OWL-S 기반의 시맨틱 웹 서비스를 이용한 동적 소프트웨어 재구성

이재정°, 김진한, 이병정
서울시립대학교 컴퓨터과학부
{leejaejung°, kimjinhan, bjlee}@uos.ac.kr

Dynamic Software Reconfiguration using Semantic Web Services based on OWL-S

Jaejung Lee°, Jinhan Kim, Byungjeong Lee
School of Computer Science, University of Seoul

요 약

최근의 비즈니스 환경은 고객, 공급자, 파트너 등 다수 기업과의 관계적 협업관계가 중시되는 수평적 통합환경으로 변화하고 있다. 이러한 관계적 협업이 중시되는 비즈니스 환경에서 경쟁력을 갖추기 위해서 기업은 급변하는 사용자와 시장 요구에 민첩하게 대응하고 적응해야 한다. 본 연구에서는 웹 서비스를 이용하여 다수의 협업관계를 가능하도록 동적 환경에서 배치와 재구성이 가능한 모델을 제시한다. 본 논문에서는 시맨틱 웹 서비스 기술을 이용하여 필요한 서비스들을 의미적으로 발견하고 동적으로 재구성한다. 웹 서비스 발견 및 재구성 프토토타입을 통하여 본 모델의 유효성을 보인다.

1. 서 론

최근의 비즈니스 환경은 과거 독립적인 조직 및 프로세스에 의해 주도되는 수직적 통합에서 고객, 공급자, 파트너 등 다수 기업과의 관계적 협업관계가 중시되는 수평적 통합환경으로 변화하고 있다. 이러한 관계적 협업이 중시되는 비즈니스 환경에서 경쟁력을 갖추기 위해서 기업은 급변하는 사용자와 시장요구에 민첩하게 대응하고 적응할 수 있어야 한다. 이것을 위한 기업 IT 아키텍처로 대표되는 SOC (Service-Oriented Computing)는 분산되고 상호 운용적인 비즈니스 구축을 위한 널리 퍼진 패러다임으로서 발전하였다 [1]. SOC는 기본적으로 SOA (Service-Oriented Architecture)에 의존한다. SOA 에서 말하는 서비스는 비즈니스 프로세스를 수행하는 소프트웨어 단위이며 각각의 서비스는 위치투명성과 상호연동을 보장하는 메시지 프로토콜을 통해 느슨하게 결합 (loosely-coupled) 한다 [2]. ESB (Enterprise Service Bus)를 활용하여 기능 중심의 단위 서비스를 연결, 조합하고 비즈니스 프로세스 기반의 처리를 지원한다. 대표적인 표준으로 BPEL (Business Process Execution Language)[3]이 있다.

제 3 의 기관에서 제공하고 운용하는 서비스에 대하여 신뢰도나 서비스의 질 (QoS), 성능에 대한 보증이 필요하다[4]. 전체 어플리케이션을 구성하는 각각의 서비스들은 그들이 변경되거나 또는 추가, 삭제가 되더라도 서로에게 영향을 미치지 않아야 한다. 이러한 가용성 (availability)과 확장성 (scalability)을 위하여 응용프로그램에 영향을 주지 않고 수정과 유지보수가 가능하여야 하고 소프트웨어가 재구성하는 동안 시스템의 중지와 재시작을 허락하지 않아야 한다.

재구성을 위한 요구사항에 맞은 서비스를 찾을 때, 그 서비스는 단일 서비스일 수 있으며, 또한 여러 서비스가 조합된 복합 서비스일 수 있다. 이들을 응용프로그램을 구성하는 수평적인 모듈로 이해하고 이들의 생명주기를 관리하는 방법을 본 논문에서 제시한다.

본 논문의 구성은 다음과 같다. 2장에서 시맨틱 웹 서비스 기술과 기존의 컴포넌트 기반 동적 재구성을 설명하고 3장에서 소프트웨어의 동적인 재구성 모델과 서비스 재구성 시나리오를 설명한다. 4장에서는 실행시간 재구성 모니터링 프로토타입을 설명한다. 그리고 5장에서 향후 과제와 결론을 기술한다.

2. 관련 연구

2.1 시맨틱 웹 서비스

OWL-S (OWL Web Ontology Language for Services)는 서비스를 위한 웹 온톨로지 (ontology) 언어이다. OWL-S의 목적은 웹 서비스에 관해서 추론과

본 연구는 21세기 프론티어기술개발사업인 인간기능생활지원 지능로봇기술개발사업단의 기술비 지원으로 수행되었음

서비스 조합을 위한 계획과 에이전트들에 의한 서비스의 이용의 자동화가 가능하게 한다. OWL-S를 사용하여 서비스 제공자들은 잠재적인 사용자 들에게 자신들의 서비스의 기능을 알릴 수 있다 [5, 6].

시맨틱 웹 서비스는 온톨로지를 활용하여 서비스를 기술하고, 온톨로지의 의미적 상호운용성을 이용해서 서비스 관련 제반 처리를 자동화 한다. 그 중 서비스 발견 (discovery)은 요청 (requirement)으로부터 만족하는 서비스를 찾는 과정이다. 이를 위한 프레임워크는 매치 메이킹 (matchmaking), 입출력 타입 매칭 (input/output types matching) 그리고 선조건과 효과분석 (precondition/effect analysis)으로 구성되어 자동화된 발견 서비스를 지원한다 [7, 8]. 그러나 온톨로지의 개념(concept)에 대한 의미적 매칭의 정도(degree) 와 순위(ranking)를 지원하지 못하는 단점을 가지고 있다.

2.2 컴포넌트기반 동적 재구성

동적 재구성은 어플리케이션을 중지하는 일 없이 어플리케이션을 구성하는 기능 유닛을 교체할 수 있는 메커니즘이다. 동적 재구성 기능은 높은 적응성과 가용성을 위한 시스템에서 매우 유용하다. 동적 소프트웨어의 재구성을 위한 연구들은 많이 진행되어왔다. 특히 전통적인 컴포넌트 기반의 아키텍처를 제시하는 많은 연구들이 제시되었다 [9, 10]. 이러한 컴포넌트 프레임워크는 컴포넌트들이 서로 통신하기 위한 컴포넌트 인터페이스를 제공한다. 하지만 특정 기술에 종속적이면서 상대적으로 강한 결합(tightly coupled)을 갖는 단점이 있다.

기존의 컴포넌트 기반의 연구들은 주로 비기능적인 요구사항을 고려한 재구성 방법들이 제시되었고, 기능적인 부분에서는 기능을 명세하기 위한 방법과 대체 자원의 부재 시, 동적 재구성를 달성하지 못하는 문제들을 가지고 있다. 웹 서비스를 활용하는 경우에도 인터페이스의 역할을 하는 웹 서비스 컴포넌트의 경우 느슨한 결합을 지원 하지만, 비즈니스 로직을 가지고 있는 클래스와는 상대적으로 강한 결합을 갖는 단점이 있다 [11].

3. 소프트웨어의 동적인 재구성

3.1 소프트웨어의 동적인 재구성 모델

본 논문에서는 소프트웨어 아키텍처 레벨에서 실행시간 중 동적인 재구성 기능에 관해 논의한다. 소프트웨어의 동적인 배치와 재구성 능력은 응용프로그램을 이루는 서비스가 실행 중에 추가, 제거, 교체되는 것이다. 이 논문에서는 또한 소프트웨어의 일관성 (consistency)을 유지하는 메커니즘을 소개한다. 그리고 원자성 (atomicity)을 보장하는 동적인 재구성을 위해 2PC (two-phase commit protocol)을 이용한다.

그림 1은 응용프로그램 실행 중에 동적인 소프트웨어 재구성을 위한 모델을 보여준다. 그림은 두 부분으로 나뉘는데 첫 번째 부분은 결정관리자 (decision manager)[12], 재구성관리자 (reconfiguration manager), 프록시 에이전트 (proxy agent), 버전관리자 (version manager), 요청 에이전트 (request agent)[12]로 구성되는 동적 재구성 프레임워크의 모델이다. 두 번째 부분은 응용프로그램을 구성하는 동적인 재구성의 대상이 되는 서비스 모듈이다. 각각의 서비스는 서비스 제공자들에 의해 분산되어 위치한다.

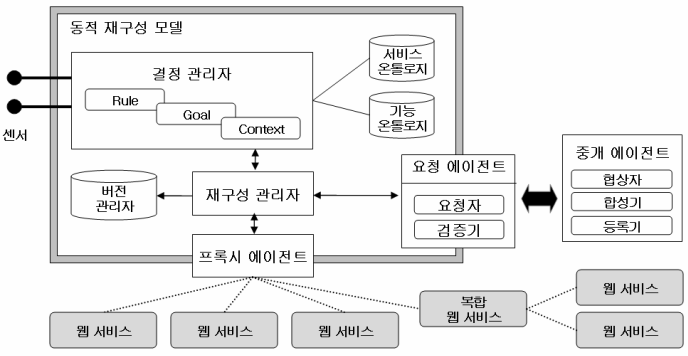


그림 1. 동적 재구성 모델

- **결정관리자**: OWL-S는 특정한 기능을 수행하는 서비스의 수행절차 및 제약조건을 명시한다. 하나 혹은 여러 개의 서비스가 조합되어 더 큰 단위의 비즈니스 프로세스를 제공하고 결정관리자에 의해서 실행된다. 내부에는 규칙 (rule), 상황정보 (context), 목표 (goal)를 가진다. 규칙은 어떠한 조건에 의해서 실행되는 기능들의 관계들의 집합이다. 상황정보는 규칙에 의해 실행되는 기능에 의해 값이 변하는 사용자 시스템의 속성 값들의 전 상태와 현 상태정보를 말한다. 그리고 목표는 속성 값들이 유지하고자 하는 상태를 정의한다. 또한 서비스의 기능과 요구사항에 필요한 개념 (concept)을 기능 온톨로지에 저장하고, 재구성하는 서비스에 대한 개념은 서비스 온톨로지에 저장한다.
- **재구성관리자**: 결정관리자에서 재구성에 대한 요청이 들어오면 우선 버전관리자를 통해 이전 서비스들의 목록을 조사한 후 외부로부터 서비스를 도입 유무를 결정한다. 이후 외부로부터 서비스를 도입하는 경우 요청 에이전트의 요청자를 통해 찾고자 하는 서비스의 명세를 전달한다. 또한 비즈니스 프로세스에 참여하는 서비스들을 실행 시간 중에 동적으로 재구성한다. 그리고 실행시간 재구성 트랜잭션 (runtime reconfiguration transaction)의 일관성을 보장하기 위해서 수행결과를 최종적으로 완료 (commit) 또는 복귀 (rollback)을 결정하는

역할을 한다. 서비스를 재구성한 후 재구성된 서비스의 버전 정보를 버전관리자에 등록한다.

- **프록시 에이전트**: 결정관리자는 서비스를 호출할 때 직접적으로 서비스의 끝 지점 (end-point)을 실행하지 않고 프록시 에이전트를 통하여 로컬 컴퓨터의 서비스를 호출하는 것처럼 수행한다. 프록시 에이전트는 버전관리자에 의해서 관리되는 실제 서비스를 호출하는 역할을 한다. 만약 실행시간 중에 서비스의 재구성 요청이 발생할 때 프록시 에이전트와 연결된 서비스가 바뀌게 된다.
- **버전관리자**: 서비스 모듈의 버전정보를 유지하고 소프트웨어의 형상관리 기능을 수행 한다. 만약 소프트웨어가 동적인 재구성을 수행하는 도중에 실패를 하면, 이전 버전의 서비스로 되돌릴 수 있는 정보를 제공한다.
- **요청 에이전트**: 시스템의 요구사항 (requirement)을 받아서 중개 에이전트 (Broker Agent)로 전달하는 역할을 가지고 있다. 요구사항은 시스템이 요구하는 기능을 기술하는 온톨로지의 개념과 IOPE (Input, Output, Precondition, Effect)같은 서비스를 찾기 위해 필요한 정보들을 명세 한다. 그리고 외부로부터 가져온 서비스에 대해 동적인 재구성을 수행하기 이전에 서비스의 유효성을 검증하는 역할을 한다.
- **중개 에이전트**: 요청 에이전트와 제공자 에이전트의 요청을 처리하는 에이전트이다. 협상자는 사용자의 요구사항을 만족하는 서비스를 찾기 위해 온톨로지의 개념정보를 이용한다. 요청을 만족하는 개념을 찾고, 그 개념에 대해서 참조를 가지는 웹 서비스를 찾는다. 검색의 결과로 후보 서비스들을 찾아서 최종 서비스를 선택한다. 이 때 선택된 서비스는 하나의 서비스일수도 있고, 여러 서비스가 조합된 복합 서비스일 수도 있다. 합성자는 최종적으로 선택된 서비스들의 조합을 기술한다. 이 내용에는 서비스 조합에 대한 순서와 제어, 그리고 서비스 간의 메시지 변환에 대한 내용 등이 표현된다. 표현 방식은 OWL-S의 프로세스 모델을 사용하여 나타낸다.

그림 2는 사용자가 필요로 하는 서비스를 요청하기 위한 요청 설명(request-description)의 예를 보여준다. 어떤 도시의 이름을 입력하면 해당 도시의 기후정보를 제공하는 서비스이다. serviceProduct 요소(element)와 serviceClassification 요소는 각각 UNSPSC와 NAICS의 명세서를 따르고, 이 둘은 profile로부터 서비스의 OWL 온톨로지로 매핑할 수 있는 정보를 제공한다. 또한 serviceCategory는 서비스 매칭의 좀 더 제한적인 검색을 도와준다. hasInput, hasOutput, hasPrecondition 그리고 hasResult는 찾고자 하는 서비스의 IOPE를 표현한다. IOPE 정보에는 온톨로지의 URI를 명시한다. 이로 인해 IOPE를 구성하는 요소가 의미적으로 무엇인지를 온톨로지를 통해 알 수 있다.

```
<profile:Profile rdf:ID="request">
<profile:serviceProduct rdf:ID="example.owl₩climate">
</profile:serviceProduct>
<profile:hasInput rdf:ID="example.owl#CityName">
</profile:hasInput>
<profile:hasOutput rdf:ID="example.owl#Temperature">
</profile:hasOutput>
<profile:hasPrecondition
rdf:ID="example.owl#validCityName">
</profile:hasPrecondition>
<profile:hasResult rdf:ID="example.owl#Celsius">
</profile:hasResult>
<profile:textDescription>
(weight{ I=1; O=1; P=1; E=1;})
</profile: textDescription>
```

그림 2. Request description of OWL-S based

요청 설명에 명시된 IOPE 의 타입과 서비스 매핑에 참여하는 서비스들의 타입 매치 정도는 [12]의 연구에서 제안된 방법을 수정하여 적용한다. 이 방법은 온톨로지의 매치의 정도에 따라 Exact, Subsume, Relaxed 그리고 Fail로 나누어 정수 값을 할당한다. textDescription은 IOPE의 각각 구성요소에 가중치를 조건으로 명세할 수 있다. 이는 서비스를 요청하는 소비자 측면에서 각자에 맞은 환경의 제한을 서비스의 검색 조건에 명시할 수 있다.

3.2 소프트웨어의 동적인 재구성 단계

그림 3은 소프트웨어의 동적인 재구성 트랜잭션 과정을 나타낸다.

1. 외부의 이벤트나 사용자의 요청에 의해 재구성 요청이 발생한다. 결정관리자는 재구성관리자에게 실행시간 재구성 트랜잭션 수행을 명령한다.
2. 재구성관리자는 요청 에이전트에게 새로운 서비스의 탐색을 요청하다. 그러면 중개 에이전트를 통해 외부 환경으로 접근이 이루어지며 적절한 서비스의 제공자에 대한 참조를 얻는다.
3. 만약 복합 서비스일 경우 그들의 프로세스가 유효한지를 요청 에이전트의 검증기를 통해 확인한다. 유효성 검증에 실패한다면 런타임 재구성 명령은 무효화 된다.
4. 재구성관리자는 재구성 트랜잭션을 준비한다.
5. 유효성 검증이 성공한다면 재구성관리자는 실행시간 재구성 트랜잭션을 시작한다.
6. 재구성관리자는 새로운 서비스의 핸들을 프록시 에이전트에 갱신할 것을 요청한다.

7. 프록시 에이전트는 이전 서비스에 대한 참조를 새로운 서비스로 바꾼다. 그리고 수행 결과를 재구성관리자에게 알려준다
8. 재구성관리자는 수집한 결과를 이용하여 최종적으로 실행시간 재구성 트랜잭션의 완료 또는 복귀를 결정한다.
9. 실행시간 재구성 트랜잭션이 완료한다면 새로 참조되는 서비스의 정보를 버전관리자에 저장하고, 복귀한다면 이전의 서비스 구성으로 되돌아 간다.

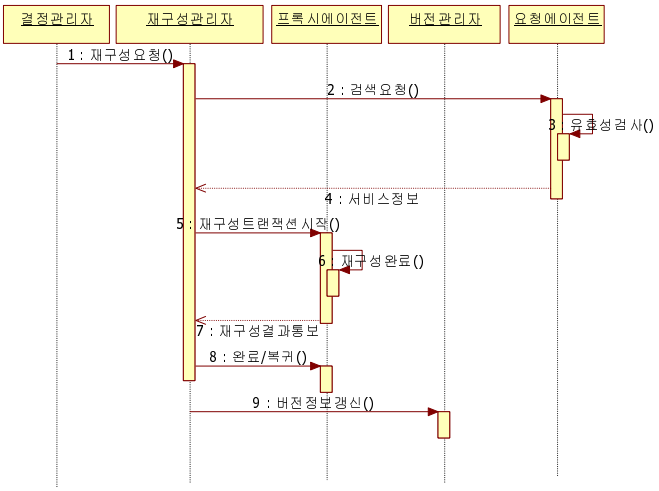


그림 3. 동적 재구성 단계

재구성관리자는 한 개 혹은 여러 개의 서비스를 동시에 재구성 할 수 있다. 만약 여러 개의 서비스를 재구성할 경우에 소프트웨어 갱신의 일관성(consistency)을 보장해 주어야 한다. 그리고 갱신 도중에 발생하는 서비스 호출에 대해 중지 없는 서비스(non-stop service)를 보장해 주어야 한다. 그림 4는 프록시 에이전트가 새로운 서비스를 재구성하는 단계를 보여준다.

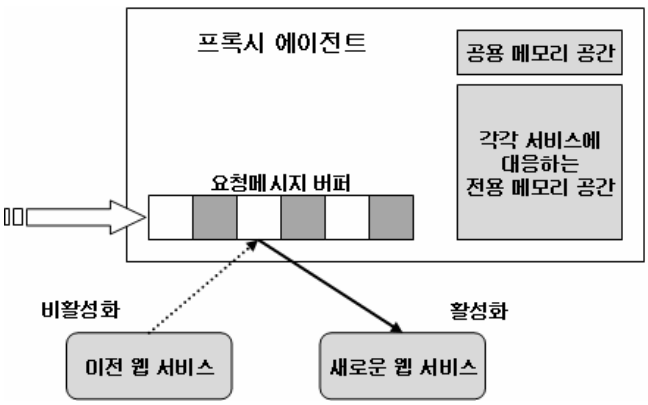


그림 4. 프록시 에이전트의 서비스 재구성 단계

1. 이전의 서비스의 상태를 버전관리자에게 비활성화를 요청한다.
2. 결정관리자로부터 들어오는 서비스 호출 요청을 내부 서비스 요청 큐 (queue)에 저장한다.
3. 프록시 에이전트는 새로운 서비스에 대한 핸들을 등록하고 버전관리자에게 활성화를 요청한다.
4. 프록시 에이전트는 재구성관리자에게 갱신 결과를 보고하고 최종 완료 요청을 기다린다. 만약 재구성관리자로부터 실행시간 재구성 트랜잭션의 완료 통지를 받으면 서비스 요청 큐의 메시지를 새로운 서비스에게 전달한다.
5. 재구성관리자는 새로운 서비스의 버전 정보를 버전관리자에 저장한다.

프록시 에이전트의 공용 메모리 공간 (public memory space)은 모든 서비스 모듈들이 서로 공유하는 지식을 저장하는 공간이며, 전용 메모리 공간 (private memory space)은 각각의 서비스에 대응하는 전용 지식에 대한 저장 공간이다. 이 에이전트로 들어오는 메시지는 모두 메시지에 대응하는 서비스에 대한 고유한 키 값을 명시한다.

4. 사례연구

4.1 시나리오

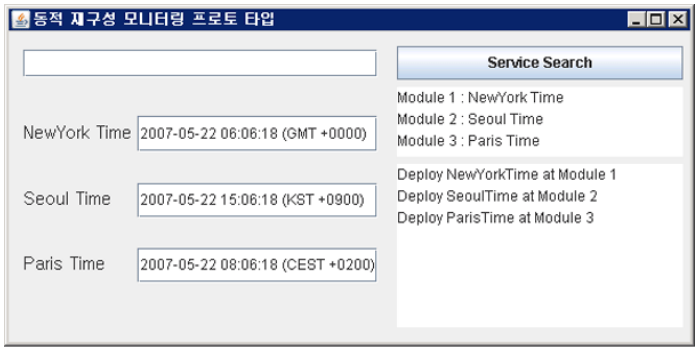
앞장에서 설명한 실행시간에 서비스 재구성을 모니터링 할 수 있는 프로토타입 수준의 모델을 자바언어로 구현하였다. 프로토타입을 위해 아래와 같은 구현 요구사항을 가정한다.

1. 서비스 제공자에 의해서 미리 구현된 분산된 서비스를 사용한다.
2. 결정관리자는 3개의 서비스 모듈로 이루어진 비즈니스 프로세스를 수행한다.
3. 각각의 서비스 모듈은 계속적으로 발생하는 이벤트에 의해 서비스가 호출되고 각각의 수행결과 정보를 되돌려준다.
4. 사용자 요구사항에 적합한 새로운 서비스를 검색하고 대체하여 수행시킨다.

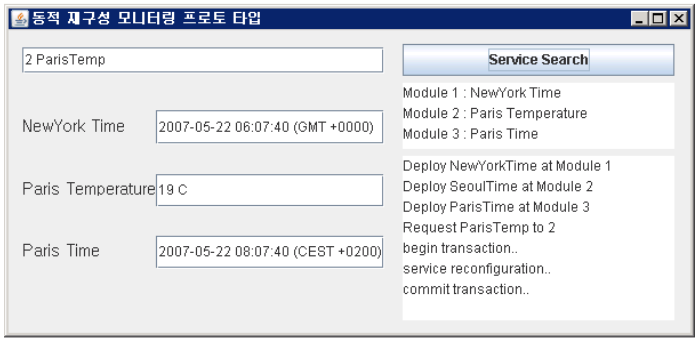
4.2 프로토타입

본 연구에서 이용되는 서비스 모듈은 특정 지역의 시간대를 이용한 현재시각을 알려준다. 그리고 새로 재구성되는 서비스는 특정 지역의 현재기온을 알려주는 서비스를 이용한다. 재구성 수행 명령을 위해서 갱신하고자 하는 모듈의 번호와 새로운 서비스의 이름을 사용한다. 그림 5는 본 논문에서 구현한 동적 재구성 모니터링의 프로토타입 화면이다. 화면 왼편은 결정관리자에 의해서 수행되는 서비스들이다. 화면

오른편의 위쪽은 현재 구성된 서비스 모듈의 버전정보와 서비스 이름을 나타내고 아래쪽은 서비스 재구성 중에 발생하는 응용프로그램 메시지 정보를 보여준다.



(a). 재구성 전



(b). 재구성 후

그림 5. 동적 소프트웨어 재구성

그림 5의 (a)는 재구성 전의 응용프로그램의 구성을 보여준다. 그리고 (b)는 동적 재구성 완료 후의 응용프로그램의 구성을 보여준다. 재구성 요청의 결과 2번째 서비스 모듈이 프랑스 파리의 현재 기온을 알려주는 새로운 서비스로 재구성 완료 하였다.

5. 결론 및 향후 연구

SOC는 인터넷 환경의 응용프로그램들을 위한 주목을 끄는 컴퓨팅 패러다임이 되었다. SOC는 이미 존재하는 서비스들을 위한 부가적인 기능을 제공한다는 점에서 큰 관심을 끌고 있다. 본 논문에서는 SOC의 주체인 웹 서비스를 이용하여 동적인 환경에서 서비스들을 배치하고 재구성을 위한 기법을 소개하였다. 전체 응용프로그램을 구성하는 각각의 서비스들은 변경되거나 또는 추가, 삭제가 되더라도 서로에게 영향을 미치지 않도록 서비스 들을 수평적으로 독립시켰다. 또한 다양한 사용을 위하여 응용프로그램에게 영향을 주지 않고 수정과 유지보수가 가능하도록 하였으며 만약 재구성 중 실패 하더라도 기존의 구성에 대한 이력정보를 이용하여 복귀가 가능하도록 제안하였다.

동적 재구성에 대한 범위를 모듈의 갱신이나 새로운 모듈의 추가로 제한하였고, 새로운 모듈에 대한 검증 작업이 없어 잦은 복귀 현상이 발생할 수 있다. 또한 모듈의 갱신일 경우 갱신되는 새로운 모듈이 이전 모듈에 비해 많은 변화가 있는 경우 상태 정보를 그대로 사용할 수 없다는 문제도 생길 수 있다.

향후 연구로는 사용자로부터 얻는 요구사항으로부터 구체적인 서비스 명세를 얻은 방법에 대한 연구와 본 논문에서 제안하는 모델의 전체적인 프로토타입 개발이 과제로 남는다.

참고 문헌

[1] OASIS, Reference Model for Service Oriented Architecture 1.0, Aug. 2006.

[2] 한상우, 박선희, 노재호, “Service Oriented Architecture 적용을 위한 서비스 식별 기법,” 한국정보과학회 학회지, 제 24권 제11호, pp. 27-31, Nov. 2006.

[3] T. Andrews, et. al., Business Process Execution Language for Web Services, Version 1.1 BPEL4WS specification, May 2003.

[4] S. Chatterjee and J. Webber, Developing Enterprise Web Services: An Architect's Guide, Prentice Hall PTR, Nov. 2003.

[5] M. P. Singh and M. N. Huhns, Service-Oriented Computing, John Wiley & Sons, Ltd., 2005.

[6] N. Milanovic and M. Malek, “Current solutions for web service composition,” IEEE Internet Computing, Vol. 8, Iss. 6, pp. 51-59, 2004.

[7] D. Fensel and C. Bussler, “The Web Service Modeling Framework,” Proc. of International Semantic Web Conference, 2002.

[8] S. Chaiyakul, K. Limapichat, A. Dixit and E. Nantajeewarawat, “A Framework for Semantic Web Service Discovery and Planning,” Proc. of IEEE Conference on Cybernetics and Intelligent Systems, pp. 1-5, June 2006.

[9] Y. Qun, Y. Xan-Chun and X. Man-Wu, “A Framework for Dynamic Software Architecture based Self-healing,” ACM SIGSOFT Software Engineering Notes, Vol. 30, July 2005.

[10] J. Malek, M. Laroussi and A. Derycke, “A Middleware for Adapting Context to Mobile and Collaborative Learning,” Proc. of Annual IEEE International Conference on Pervasive Computing and Communications Workshops, pp. 221-225, Mar. 2006.

[11] 임철홍, “서비스 컴포넌트 아키텍처,” 한국정보과학회 학회지, 제 24 권, 제 11 호, pp. 33-39, Jan. 2007.

[12] R. Gue, J. Le and X. Xia, “Capability Matching of Web Services Based on OWL-S”, Proc. Of the 16th International Workshop on Database and Expert Systems Applications, 2005.