다면 그 서비스는 아예 제공이 불가능하고, 명시된 서비스들 중에서 단 1개라도 실행 시간에 유효하지 않으면 목표 자체의 달성이 실패하게 된다.

본 논문에서 제안하는 서비스 온톨로지를 이용한 서비스 합성 기법은 서비스들의 1:1 정적 매칭으로 인한 문제점을 극복하고 서비스 제공의 성공 확률을 높이기 위해 동일한 효과를 낼 수 있는 대체 서비스를 발견한다. 서비스 온톨로지는 위에서 설명한 규격의 "AbsLevel"과 "Child" 항목에 의해 구조화된다. Effect에 의해 분류된 Abstract 메타 서비스를 루트로 해서 해당 Effect와 관련된 서비스들이 여러 레벨로 계층적으로 그룹화되어 있다. 또한, Effect 자체도 온톨로지로 관리함으로써 서비스 발견시에 참조할 수 있다. 다음 (그림 4)는 Effect 온톨로지 내에 포함된 Effect들의 계층 구조를 보여주고 있다.

예를 들어, (그림 5)는 "온도 조절(E-Temperature)"을 Effect로 갖는 서비스들의 구조를 보여주고 있다. <표 2>의 *'manageHeater'*, 서비스를 포함하여 *'manageCooler'*, *'manageWindow'*, *'manageCurtain'*의 4종류의 서비스로 크게 분류되는데, 계절이나 현재의 외부 온도와 같은 상황 정보와 사용자의 선호도 등에 따라 우선 순위가 결정될 것이다. (그림 5)는 *'manageTemperature'* Abstract 서비스를 루트로 총 12개 Composite과 16개 Atomic 서비스를 포함하고 있고, 이



(그림 4) Effect 온톨로지 구조의 일부분

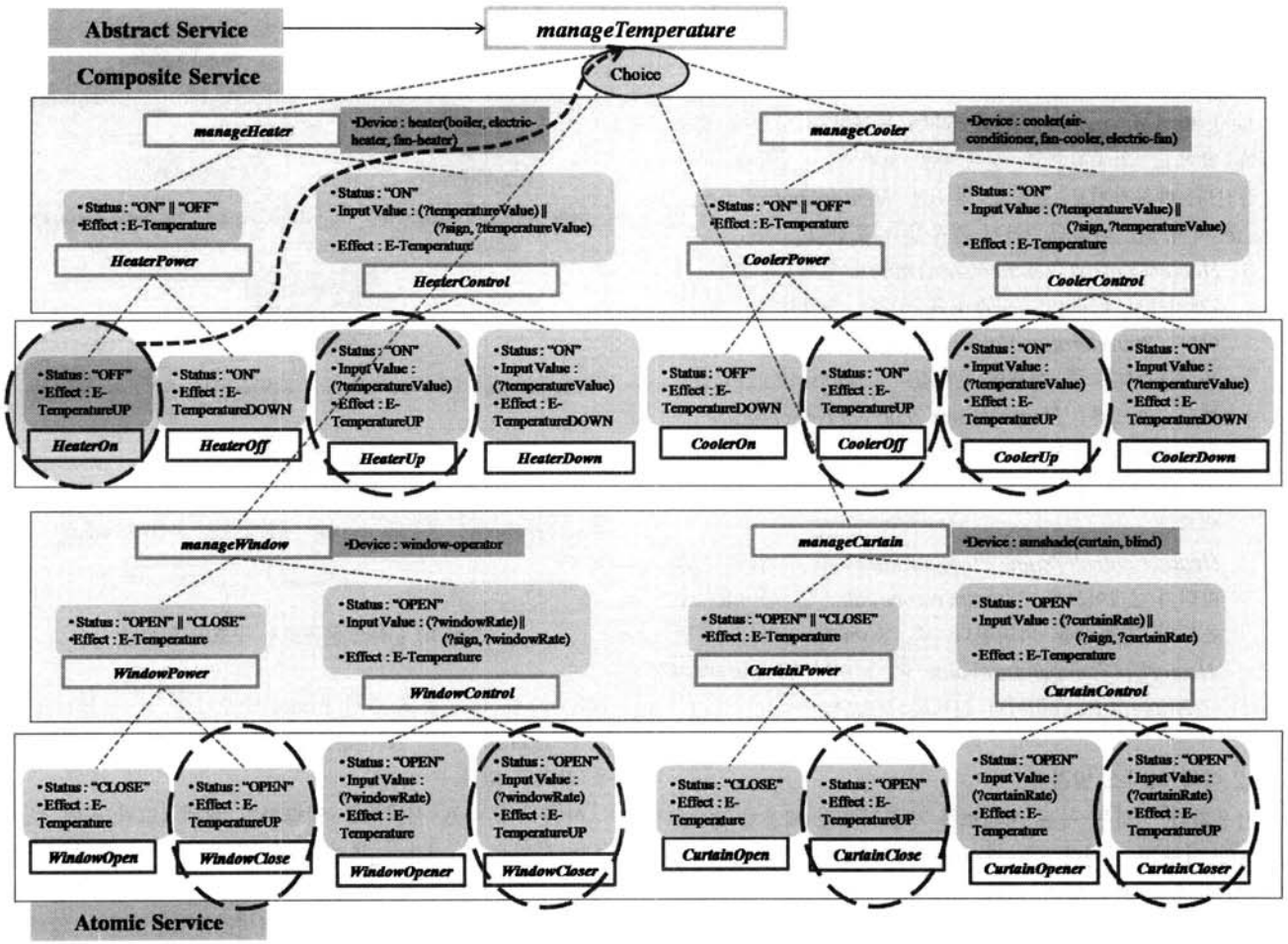
들은 (그림 4)의 온도 관련 Effect 구조처럼 3가지 Effect - E-Temperature, E-TemperatureUP, E-TemperatureDOWN을 가지고 있다.

사용자의 목표 서비스를 기술하기 위해 (그림 5)에 보인 모든 서비스 즉, 3가지 레벨의 서비스가 모두 사용될 수 있다. 우리의 서비스 합성 기법에서는, 서비스 온톨로지가 구축되어 있는 환경이고, 명시된 서비스가 유효하지 않다고 판단되었을 때, 온톨로지의 구조를 따라가면서 동일한 Effect를 낼 수 있는 대체 서비스를 찾는다. 예를 들어, 사용자가 온도를 높일 수 있는 *'HeaterOn'* 서비스를 명시했지만, 실시간에 이 서비스를 실행할 수 있는 기기를 찾지 못했다고 가정하자. 우리의 온톨로지 내에서, *'HeaterOn'* 서비스는 'E-TemperatureUP' Effect를 가지고 있고, *'manageTemperature'*라는 Abstract 서비스의 후손이다. 그 하위에서 동일한 Effect를 가진 *'HeaterUp'*, *'CoolerOff'*, *'CoolerUp'*, *'WindowClose'*, *'WindowCloser'*, *'CurtainClose'*, *'CurtainCloser'*를 찾을 수 있고, 그 중에서 유효한 서비스가 최종적으로 선택되어 명시된 서비스를 대체할 것이다.

3.3 서비스 합성

제안한 서비스 온톨로지는 동적 서비스 합성을 가능케 하기 위한 서비스 기술 및 구조화 방법이다. 또한, 위에서 설명한 서비스 오버로딩과 대체 서비스 발견 기법은 서비스 온톨로지 구조를 통해 동적 서비스 합성의 효과를 극대화할 수 있다. 즉, 실시간의 상황을 고려하여 가장 적절한 서비스가 선택될 수 있도록 한다.

선택된 서비스들의 실제 합성은 유비쿼터스 컴퓨팅 환경을 제어하는 모듈(매니저)이 관장하고, 특정 정책에 따라 서비스들이 협력적으로 구성 및 실행되어 사용자의 목표를 달성할 수 있도록 한다[16].



(그림 5) 서비스 온톨로지를 이용한 대체 서비스 발견 예

4. 실 험

우리는 서비스 온톨로지를 활용하여 효과적인 서비스 합성이 가능함을 보이기 위해 현재까지 구축된 온톨로지에 대한 실험을 진행하였다. 유비쿼터스 환경에서의 실제 서비스를 사용한 실험이 어렵기 때문에, PICO[4]는 그래프 형태의 서비스와 사용자 태스크를 가상으로 생성하고, 서비스의 밀도(service density)에 변화를 주면서 서비스 그래프의 합성으로 태스크의 입력으로부터 출력을 도출할 수 있는지를 평가하였다. 이 실험 방법과 유사하게 우리는 가상의 목표를 설정하고 서비스의 가용성을 다양화하여 모의 실험하였다. 하지만, 실험에 사용된 서비스는 서비스 온톨로지에 포함된 실제 서비스들이다. 사용된 서비스 온톨로지는 홈 및 공공 도메인의 17개 시나리오로부터 구축되었고, 150개의 Atomic 서비스를 비롯하여 총 240개의 서비스를 포함하고 있다. 우리는 실험을 위해 <표 4>와 같은 4가지의 실험 데이터 셋을 설정하였다.

(a)는 가상의 목표를 달성하기 위해 필요한 서비스를 모두 Atomic 서비스라고 가정하고, 구간을 10% 단위로 나누어 서비스 개수를 다양하게 지정한다. (b)는 (a)의 서비스들 중에서 Atomic 서비스로 명시할 것들의 비율로서, 나머지는

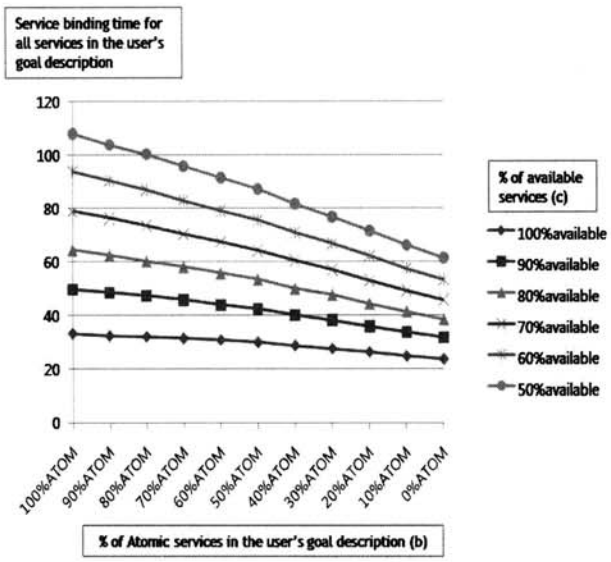
〈표 4〉 실험 데이터

	Simulation Data	Data Value
(a)	# of services for an assumed goal	10%~100% of Atomic services
(b)	# of Atomic services among (a)	0%~100% of (a)
(c)	# of available services in the our Service Library	70%~100% available
(d)	wait time for an unavailable service	1~10 unit time

Abstract하게 명시할 수 있는 것들, 즉 서비스 온톨로지를 통해 구조화되어 있는 것들의 비율을 다양하게 할당한다. (c)는 현재 구축된 서비스 온톨로지 내의 서비스들 중에서 실행 시간에 유효한 것들의 비율을 의미하는데, 각 서비스의 유효 여부는 임의로 결정된다. (d)는 유효하지 않은 서비스를 사용할 수 있을 때까지 기다려야 하는 시간을 각 서비스에 임의로 할당하였다. 예를 들어, 복잡한 목표 서비스 A를 이루기 위해서 {1, 17, 19, 20, 39, 44}번의 6개의 서비스(a)가 요구되는 시나리오가 있다고 가정하자. 실험 데이터 셋의 하나로, 이중에서 80%인 4개는 Atomic으로, 나머지 2개는 Abstract하게 명시할 수 있다(b). Atomic 서비스는 (c)

에서 유효하면 단위 시간 1이, 유효하지 않으면 (d)에서 설정된 만큼의 지연 시간이 걸린다고 가정한다. Abstract 서비스는 서비스 온톨로지의 구조에 따라 대체 서비스를 찾게 되고, 대체 서비스가 유효하면 역시 단위 시간 1을, 유효한 대체 서비스를 찾지 못했을 경우에는 지연 시간을 적용한다.

(그림 6)은 1차 실험 결과로서, 61개의 Atomic 서비스를 포함하여 총 96개의 서비스에 대한 결과이다. 서비스 바인딩 시간(binding time)은 명시된 서비스가 모두 성공적으로 바인딩될 때까지 걸린 단위 시간의 총합을 뜻한다. 1차 실험에서는 <표 4>의 (c)항목을 50%까지 고려하였는데, 70% 이하에서도 서비스 바인딩 시간의 감소율이 유사하였다. 즉, 실행 시의 서비스 가용성이 70% 이하로 떨어지더라도 서비스 바인딩 시간의 변화율에는 영향을 미치지 않음을 확인할 수 있었기 때문에, 2차 실험에서는 100%~70% 경우만을 고려하였다.



(그림 6) 1차 실험 결과 - 서비스 바인딩 시간

(그림 7(a))의 2차 실험과 비교해보았을 때, 서비스 온톨로지를 참조하여 서비스를 바인딩하는 방법이 서비스 온톨로지의 규모, 즉 서비스의 개수에 상관없이 서비스 바인딩 시간을 감소시킬 수 있음을 알 수 있다.

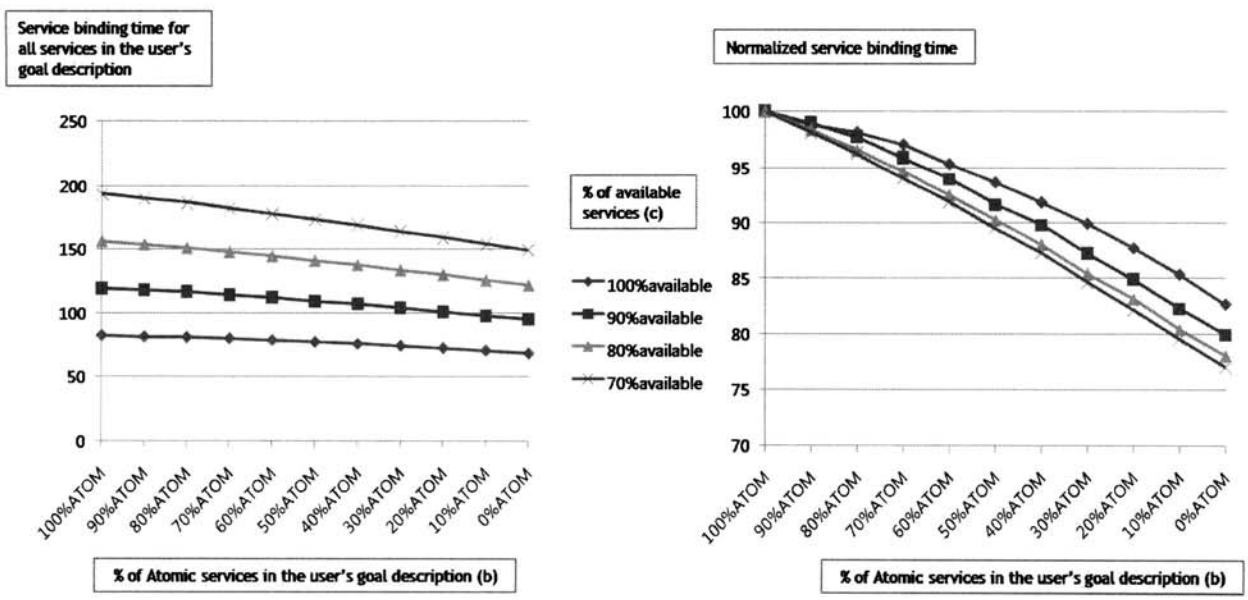
(그림 7)과 (그림 8)은 <표 4>의 (a), (b), (c)의 조합에 따라 10×11×4=440가지의 데이터 셋에 대해 2가지 측정값을 계산한 실험 결과이다. (각 조합에 대해 1000번의 실험 결과의 평균값이다.) (그림 7)은 목표를 달성할 때까지의 총 서비스 바인딩 시간을 의미한다. (그림 7(a))에서, x축은 <표 4>의 (b)를, 각 선은 (c)를, 각 결과값은 (a)에 대한 평균값을 나타낸다. 최종 결과값인 y축은 사용자가 목표를 기술할 때 사용된 서비스들을 모두 성공적으로 실행하기까지의 단위 시간을 계산한 값으로, 다음과 같이 계산한다.

$$ServiceBindingTime = \sum_{i=1}^{i=a} dt(S_i),$$

$$dt(S_i) = \begin{cases} 1, & S_i \text{ is available} \\ \{1 \sim 10\}, & S_i \text{ is not available} \end{cases}$$

(그림 7(b))는 '100%ATOM'일 때(Atomic 서비스만 사용했을 경우)의 값을 100으로 정규화한 결과이다. 실험 결과는 다음과 같이 분석할 수 있다.

- '100%ATOM' 즉, 모두 Atomic 서비스만 사용했을 경우에, '100%available'일 때는 82.5가, '70%available'일 때는 193.5가 걸렸다 - 대체 서비스를 찾을 수가 없기 때문에 명시된 서비스가 유효해질 때까지



(a) 서비스 바인딩 시간 (Service Binding Time)

(b) 정규화된 서비스 바인딩 시간 (Normalized Service Binding Time)

(그림 7) 실험 결과 - 서비스 바인딩 시간

기다려야만 하고, 따라서 지연 시간이 한없이 증가할 수 있다. (당연히 모든 Atomic 서비스가 유효한 '100%available' 경우의 서비스 바인딩 시간이 가장 작다.)

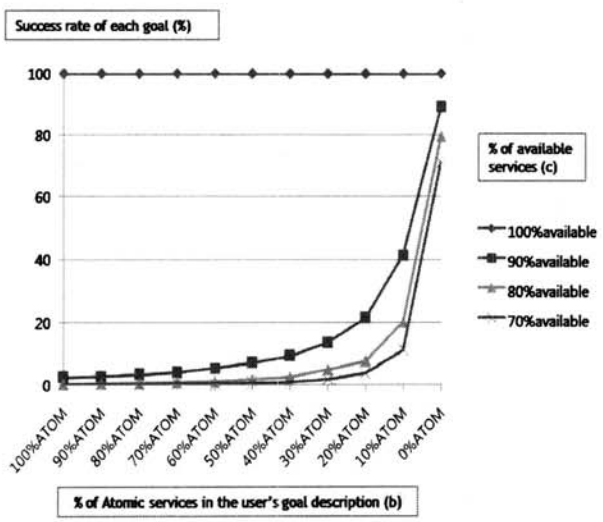
- Abstract 서비스의 사용 비율이 높아질수록 즉, '0%ATOM'으로 갈수록 서비스 바인딩 시간이 감소하였다 - 이는 특정 Effect를 내기 위해 서비스를 정적으로 매칭하고 그 서비스가 유효해질 때까지 기다리는 대신에, 서비스 온톨로지의 구조를 활용해서 대체 서비스를 제공할 수 있기 때문이다.
- 서비스 온톨로지 내의 서비스들 중에서70%가 available한 경우에, 서비스 바인딩 시간의 감소율이 제일 컸다.
- 유효한 서비스의 비율이 아무리 감소하더라도, Abstract 서비스가 사용될수록 크게는 약 23%에서 약 17%까지 서비스 바인딩 시간이 감소하였다.

요약하면, 제안된 서비스 온톨로지를 사용하지 않았을 경우에 서비스가 유효하지 않으면 지연 시간에 따라 서비스 바인딩 시간이 증가하였다. 즉, 해당 서비스가 유효해질 때까지 무한정 기다리거나 목표 달성 자체가 실패하게 된다. 이에 비해, 우리의 서비스 온톨로지를 사용하면, 계층 구조에 기반하여 동일한 Effect를 갖는 서비스를 찾아 대체할 수 있기 때문에 서비스 바인딩 시간이 확연하게 줄었다.

(그림 8)은 사용자의 목표가 성공적으로 달성되었는지를 측정한 결과인데, 명시된 모든 서비스가 바인딩되어야만 성공이라고 간주한다. 즉, 서비스가 1개라도 유효하지 않다면 해당 목표는 실패한다. (그림 8)의 y축이 목표 달성 성공률을 나타내며, 다음과 같이 계산한다.

$$SuccessRate(\%) = \frac{\#\ of\ Success}{\alpha \times \beta} \times 100$$

α : 표 4 (a) 항목의 구간의 수
β : 평균값을 위한 실험 반복 횟수



(그림 8) 실험 결과 - 목표 달성 성공률

이 측정치를 통해 다음과 같은 사실을 알 수 있다.

- 당연히, '100%available'의 경우에는 모든 서비스가 유효하므로 어떤 목표라도 실행에 성공한다(SuccessRate = 100%).
- Abstract 서비스의 사용 비율이 높아질수록 즉, '0% ATOM'으로 갈수록 목표 달성 성공률이 높아진다 - 동일한 Effect를 제공할 수 있는 서비스를 찾아 대체할 수 있기 때문이다.
- 서비스 온톨로지 내의 서비스들 중에서70%가 available한 경우에, 목표 달성 성공률이 가장 크게 증가했다.
- 유효한 서비스의 비율이 아무리 감소하더라도, Abstract 서비스가 사용될수록 크게는 약 2000배에서 약 39배까지 목표 달성 성공률이 증대되었다.

5. 결 론

본 연구에서는 특정 도메인으로부터 서비스를 추출하고 추상화하여 기술하기 위한 규격과 서비스들을 계층적으로 구조화하기 위한 방법을 제안하였다. 제안된 방법에 따라 구축된 서비스 온톨로지는 실행 시간에 동적으로 변하는 상황 정보를 고려하여 필요한 서비스가 선택되어야 하는 유비쿼터스 컴퓨팅 환경의 특성을 지원할 수 있고, 유효하지 않은 서비스의 대체 서비스를 발견함으로써 서비스 제공의 효율을 증대시킬 수 있다는 장점이 있다. 결과적으로, 서비스 온톨로지를 사용함으로써 얻을 수 있는 동적 특성은 두 가지 측면으로 설명할 수 있다. 첫째, 기기의 기능을 '서비스'라는 추상적인 개념으로 표현하기 때문에, 해당 서비스를 제공할 실제 기기와의 바인딩은 실행 시간에 이루어진다. 둘째, 서비스가 유효하지 않은 경우에 동일한 Effect를 갖는 서비스를 발견할 수 있도록 서비스들을 구조화하였기 때문에, 사용자의 목표 달성 확률을 향상시킬 뿐만 아니라 짧은 시간에 목표 달성을 가능케 한다.

현재는 사용자가 반드시 실행되기를 원하는 서비스임에도 불구하고 대체 가능한 서비스가 발견되면 기다리지 않고 실행해버리는 문제점을 극복하여 서비스 합성 방법을 개선하기 위한 연구를 진행하고 있다. 또한, 다양한 도메인을 지원할 수 있도록 서비스 온톨로지를 확장하여 일반적인 유비쿼터스 환경에 적용할 수 있어야 할 것이며, 현재는 수동적으로 이루어지는 서비스 온톨로지 구축 과정의 자동화도 필요하다.

참 고 문 헌

[1] D. Chakraborty, F. Perich, A. Joshi, T. W. Finin and Y.Yesha, "A Reactive Service Composition Architecture for Pervasive Computing Environments," In Proceedings of PWC '02, pp.53-62, 2002.

[2] D. Chakraborty and A. Joshi, "Dynamic Service Composition:

State-of-the-Art and Research Directions," Technical Report TR-CS-01-19, University of Maryland, 2001.

[3] S. Kalasapur, M. Kumar and B. Shirazi, "Seamless service composition (SeSCo) in pervasive environments," In Proceedings of MSC '05, pp.11-20, 2005. 11.

[4] S. Kalasapur, M. Kumar and B. Shirazi, "Dynamic Service Composition in Pervasive Computing," IEEE Transactions on Parallel and Distributed Systems, Vol.18, No.7, 2007.07.

[5] J. Sousa., V. Poladian, D. Garlan, B. Schmerl and M. Shaw, "Task-based Adaptation for Ubiquitous Computing," IEEE Transactions on Systems, Man, and Cybernetics, Vol.36, No.3, 2006.05.

[6] A. Ranganathan, C. Shankar and R. Campbell, "Application Polymorphism for Autonomic Ubiquitous Computing," Multi-agent and Grid Systems, Vol.1, No.2, pp.109-129, 2005.

[7] M. Klein and B. Konig-Ries, "A Process and a Tool for Creating Service Descriptions based on DAML-S," 4th VLDB Workshop on Technologies for E-Services (TES'03), pp. 143-154, 2003.07.

[8] D. Chakraborty, "Service Discovery and Composition in Pervasive Environments," Ph.D. Thesis, 2004.06.

[9] S., Ponnekanti, L. Brian, F. Armando, H. Pat and W. Terry, "ICrafter: A Service Framework for Ubiquitous Computing Environments," Ubicomp, pp.56-75, 2001.

[10] N. F. Noy and D.L. McGuinness, "Ontology development 101: A guide to creating your first ontology," Stanford Knowledge Systems Laboratory Technical Report, 2001.

[11] D. L. McGuinness and F. van Harmelen, "OWL Web Ontology Language Overview," W3C Member Submission, 2004.

[12] D. Martin, et al., "OWL-S: Semantic Markup for Web Services," W3C Member Submission, 2004.

[13] B. David and C. K. Liu, "Web Services Description Language (WSDL) Version 2.0 Part 0: Primer," W3C Member Submission, 2007.

[14] B. Steve, et al., "Semantic Web Services Language (SWSL)," W3C Member Submission, 2005.

[15] Protégé, http://protege.stanford.edu/.

[16] H.Kim, Y.Shim, D.Choi, S.Kim, and W.Cho, "Community Manager: A Dynamic Collaboaration Solution on Heterogeneous Environment", In Proceedings of ACS/IEEE International Conference on Pervasive Services, pp.39-46, 2006. 6.

이 미 연

e-mail : ailmy@ewhain.net

2003년 이화여자대학교 컴퓨터학과 (학사)

2005년 이화여자대학교 과학기술대학원 컴퓨터학과 (석사)

2007년~현 재 이화여자대학교 컴퓨터정보통신공학과 박사과정

관심분야 : 유비쿼터스컴퓨팅, 온톨로지, SOA, 인공지능 등

이 정 원

e-mail : jungwony@ajou.ac.kr

1993년 이화여자대학교 전자계산학과 (학사)

1995년 이화여자대학교 전자계산학과 (석사)

1995년~1997년 LG종합기술원주임연구원

2003년 이화여자대학교 컴퓨터학과 (박사)

2003년~2006년 이화여자대학교 컴퓨터학과 BK교수, 전임강사(대우)

2006년~현 재 아주대학교 정보통신대학 전자공학부 조교수

관심분야 : SOA, 유비쿼터스 컴퓨팅, 임베디드 소프트웨어

박 승 수

e-mail : sspark@ewha.ac.kr

1974년 서울대학교 수학과 (학사)

1976년 한국과학기술원 전산학 (석사)

1988년 미국 텍사스대학 전산학 (박사)

1988년~1991년 미국 켄사스대학 컴퓨터학과 조교수

1991년~현 재 이화여자대학교 공과대학 컴퓨터정보통신공학과 교수

관심분야 : 시맨틱웹, 온톨로지, 인공지능 등

조 위 덕

e-mail : chowd@ajou.ac.kr

1981년 서강대학교 전자공학과 (학사)

1983년 한국과학기술원 전기 및 전자공학과 (석사)

1987년 한국과학기술원 전기 및 전자공학과 (박사)

1983년~1990년 금성전기(현 LG전자) 기술연구소 DSP 연구실장

1990년~1991년 한국생산기술연구원 전자정보시스템연구부 팀장/조교수

1991년~2003년 전자부품연구원시스템연구본부 본부장

2003년~현 재 유비쿼터스컴퓨팅사업단 단장, 아주대학교 전자공학부 교수

관심분야 : 유비쿼터스 컴퓨팅/네트워크, 센서네트워크, Post-PC(차세대 Smart PDA), Interactive DTV 방송기술, 고품질 홈서버/게이트웨이 기술, 디지털방송/이동통신 연계 융합플랫폼기술, 무선인터넷응용기술