상태 유도 조합을 이용한 규칙 기반의 적응형 서비스 프레임워크

정우성*○ 유찬우* 박동훈* 이병정** 김희천*** 우치수*
서울대학교 컴퓨터공학부*, 서울시립대학교 컴퓨터과학부**, 한국방송통신대학교 컴퓨터과학과***
{wsjung○, chanwoo, donghun, wuchisu}@selab.snu.ac.kr*, bjlee@venus.uos.ac.kr**, hckim@knou.ac.kr***

A Rule-based Adaptive Service Framework with State-driven Composition

Woosung Jung○* Chanwoo Yoo* Donghun Park* Byoungjeong Lee** Heechern Kim*** Chisu Wu*
School of Computer Science & Engineering, Seoul National University*
Dept. of Computer Science, The University of Seoul**
Dept. of Computer Science, Korea National Open University***

요 약

적응형 소프트웨어는 유비쿼터스 컴퓨팅의 핵심 분야로 홈 네트워크, 지능 로봇 등 다양한 분야에 응용이 가능하다. 하지만, 대부분의 연구가 적응형 소프트웨어의 요구사항이나 시나리오의 구체화를 위한 비전을 제시하거나, 응용 구현의 사례를 보임으로써 실현 가능성을 확인하는데 초점을 맞추고 있다. 본 연구에서는 동적인 재구성이 가능한 규칙 기반으로 동작할 수 있기 때문에 진화가 가능하며, 상태에 기반하여 행위를 판단하는 RASC 프레임워크를 제안한다. 기존 서비스 조합의 개념을 실현하기 위해 중개자를 이용하였으며, 변이와 교차와 같은 유전 알고리즘 연산을 쉽게 적용할 수 있도록 RASC 도메인을 정의하였다. RASC 프레임워크는 블랙보드 아키텍처를 기반으로 규칙을 공유하며, 서비스와 서비스 조합체 모두 자극-반응 모델을 따르고 자기유사성을 가지는 일종의 복잡계를 구성한다.

1. 서 론

적응형 소프트웨어는 유비쿼터스(Ubiquitous) 컴퓨팅의 핵심 분야중 하나로 지능형 홈 네트워크 및 로봇 등 다양한 도메인 응용에 필요할 뿐 아니라 상황인식(Context-aware) 컴퓨팅과의 연계에 있어서도 중요한 위치를 차지하고 있다. 소프트웨어의 적응은 "소프트웨어가 자신의 행위를 평가하여 원래 목적에 맞지 않거나, 보다 나은 기능 또는 성능을 발휘하는 것이 가능할 때 스스로의 행위를 변경하는 것"을 의미한다[1]. 하지만, 대부분의 연구가 적응형 소프트웨어의 요구사항이나 시나리오의 구체화를 위한 비전(Vision)을 제시하거나, 단순한 응용 구현의 사례를 보임으로써 실현 가능성을 확인하는데 초점을 맞추고 있다. 또한, 일부 연구는 상황 정보를 종합적으로 분석하려고 함으로써 증가한 복잡도에 비해 좋은 품질의 결과를 얻지 못하는 경향을 보이기도 한다. 이러한 기존 연구들은 적응형 소프트웨어 도메인에서 재사용이 가능한 설계나 프레임워크를 연구함에 있어서 부족한 점이 있다.

한편, 최근 등장한 서비스 지향 아키텍처(Service oriented architecture)는 컴포넌트 기술의 확장성 및 호환성의 한계를 극복함에 있어 좋은 방향을 제시해 주고 있다. 더불어, 인공지능 분야에 바이오 기술이나 복잡계(Complex system)의 연구 성과를 적용하려는 움직임이 활발해지면서 적응형 소프트웨어는 새로운 국면을 맞이하고 있다. 뿐만 아니라, 시멘틱 웹(Semantic web)이나 온톨로지(Ontology) 등의 연구는 엄격한 문법을 따르는 인터페이스를 만족해야만 조합이 가능하였던 기존의 틀을 깨고, 의미 중심의 통신이 가능함을 보여 주었다.

본 연구에서는 전체 시스템이 만족하고자 하는 전역 목표(Global goal)를 서비스 단위 상태들의 집합으로 보고 서비스별로 독립적인 적응이 이루어지도록 하였다. 즉, "다양한 서비스들이 각각의 해당 규칙을 만족함으로써 전체 목적에 부합한 방향으로 동적인 조합을 이루는 것"을 서비스의 적응으로 정의한다.

논문의 구성은 다음과 같다. 2장에서는 소프트웨어나 서비스의 적응과 관련한 기존 연구들에 대해 알아보고 문제점들을 확인한다. 3장에서는 규칙에 기반하여 동적 적응이 가능한 서비스 환경을 제공하는 RASC(Rule-based Adaptive Service Composition) 프레임워크의 아키텍처 개요 및 해당 구성 요소들에 대해 자세히 설명한다. 4장에서는 RASC 프레임워크의 도메인을 정의하고, 5장에서는 규칙과 RASC 도메인 등을 구현함에 있어서 제스(Jess)와 관계형 데이터베이스와 같은 기존 기술을 이용한 구현 전략 및 응용 시나리오에 대해 기술한다. 6장에서는 본 연구의 의미, 한계점 및 향후 과제를 언급하고 결론을 맺는다.

2. 관련 연구

퍼베이시브 컴퓨팅(Pervasive computing) 환경을 위한 PICO(Pervasive Information Community Organization)라는 미들웨어 프레임워크에 대한 연구가 있다[2]. PICO는 구성 요소로 델리전트(delegents, intelligent delegates)라는 지능을 가진 소프트웨어와 카밀레온(camileuns)이라는 하드웨어 장비로 구성된다. 원격진료(Telemedicine)에 대한 시나리오를 예로 들고 있으며, 임무(Mission)가 주어지면 PICO의 구성 요소들이 목적을 달성하기 위해 협업을 하는 커뮤니티(Community)를 구성하게 된다. 새로운 환경을 위해 필요한 요구사항과 관련 연구 주제들을 제안하고 있지만, 내용이 시나리오에 중점을 두고 있기 때문에 구현과의 연계성을 파악하기 어렵다.

그래비티(Gravity) 프로젝트의 경우는 서비스를 제공하는 컴포넌트 모델(Service component model)을 제안하고, 컴포넌트 설명(Component description)과 조합 설명(Composition description)을 XML로 표현 및 구현하고 있으며, 동적 가용성(Dynamic availability)을 처리할 수 있도록 구현하는데 초점을 맞추었다[3]. 하지만, XML로 기술된 내용이 상세 설계와

• 본 연구는 한국과학재단 특정기초연구(R01-2006-000-11150-0)지원으로 수행되었음.

유사하기 때문에 규칙의 변화를 주기 어려워 실제 사례로 보이고 있는 웹브라우저의 플러그인(Plug-in)과 같은 응용에 적합하다. 이 밖에 동적 적응 모델에 대한 원칙이나 문제들을 정리하고 있다.

그 밖에, 적응형 시스템에 대한 프레임워크를 아키텍처 기반으로 제시하려는 연구들이 진행되었다[4][5][6][7]. 첫째, 소프트웨어 적응을 위한 아키텍처를 제시하고 그에 따라 실제로 적응형 시스템을 구현한 연구들이 있다. 이 연구들에서는 아키텍처 적응에 있어 필요한 제약과 목표를 기술하기 위한 아키텍처 스타일을 정의하고, 이를 기반으로 시스템을 구현하였다. 그러나 시스템의 일부가 이미 존재한다고 가정한 상태에서 추상적인 프레임워크만을 제시하였고, 구현물에 대한 정형화 작업이 부족하여, 그 구현이 프레임워크의 유효성과 효과성을 입증하기 어렵다는 한계를 지닌다. 둘째, 아키텍처의 전 부분에 대해 조망하기보다는 간단한 모델 제시 후, 특정 부분의 설계와 구현에 초점을 맞춰 진행한 연구들이 있는데, 여기에는 적응형 소프트웨어의 아키텍처를 모니터링하고, 기록하고, 변경하기 위한 관리 도구를 제시하고 구현한 연구[8]와 유닛들의 조합을 통한 적응형 시스템 구축을 위해, 적응형 시스템에 적합한 특성들을 지원하는 유닛들의 아키텍처를 기술한 연구[9] 등이 있다. 하지만 적응형 시스템에 필요한 구성 요소들과 아키텍처가 충분히 정의되지 않은 상태에서 특정 부분만을 제한적으로 가정하여 구현하는 것은 다른 부분들에 대한 가정의 유효성을 입증하기 어렵다.

3. RASC 프레임워크

3.1 개요

RASC 프레임워크는 서비스 및 서비스들의 조합체 모두 자극-반응(Stimulus-Response) 모델을 기본 골격으로 하고 있으며, 자기유사성(Self-similarity)을 특징으로 가지는 복잡계의 일종이다. RASC에서의 서비스는 완전한 역할을 수행하는 하나의 시스템이며, 독립적으로 자원을 할당받아 병렬적으로 실행 가능하다. 이는 실제로 대부분의 적응체들이 순차적 방식이 아닌 동시적 반응을 보이는 것과 같은 이유에서이다. 각각의 자극-반응체들은 독립적인 역할을 수행하면서 결합하여 하나의 새로운 자극-반응체를 구성하게 된다.

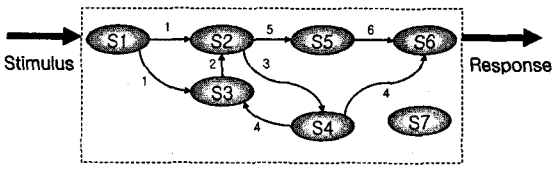

그림 1 직접 연결을 통한 서비스 조합

하지만, 서비스 조합의 의미를 있는 그대로 구현한 경우, 전체 서비스들의 협업 관계를 파악하거나 각 서비스들끼리의 메시징을 처리하기 위해서는 분산되어 있는 서비스들을 P2P (Peer-to-peer)로 처리해야 하는 어려움이 있다. 주로, 적응형 소프트웨어의 시나리오나 비전을 제시하는 대부분의 논문들이 그림 1과 같은 순수 그래프 상태로 동적인 서비스의 조합 문제를 다루고 있다. 하지만, 그림 1의 실현을 위해서는 미리 고정된 시나리오를 따르거나 각각의 컴포넌트나 조합에 대해 미리 정의한 상세 기술이 주어져야 한다.

RASC 프레임워크는 서비스의 조합을 중앙 제어를 담당하는 중개자(Broker)를 이용한 메시지 교환으로 처리하기 때문에 그림 2에 나타난 것처럼 서비스의 조합이 이루어진다. 이로 인하여, RASC 프레임워크는 서비스간의 직접 연결에 비해 불필요한 메시지 전송이 2배 가까이 이루어지고, 메시지 처리를 위한 부하가 중개자에 집중된다는 단점이 있다. 하지만, 메시지의 대부분이 서비스들간의 협업과 관련해 필요한 요약된 상태변화에 따른 이벤트들이므로 실시간으로 이루어지는 메시지는 발생하지 않는다. 만약, 서비스간에 실시간으로 전달이 필요한 동영상이나 용량이 큰 자료의 공유가 필요한 경우는 서비스 중개자를 통해 해당 자료의 형식(ValueType)과 위치(URI)를 전달하여 서비스간의 직접 연결이 가능하다.

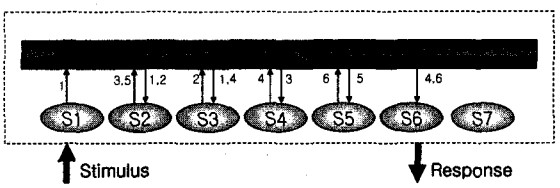

그림 2 중개자를 통한 서비스 조합

3.2 아키텍처 정의

그림 3은 RASC 프레임워크의 구성 요소와 저장소 및 이들의 관계를 보여준다. 서비스를 중개자에 연결하기 위해서는 S-연결자(S-Connector)를 거쳐야 하며, 이는 IServiceBroker라는 인터페이스를 구현한 것이다. 하지만, 메시징 방법은 COM, CORBA, XML 웹서비스 등 어떠한 것을 이용해도 상관없다.

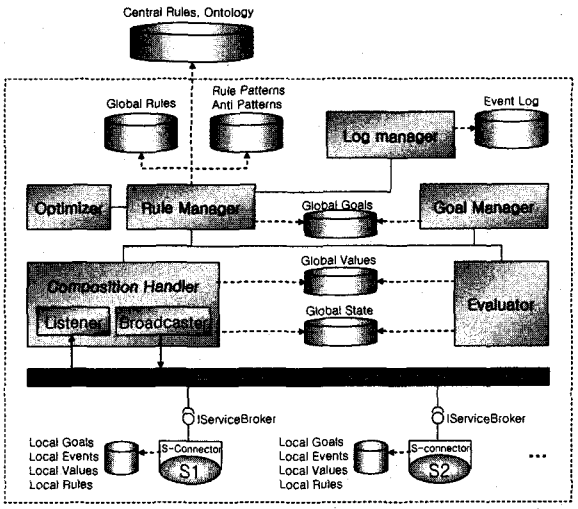

그림 3 RASC 프레임워크 아키텍처

RASC 프레임워크 아키텍처의 서비스 및 서비스 조합의 기본 구성은 앞서 언급한 자극-반응체 모델을 따르고 있으나, 적응을 위해 필요한 규칙 정보를 공유함에 있어서는 블랙보드 아키텍처(Blackboard architecture)를 이용한다[10]. 블랙보드 아키텍처는 다양한 종류의 문제 해결을 위한 에이전트들(Heterogeneous problem-solving agents)간의 정보 공유를 목적으로 설계된 것이다. RASC 프레임워크의 경우도 중앙에 정

보 저장소(Blackboard)를 두고, 다양한 지역 규칙들(Local rules)을 통합하여 전역 규칙들(Global rules)을 만들어내고 이를 모든 서비스들의 협업을 위해 사용하게 된다.

RASC 프레임워크는 미들웨어 역할을 하는 S-연결자(S-Connector), 이벤트 메시지를 처리하는 조합 처리자(Composition Handler), 전역 규칙의 필터링, 검색 등의 역할을 하는 규칙 관리자(Rule Manager), 서비스 중개자의 실행 과정에서 발생하는 주요 이벤트를 남기는 로그 관리자(Log Manager), 목표 설정을 위한 목표 관리자(Goal Manager) 및 서비스 조합의 진행 방향을 평가하는 평가자(Evaluator) 등으로 구성된다. 그 외, 실제 구현을 위해서는 규칙 적용에 있어서 우선 순위 등과 같은 여러 가지 환경 정보를 관리해주는 환경 관리자(Configuration Manager)가 필요하지만, 여기서는 적용과 직접적으로 관련하여 설명이 필요한 주요 구성 요소만을 다루기로 한다.

3.2.1 S-연결자(S-Connector)

서비스는 데이터 처리를 하거나, UI를 생성하거나, 특정 알고리즘을 수행하여 결과를 반환하는 역할로 구분하거나, 메시지의 입/출력에 따라 센서(Sensor), 행위자(Actuator), 제어자(Controller) 등으로 나눌 수 있다. 하지만 같은 종류라 하더라도 그 구현 의미나 인터페이스에 따라 다양한 종류의 서비스가 존재한다. 이러한 서비스를 서비스 중개자의 입장에서 동일한 방식으로 관리하기 위해서는 S-연결자와 같이 서로 약속된 인터페이스를 통해서 구현된 서비스로의 접근이 가능해야 한다.

표 1 의사코드(Pseudo code)로 표현한 IServiceBroker

```
interface IServiceBroker
{
    public Register(connectionString:String)
    public Unregister()

    public Execute(acts:Action[])
    public delegate EventRaised(evts:Event[])
    public GetValue(a:Attribute)

    public _UploadRule()
    public _DownloadRule()
    public _UploadValueTable()
}
```

표 1은 RASC 프레임워크에서 사용하는 인터페이스를 나타낸다. IServiceBroker는 실제 구현을 위한 인터페이스라기보다는 RASC 프레임워크가 동작함에 있어서 필요한 최소한의 인터페이스를 언급하기 위한 수단으로 의사코드로 표현하였다. 하지만, EventRaised와 같이 델리게이션(Delegation)을 이용한 함수 포인터는 XML 웹 서비스(Web services)의 경우 직접적인 구현이 불가능하므로 해당 서비스는 관련한 이벤트 메시지를 중개자로 능동적으로 보낼 수 있고, 중개자는 그 메시지를 받아서 처리할 수 있도록 구현해야 한다.

S-연결자에는 서비스의 등록과 해지, 서비스가 가지고 있는 값에 대한 접근이나 실행, 규칙 정보의 업로드와 다운로드를 위한 기본 인터페이스가 구현되어 있으며, 해당 서비스가 지역 목표에 따라 어떻게 적응할 수 있을지를 알려주는 규칙과 추상화된 이벤트값들이 정의되어 있어서 무한한 개수의 실제 이벤트 도메인을 의미 있는 유한 개의 이벤트로 필터링 해주는 역할도 한다.

3.2.2 조합 처리자(Composition Handler)

기본적으로 서비스들로부터 전달되는 메시지를 수신하고, 처리된 결과에 따라 특정 서비스들을 실행시키기 위한 메시지를 송신하는 역할을 담당한다. 특정 포트(port)를 열어 놓고, IServiceBroker를 이용하여 통신을 하게 되는데, 실제 프로토콜을 구현할 경우는 XML기반으로 하게 되므로 고성능의 XML파서를 포함하게 된다.

조합 처리자는 특정 서비스의 실행에 필요할지도 모르는 전역값들(Global values)과 규칙을 적용하는데 필수적인 전역 상태(Global state)도 관리한다. 이 데이터는 등록된 서비스로부터 전달된다. 만약, 스트림(Stream) 형식으로 전달해야 하는 동영상과 같은 대용량 데이터의 경우는 직접 연결에 필요한 URI를 알려주기도 한다. 조합 처리자는 규칙 관리자에게 규칙을 전달하거나 규칙이나 규칙의 적용 결과인 액션에 대한 정보를 전달받게 된다.

3.2.3 규칙관리자(Rule Manager)

각 서비스로부터 업로드된 지역 규칙들을 통합하여 하나의 전역 규칙으로 관리한다. 또한 목표와 전역 상태 등에 따라 규칙을 적용하고, 해당 상태가 제한 패턴(Anti patterns)에 해당하지 않는지 항상 감시한다. 이는 보안, 안전성 등의 제어를 위해 필수적으로 제공되어야 할 부분으로 패턴 기술 언어(Pattern description language)를 통해 프로그래밍 된 후에 컴파일(Compile)되어 규칙 도메인 형태로 변환된다. 만약, 서비스 부재와 같이 접근하려는 대상이 존재하지 않아서 발생하는 규칙 적용 실패, 또는 제한 패턴 발생, 최적화된 규칙 발견 등의 이벤트와 같은, 프레임워크에 미리 정의된 전역 이벤트에 대해서는 이를 로그 관리자에 전달하여 로그로 남긴다. ServiceNotFound, AttributeNotFound, ActionNotFound, ServiceRegistered, ServiceUnregistered 등이 대표적인 전역 이벤트이며, 상세한 내용을 담은 추가정보도 함께 전달된다. 로그는 정기적으로 분석되어 특정 목표 수행을 위해 추가적으로 필요한 서비스나 기능을 검색하거나, 이를 외부 서비스 조합체에서 찾도록 하는 서비스 조합체간 협업을 위해 필요한 메시지를 만드는데 사용할 수 있다. 외부에서 필요한 서비스를 찾는데 성공하면 경우에 따라 서비스의 복제가 이루어질 수도 있다.

간혹, 충분히 검증된 규칙 패턴이 반복적으로 발생할 수 있다. 이러한 규칙은 중앙 규칙 데이터베이스(Central Rules)로 전송하게 된다. 중앙 규칙은 각 서비스 조합체가 자체적인 규칙을 사용하거나 조작하는 것만으로 목표 달성이 어렵다고 판단될 경우 참조할 수 있다. 또한 중앙 규칙 데이터베이스로부터 규칙을 만드는데 필요한 온톨로지(Ontology) 정보를 얻을 수도 있다.

3.2.4 평가자(Evaluator)

규칙이 간단한 경우에는 큰 문제가 없지만, 복잡한 문제에 대한 목표를 해결함에 있어서는 방향을 지시해줄 수 있는 평가자가 반드시 필요하다. 이는 유전 알고리즘에서의 평가 함수(Evaluation function)에 해당하는 것으로, 기존 규칙만으로 목표 달성이 어려울 경우 변이(Mutation)나 교차(Crossover) 등을 통해 새로운 규칙을 생성함으로써 목표 달성을 위한 돌파구를 찾게 된다[11]. 이것은 지역 최적점(Local optimum)을 벗어나 전역 최적점(Global optimum)을 찾기 위한 방법이기도 하다. 하지만, RASC 프레임워크는 평가자 구현을 위한 내부 알고리즘에 대해서는 언급하지는 않고, 아키텍처에서의 역할과 패턴의 도메인 형식만 정의한다.

평가자를 자동화하지 않고, 사람이 직접 상과 벌에 해당하는 점수를 줄 수도 있다. 평가자가 확보한 목표, 규칙에 따른 평가 결과를 기록(History)으로 남겨둠으로써 향후 비슷한 목표를 달성하려고 할 때 제한 또는 적용 패턴으로의 재사용이 가능하다.

4. RASC 도메인

RASC 프레임워크의 도메인을 정형적으로 정의하여, 필요한 함수나 데이터들의 관계나 형식을 검증하고 구현하기 위한 기초 명세로 활용할 수 있다.

4.1 서비스, 속성, 값(Service, Attribute, Value)

표 2 서비스, 속성, 값 관련 도메인

$s \in Service = SemanticServiceId$
$attr \in Attribute = SemanticAttributeId$
$v \in Value = Variant + ValuePointer$
$vp \in ValuePointer = ValueType \times URI$
$valueTable \in (Service \times Attribute) \rightarrow Value$
$[[Cond]] \in ((Service \times Attribute) \rightarrow Value) \rightarrow Boolean$

표 2는 서비스, 속성 및 값과 관련한 도메인을 정의해 놓은 것이다. $Service$,$Attribute$는 모두 문자열(String) 형식과 같은 $SemanticId$ 도메인의 값을 원소로 하는 도메인이며, $Value$의 경우는 앞서 언급한 바와 같이 $Variant$나 $ValuePointer$ 형식을 따른다. $ValuePointer$는 형식과 URI의 투플(tuple)로 표현된다.

$valueTable$은 서비스와 속성을 지정해주면 해당값을 반환하는 함수로 각 서비스들이 필요로 하는 값을 가져오기 위한 도메인으로 볼 수 있다. 뿐만 아니라 서비스, 속성, 그리고 조건자로 구성되는 조건식 문법을 따르는 $Cond$의 값을 얻기 위한 하나의 환경으로 볼 수 있기 때문에 $Cond$가 동일하더라도 $valueTable$이 바뀜에 따라 결과값도 달라지게 된다.

4.2 이벤트(Event)

표 3 이벤트 도메인에서의 갈로아 연결

$evt \in Event,\ \widehat{evt} \in \widehat{Event} = SemanticEventId$
$\forall evt \in Event,\ \widehat{evt} \in \widehat{Event} : \alpha(evt) \sqsubseteq \widehat{evt} \Leftrightarrow evt \sqsubseteq \gamma(\widehat{evt})$

서비스의 상태를 변화시키는 기초가 되는 무한하며(infinite) 연속적인(continuous) 실제의 외부 이벤트 도메인을 $Event$로 정의하고, RASC 프레임워크에서 사람이 요약한(abstract), 유한하고(finite), 불연속적인(discrete) 이벤트 도메인을 $\widehat{Event}$라고 정의한다. 무한개의 원소를 가진 도메인이 유한개의 원소를 가진 도메인으로 매핑이 되는 것은 $valueTable$이 주어졌을 때, 서비스와 속성, 그리고 조건식을 구성하는 문법을 따르는 $Cond$의 의미인 $[[Cond]]$의 결과에 따라 요약된 이벤트가 매핑되기 때문이다.

도메인의 원소들은 집합 포함 관계에 의해 CPO(Complete Partial Order)가 되며, 요약된 원소인 $\widehat{evt}$가 실제 원소인 evt를 포섭하는 갈로아 연결(Galois connection)을 이루게 된다. 이 관계를 나타낸 것이 표 3과 같다. 여기서, α는 도메인이 $Event \rightarrow \widehat{Event}$로 정의되는 요약(abstraction)함수가 되고, γ는 $\widehat{Event} \rightarrow Event$인 구체(concretization)함수가 된다.

4.3 상태(State)

표 4 상태 관련 도메인

$ls \in LocalState,\ \widehat{ls} \in \widehat{LocalState} = Service \times 2^{\widehat{Event}}$
$gs \in GlobalState,\ \widehat{gs} \in \widehat{GlobalState} = 2^{\widehat{LocalState}}$

$s_i \in Service$ $(1 \le i \le n,\ n \in N)$일 때, $NumOfEvent$의 도메인은 $Service \rightarrow Integer$이고, 서비스를 주면 해당 서비스가 가진 지역 이벤트의 개수를 반환하는 함수라고 했을 때, 전역 상태의 경우의 수는 $\prod_{i=1}^{n} 2^{NumOfEvent(s_i)}$가 된다. 유한으로 요약된 도메인임에도 불구하고 모든 경우를 고려했을 때, 평균적으로 e개의 이벤트를 가지는 경우 전역 상태의 경우의 수는 2^{en}가 된다. 평균적으로 4개의 이벤트를 가지는 5개의 서비스만 있다고 하더라도 경우의 수는 백만이 넘는다. 하지만, 규칙을 표현함에 있어서 모든 상태에 대해 정의해야 하는 것은 아니다. 와일드카드를 이용하여, 고려하지 않아도 되는 상태를 지정할 수 있기 때문이다.

$Event$,$\widehat{Event}$ 도메인의 원소들이 갈로아 연결을 만족하므로 $LocalState$와 $\widehat{LocalState}$, $GlobalState$와 $\widehat{GlobalState}$의 원소들 간에도 갈로아 연결이 성립하게 된다. 이는 실제 서비스들의 전체 상태를 요약된 전체 상태가 포섭할 수 있음을 의미한다.

4.4 행위(Action)

표 5 행위 관련 도메인

$act \in Action = SemanticActionId$
$la \in LocalAction = Service \times 2^{Action}$
$ga \in GlobalAction = 2^{LocalAction}$

행위는 서비스 반응에 해당하며, 상태와 유사하게 지역 행위와 전역 행위로 구분되어 있다. 구현을 위해서는 단순한 행위 정보 외에 서비스 인스턴스(Instance)들의 구분, 문법(Syntax) 일치 여부 등을 판단하기 위한 메커니즘이 필요하다. 의미가 다른 인스턴스는 새로운 $SemanticActionId$를 부여하거나, 부분 일치(Partial matching)하는 인터페이스를 처리할 수 있어야 한다.

본 연구에서는 $SemanticId$가 일치할 경우 문법도 일치함을 가정한다. RASC 도메인에서의 $Action$은 파라미터 값을 가져올 경우, 다른 서비스로부터 직접 넘겨받는 것이 아니라 $< Service, Attribute >$를 통해 중앙 저장소의 $valueTable$에서 얻어진 값을 가져오는 것으로 한다. $Action$은 시퀀스(Sequence)와 병렬(Concurrent) 행위로 구분할 수 있으나, 시퀀스 행위의 경우는 2개의 규칙으로 나누어서 처리하면 같은 결과를 얻게 되므로, 구분하지 않고 모두 병렬 행위로 처리한다.

4.5 규칙, 목표(Rule, Goal)

규칙 도메인($Rule$)의 원소는 목표($Goal$)에 따라 현재 전역 상태($\widehat{GlobalState}$)가 주어지면, 실행해야할 관련 전역 행위

(*GlobalAction*)를 우선순위(*Priority*)에 따라 찾을 수 있도록 해준다. RASC에서는 목표의 경우도 우선순위로 구분되는 전역 상태로 본다. 그렇기 때문에, 규칙은 현재의 전역 상태를 목표에 해당하는 전역상태로 바꾸기 위한 행위들을 묶어 놓은 것으로 볼 수 있다. 이러한 규칙은 각 서비스들의 의미를 명확하게 해주는 사전,사후 조건(Pre/Post condition) 등과 같은 제약을 정의해주는 역할도 한다.

표 6 규칙, 목표 관련 도메인

$$r \in Rule = Goal \rightarrow (GlobalState \rightarrow (Priority \rightarrow GlobalAction))$$
$$pri \in Priority = Integer$$
$$g \in Goal = Priority \rightarrow GlobalState$$

초기 상태를 아래쪽에 두고 목표 상태를 위쪽에 배치하여 상태의 변화가 가능한 것의 방향을 지정하여 연결함으로써 일종의 체인(chain)을 만들 수 있는데, 목표에 대한 현재 전역 상태를 평가할 수 있기 위해서는 앞서 설명하였던 평가자가 필요하게 된다. 규칙을 매번 실행하면서 바뀌는 전역 상태를 평가자가 감시하고, 전역 상태들을 목표 방향으로 진행시킬 수 있다.

간혹 목표가 아닌 특정 상태에서 정지 상태에 머무는 경우도 있다. 이를 해결하기 위해서는 규칙을 정적분석(Static analysis)함으로써 오류를 사전에 찾아내거나 다양한 시뮬레이션을 통한 검증을 실시해야 한다. 동적인 상황에 있어서는 변이나 교차를 통해 새로운 규칙을 임시적으로 적용하여 정지 상태를 탈출할 수도 있다.

Rule 도메인을 따르는 규칙을 사용할 경우, 인간이 규칙을 해석하긴 어렵지만, 유전 알고리즘 연산과 같은 다양한 조작이 쉽고 간단하게 이루어질 수 있다는 장점이 있다. 규칙 패턴을 분석하여 하나의 산출물로서 찾아내는 것도 한 연구 분야가 될 수 있다.

5. 구현 전략 및 응용 시나리오

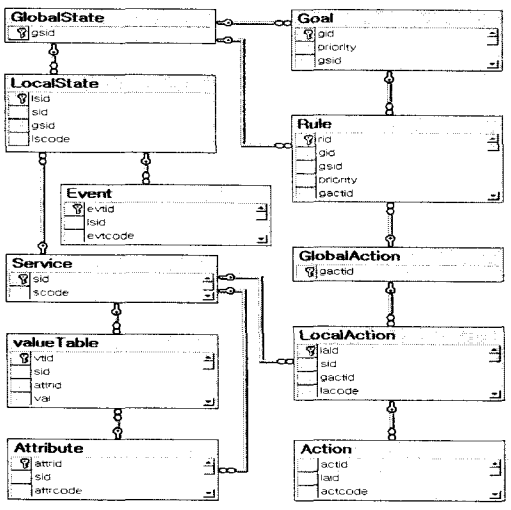

그림 4 RDB로 구현한 RASC 도메인

서비스 중개자에 연결을 만들 때, 컴포넌트가 아닌 XML 웹 서비스 구현이 가능하다. 다만 이 경우에는 EventRaised 함수의 구현을 위해 서비스가 어디에 해당 이벤트 메시지를 보내야하는지에 대한 연결 정보를 서비스 등록 시점에 알려주어야 한다. 즉, connectionString에 서비스 중개자의 접근에 필요한 정보를 미리 제공하면 된다. 그리고, 해당 서비스는 그 연결 정보를 지역값으로 계속 유지할 수 있어야 한다.

목표와 상태로부터 행위가 유도되는 규칙 함수는 비교적 단순한 구조로 복수 개의 메트릭스로 표현이 가능하기 때문에 관계형 데이터베이스로 구현이 용이하고, 캐쉬(Cache)를 사용하는 저장 프로시저(Stored procedure)와 클러스터드 인덱스 키(Clustered index key)를 이용할 경우 효율적인 서비스 조합 처리가 가능하다. 그림 4는 RASC 도메인을 구현하기 위해 관계형 데이터베이스를 사용할 때의 주요 테이블 및 기본키(Primary key)와 외래키(Foreign key)의 관계를 나타낸 것이다. 다이어그램을 통해 RASC 도메인의 형식과 테이블간의 관계가 거의 일치함을 확인할 수 있다.

표 7 RASC 도메인을 따르는 Jess 규칙의 예

```
(defrule rule220
  (f G100041)
  (or (or (or (f LS112002) (f LS112001)) (f LS001211) ) (f LS221021))
  (f priority ?n) (test (< ?n 1))
  =>
  (assert (A133110)))

(defrule rule221
  (f G100041)
  (f LS112000) (f LS001210) (not (f LS221021))
  (f priority ?n) (test (< ?n 1))
  =>
  (assert (A133111)))

(defrule rule338
  (f G100041)
  (f fLS112001)
  (f priority ?n) (test (< ?n 1))
  =>
  (assert (A120420)))

(defrule rule339
  (f G100041)
  (f LS112002) (f LS133110)
  (f priority ?n)
  (test (< ?n 1))
  =>
  (assert (A124310)))
```

ERP(Enterprise Resource Planning) 시스템이나 전자 상거래(e-Commerce) 분야에서 널리 사용되고 있는 제스(Jess)는 전문가 시스템(Expert systems)을 구성하기 위해 만들어진 규칙 기반 언어로, 클립스(Clips)에 뿌리를 두고 있다.[12]. 전문가 시스템은 규칙의 집합으로 구성되어 외부로부터 수집된 사실(Fact)을 바탕으로 이 규칙을 반복 적용 하는데, 제스는 규칙의 개수에 관계없이 효율적 지식기반(Knowledge-base) 체계를 구성할 수 있다. 이렇게, 제스는 규칙을 표현하고 관리하며 적용시키기 위한 도구로 사용될 수 있기 때문에 RASC 프레임워크의 규칙 관리자의 구현에 적합하다. 또한, 제스는 자바에서 쉽게 불러들여 사용할 수 있다. 자바 프로그램에서 제스를 사용하려면 jess.Rete를 임포트(import)하여 Rete 클래스의 인스턴스를 만들고, eval 메쏘드(Method)를 사용하여 제스의 규칙을 등록 할 수 있다. 규칙에 따라 발생하는 이벤트들을 처리하기 위해서는 JessListener 클래스를 상속받아 구현하면 된다. reset, batch 메쏘드를 사용할 경우 규칙들을 배치(batch)파일 형식으로 한꺼번에 처리할 수도 있다. 표 7은 제스로 구현한 규칙의 일부로 목표, 지역 상태들의 집합과 우선순위에 따라 행위들의 집합을 구할 수 있는 *Goal* 도메인 형식을 따르고 있다.

그림 5는 간단한 RASC 프레임워크의 시나리오를 보여준다. 공의 색, 벽을 감지할 수 있고, 목표지점에 도착했는지를 판단할 수 있는 서비스가 등록되어 있고, 공을 싣거나 파괴하거나 좌,우,앞으로 이동할 수 있도록 해주는 서비스를 가진 로봇이

미로를 통해 목표지점까지 가면서 발견된 붉은색 공은 파괴하고, 푸른색 공은 실어서 가지고 오기 위해 필요한 최소한의 지역 상태, 행위, 목표 상태 등을 보여준다. 전역 규칙의 일부는 표 7에 나와있다. 즉, 공도 없고, 장애물이 없으며, 목표지점까지 아직 도착하지 않은 상태라면 전진(Go), 만약 벽을 감지하게 되면 정지(Stop)한다. 정지된 상태(LS133110)에서 벽이 감지되면(LS001211) 로봇의 방향을 오른쪽으로 전환 시킨다(Turn_right). 오른쪽으로 회전이 완료된 상태(LS133113B)에서 푸른공을 발견하게(LS112001) 되면 케리어(Carrier)가 동작하여 정지한 상태라면 푸른공을 싣게 된다(Load_ball). 이와 유사한 방식으로 로봇은 미로를 탐사하게 되고, 목표지점에 도착할 때까지 규칙을 실행하게 된다. 이러한 로봇 외에도 가전제품들 간의 등록 상태에 따라 적합한 적응을 보이는 시나리오도 가능하다.

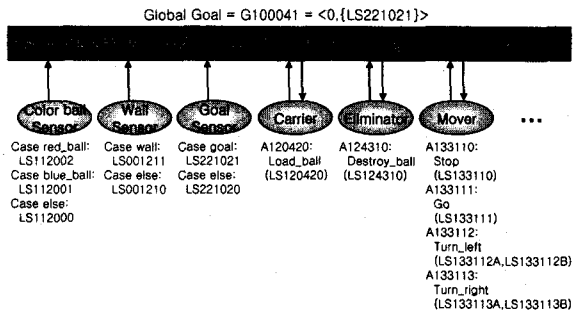

그림 5 응용 시나리오의 예

6. 결 론

본 논문에서는 규칙을 따르는 서비스의 동적인 조합을 시스템의 적응으로 정의하고 중개자 방식을 이용하여 이를 실현하는 RASC 프레임워크를 제안하였다. RASC 프레임워크는 블랙보드 아키텍처를 기반으로 규칙을 공유하고, 수정함으로써 진화할 수 있으며, 등록된 서비스 및 서비스 조합체들은 자극-반응 모델을 따르고 자기유사성을 가짐으로써 복잡계를 구성할 수 있다. 또한, RASC 아키텍처와 도메인을 정의하고 시나리오를 통한 구현 전략을 제시함으로써 실현 가능성을 보였다. RASC 도메인에서 사용하는 규칙 도메인은 상태와 이에 따른 행위라는 비교적 단순한 규칙들로 구성되기 때문에 변이 및 교차와 같은 유전자 알고리즘 연산을 적용하여 기존 규칙만으로 목표 달성이 어려울 경우 지역 최적점을 벗어날 수 있는 기회를 가질 수도 있다. 하지만, 아직 RASC 프레임워크의 구현이 규칙의 품질을 확인할 수 있을 정도로 완전하지 않기 때문에 향후 RASC 도메인 모델을 구현에 맞추어 상세화하고, 이를 구현한 시뮬레이션을 통해 프레임워크의 유효성을 검증해 보는 것이 필요하다. 이 과정에서 복잡계에서 보이는 창발(Emergence)효과에 대한 확인도 가능할 것으로 생각된다.

RASC 프레임워크는 서비스가 규칙을 서로 공유함으로써 계속적으로 진화할 수 있는 기본적인 틀을 제공한다. 즉, 적절한 평가를 통해 경쟁적이지 못한 규칙들은 사라지고, 반복 사용되거나 높은 평가를 받는 규칙들이 강화될 수 있다. 지속적인 규칙 변화를 통해 서비스들의 협업 관계와 반응이 동적으로 달라질 수 있고, 이를 통해 규칙 패턴을 학습하고, 진화할 수 있다. RASC 도메인 형식만 지킨다면 기계 학습을 위한 구현 알고리즘은 어떠한 것을 사용해도 상관이 없다.

하지만, 논문에서 제시한 프레임워크가 실제로 활용되기 위해서는 권한이나 위치 등과 같은 기타 상황 정보도 규칙을 실행할 때의 조건으로 입력될 필요가 있다. 이를 하나의 서비스로 등록하여 구현할 수도 있지만, 프레임워크 내부에 구현하는 것이 바람직하다. 이와 관련한 RASC 프레임워크 환경에 대한 연구는 앞으로도 계속 진행할 예정이다. 또한, 규칙들이 방대해지면, 스스로 규칙을 분석하여 패턴 추출이나 삭제, 병합 등을 통한 정리 작업을 할 필요도 있는데, 동적인 처리가 가능하도록 향후 구현할 계획이다. 원격지의 서비스 조합체들 간의 협업을 위해 에이전트(Agent)를 활용하는 방안도 연구중이다.

RASC 프레임워크는 다양한 가전 제품의 조합이 가능한 홈 네트워크나 부품의 결합에 따라 다양한 반응을 보이는 지능형 로봇, 기타 계속적인 확장 모듈의 추가가 가능한 대부분의 시스템에 적용이 가능할 것으로 기대된다.

7. 참고문헌

[1] J. Shen, Q. Wang, and H. Mei, "Self-adaptive Software: Cybernetic Perspective and an Application Server Supported Framework," *Proc. 28th Annual International Computer Software and Application Conference (COMPSAC'04)*, 2004.

[2] M. Kumar, B.A. Shirazi, S.K. Das, B.Y. Sung, D. Levine, and M. Singhal, "PICO: A Middleware Framework for Pervasive Computing," *IEEE Pervasive Computing*, vol. 2, no. 3, 2003.

[3] H. Cervantes and R.S. Hall, "Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model," *Proc. 26th International Conference on Software Engineering (ICSE'04)*, 2004.

[4] D. Garlen, S. Cheng, and B. Schmerl, "Increasing System Dependability through Architecture-based Self-repair," *Architecting Dependable Systems*, R. de Lemos, C. Gacek, A. Romanovsky (Eds), Springer-Verlag, 2003.

[5] S. Cheng, A. Huang, D. Garlan, and B. Schmerl, "Rainbow: architecture-based self-adaptation with reusable infrastructure," *IEEE Computer*, vol. 37, no. 10, Oct. 2004.

[6] S. Cheng, D. Garlan, and B. Schmerl, "Architecture-based Self-Adaptation in the Presence of Multiple Objectives," *Proc. ICSE international workshop on Self-adaptation and self-managing systems*, 2006.

[7] Q. Yang, X. Yang, and M. Xu, "A Mobile Agent Approach to Dynamic Architecture-based Software Adaptation," *ACM SIGSOFT Software Engineering Notes*, vol. 31, no. 3, May 2006.

[8] J.C. Georgas, A. Hoek, and R.N. Taylor, "Architectural Runtime Configuration Management in Support of Dependable Self-Adaptive Software," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, July 2005.

[9] J.M. Cobleigh, L.J. Osterweil, A. Wise, and B.S. Lerner, "Containment Units: A Hierarchically Composable Architecture for Adaptive Systems," *Proc. 10th ACM SIGSOFT symposium on Foundations of software engineering*, 2002.

[10] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture*, Wiley, 1996.

[11] 문병로, *유전알고리즘*, 두양사, 2003.

[12] Jess, http://www.jessrules.com/