KIPS Tr. Software and Data Eng.
Vol.4, No.3 pp.111~120 pISSN: 2287-5905

http://dx.doi.org/10.3745/KTSDE.2015.4.3.111

A Reusable Adaptation Strategy Extraction System for Developing Self-Adaptive Systems

Jungsik Nam† · Sukhoon Lee†† · Doo-Kwon Baik†††

ABSTRACT

Recently, self-adaptive system researches have been done to solve the problems occurred in the dynamic environment. Designing requirement in the self-adaptive system is necessary to recognize and solve the problem for the system, and if a developer reuses existing adaptation strategy to design the requirement, the designing time and cost would be reduced. Therefore, this paper proposes the system which extracts reusable adaptation strategy from the existing self-adaptive system. For the proposal, this paper conceptualizes the self-adaptation elements, defines the adaptation strategy ontology and target system ontology, and presents the process of extracting reusable strategy. This paper also implements proposed system and evaluates the reuse rate of the extracted strategy. As a result, the adaptation strategies extracted by proposed system are exactly operated, and the extraction method of proposed system shows higher reuse rate than a previous method.

Keywords : Self-Adaptive System, Adaptation Strategy, Ontology, Knowledge Reuse

자가 적응 시스템의 개발을 위한 재사용 가능한 적응 전략 추출 시스템

남 정 식† · 이 석 훈†† · 백 두 권†††

요 약

최근 동적인 환경에서 발생하는 다양한 문제를 스스로 해결할 수 있는 자가 적응 시스템에 대한 연구가 활발히 이루어지고 있다. 자가 적응 시스템에서 시스템이 문제를 스스로 인식하고 자가 적응할 수 있도록 요구사항을 설계하는 과정은 필수적이며, 만약 기존의 적응 전략들을 재사용하여 자가 적응 시스템을 설계한다면 소요되는 시간 및 비용을 절감할 수 있다. 따라서 이 논문은 새로운 자가 적응 시스템 개발 시 기존의 자가 적응 시스템으로부터 재사용 가능한 적응 전략을 추출하는 시스템을 제안한다. 이를 위하여 자가 적응 요소를 지식화하여 적응 전략 온톨로지 및 타깃 시스템 온톨로지를 정의하고, 이러한 온톨로지를 기반으로 재사용 가능한 적응 전략을 추출하는 기법을 기술한다. 또한, 이 논문은 제안 시스템을 구현하고 추출된 적응 전략에 대한 재사용률을 측정함으로써 제안 시스템을 비교 평가한다. 평가 결과, 제안 시스템은 추출된 적응 전략이 정확히 동작함을 보이며 제안 시스템의 추출 기법은 기존의 재사용 기법보다 높은 재사용률을 보인다.

키워드 : 자가 적응 시스템, 적응 전략, 온톨로지, 지식 재사용

1. 서 론

오늘날 소프트웨어 시스템의 규모는 점차 커지고 있으며 다양한 환경에서 구동되고 있다. 동작 환경이 다양해지고 끊임없이 변화하기 때문에 시스템은 예측하지 못한 상황에 놓이게 되며 이로 인해 다양한 문제가 발생한다. 이러한 문제점들을 소프트웨어 개발자가 개발 단계에서 모두 정의하기는 불가능하다. 따라서 시스템 스스로가 동적으로 변화하는 주변 상황에 대응하여 현재 발생한 문제를 인지하고 해결하는 자가 적응 시스템(Self-Adaptive System)에 대한 연구가 이루어지고 있다[1-2].

자가 적응 시스템은 일반적으로 폐쇄된 제어 루프(Closed Control Loop) 형태의 MAPE-K 모델에 기반한다. MAPE-K

※ 이 논문은 2015년 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업의 지원을 받아 수행된 연구임 (NRF-2012M3CA7033346).
† 정 회 원 : 고려대학교 컴퓨터 · 전파통신공학과 석사
†† 준 회 원 : 고려대학교 컴퓨터 · 전파통신공학과 박사과정
††† 종신회원 : 고려대학교 컴퓨터 · 전파통신공학과 교수
Manuscript Received : November 27, 2014
First Revision : January 13, 2014
Accepted : January 26, 2015
* Corresponding Author : Doo-Kwon Baik(baikdk@korea.ac.kr)

모델은 지식(Knowledge)을 이용하여 감시(Monitor), 분석(Analyze), 계획(Plan), 실행(Execute) 단계를 순환적으로 실행하며 시스템을 스스로 적응시킨다[3-5]. 시스템은 자가 적응을 위하여 주변 환경을 감시하고, 문제 발생을 인식 및 분석하며, 문제에 적응할 수 있도록 계획하여, 스스로 적응 전략을 실행한다. 지식은 주변 환경 정보, 시스템의 상태, 적응 전략 등을 나타내며 모든 단계에서 사용된다. 한편, 자가 적응 시스템은 그 동작 환경이 다양해지고 동적인 환경 변화에 적응하기 위하여 다양한 영역에서 개발되었다. Zanshin[6], Rainbow[7], DiVA[8], MADAM[9]과 같은 연구들은 각각 그 연구 영역과 적응 과정이 조금씩 다르지만 모두 MAPE-K 모델을 기반으로 하고 있다.

일반적으로 비자가 적응 시스템(Non-Self-Adaptive System)의 개발은 요구사항 분석 및 기능 명세, 아키텍처 설계, 구현 및 테스트, 배치 및 유지보수와 같은 프로세스를 거친다[10]. 자가 적응 시스템의 개발 프로세스는 대부분이 비자가 적응 시스템과 유사하지만 시스템이 자가 적응을 하기 위한 부분이 추가적으로 필요하다. 특히, 어떤 상황에서 어떻게 적응을 할지에 대한 적응 전략은 비자가 적응 시스템에서는 고려되지 않는 부분이므로 적응 전략 수립을 위하여 추가적인 비용이 소요된다. 즉, 한번 개발되었던 어떤 시스템을 자가 적응이 가능하도록 변경하는 것은 일반적인 소프트웨어 개발 과정과는 별개로 적응 전략을 정의하고 이를 적용시키기 위한 과정이 필요하다.

이와 같이 자가 적응 시스템에서 시스템이 스스로 문제를 인식하고 자가 적응을 하도록 설계하기 위한 요구사항을 정의하는 과정은 필수적이다. 하지만 앞서 언급한 자가 적응 시스템에 관련된 연구들은 지식이라 할 수 있는 주변 환경 정보, 시스템 상태, 적응 전략 등에 대하여 저장하거나 관리하기 위한 부분은 고려하지 않으므로, 적응 전략을 위한 요구사항 정의에 많은 비용을 초래한다. 실제로 자가 적응 시스템 개발 시 요구사항 정의를 위해 소요되는 시간은 전체 시스템 구축의 20% 이상을 차지한다[7]. 만약 기존에 정의된 적응 전략들을 자가 적응 시스템을 개발하고자 할 때 재사용하게 된다면, 개발 시 소요되는 시간 및 비용을 절감할 수 있다. 하지만 실제로 이와 같이 재사용을 고려하기 위해서는 재사용이 가능한 형태로 자가 적응 전략에 연관된 지식을 지식화하여 저장하고 관리하는 추가적인 지식베이스가 필요하다.

따라서 이 논문은 자가 적응 시스템 개발 시 기존의 정의된 적응 전략 중 재사용 가능한 전략을 추출하는 시스템을 제안한다. 이를 위하여 자가 적응과 연관된 적응 전략과 시스템을 개념화 및 지식화하여 적응 전략 온톨로지와 타깃 시스템 온톨로지로 정의한다. 제안 시스템은 정의된 온톨로지들을 이용하여 적응 전략 추출기를 거쳐 재사용 가능한 전략들을 추출한다. 실험을 통하여 제안 시스템에서 추출된 적응 전략의 재사용률을 평가하고, 이를 이용하여 자가 적응 시스템을 개발하였을 때의 시스템 동작을 검증한다.

단 이러한 환경을 위하여 제안 시스템은 기존의 자가 적응 시스템들을 개념화하여 온톨로지 형태로 지식베이스에서 관리하며, 개발하고자 하는 자가 적응 시스템 역시 개념화하여 온톨로지 형태로 변경한다고 가정한다. 즉, 기존의 자가 적응 시스템에 관련된 연구들은 각각 Stitch[11], GORE[12]와 같은 언어들을 이용하여 적응 전략 및 시스템을 표현하였지만, 이 논문에서는 각 시스템들의 공통적인 개념 표현 및 추론 등을 위하여 온톨로지로 변환하고 온톨로지 간의 비교를 통하여 재사용 가능한 적응 전략을 추출한다. 개발자들은 추출된 적응 전략을 이용하여 Rainbow, Zanshin 등과 같은 각 자가 적응 플랫폼에 맞추어 요구사항을 설계할 수 있다. 이는 실제 추출된 적응 전략들을 선별하여 재사용함으로써 요구사항을 설계할 때 소요되는 비용을 감소시킬 수 있으며, 더 다양한 문제에 대한 적응 전략을 설계할 수 있도록 도움을 준다.

이 논문의 구성은 다음과 같다. 2절에서는 관련 연구들의 특징과 한계점을 설명하고, 3절에서는 기존의 시스템에서 적응 전략에 연관된 지식을 관리하기 위한 적응 전략 온톨로지, 타깃 시스템에 대한 지식을 관리하기 위한 타깃 시스템 온톨로지, 그리고 적응 전략 추출 단계를 기술한다. 4절에서는 구현 및 평가를 통해 제안 시스템의 효용성을 입증하며, 5절에서는 결론 및 향후 연구를 기술한다.

2. 관련 연구

이 절에서는 기존의 자가 적응 소프트웨어의 연구들을 지식 관리의 관점에서 기술한다. 또한 지식을 재사용하기 위한 연구를 기술하며, 이 논문에서 적응 지식을 표현하기 위하여 기반하고 있는 선행 연구인 Rainbow에 대해 기술한다.

2.1 자가 적응 소프트웨어

자가 적응 소프트웨어는 공통적으로 적응을 위한 목표를 만족시키기 위하여 요구사항 정의에 초점을 맞춘다. 각 자가 적응 소프트웨어에 관한 연구들은 적응 전략에 대한 요구사항을 각각의 언어 및 모델들을 이용하여 정의한다. 이러한 언어 및 모델들은 각 자가 적응 프레임워크에 적용되어 지식을 표현한다.

Zanshin은 요구사항 기반의 프레임워크로서 자가 적응 행동을 유발시킬 수 있는 주변 환경 정보, 그에 따른 대처 방안인 적응 전략을 GORE로 표현한다[12]. GORE는 시스템 상태가 명확히 명시된 목표(Hard goal), 단순한 제약사항만 명시된 목표(Soft goal), 그리고 GORE 모델이 항상 만족되어야 하는 조건(AwReq)으로 구성된다. 시스템은 개발 단계에서 이러한 GORE 모델에 기반하여 자가 적응 요소들과 함께 개발된다.

Rainbow는 아키텍처 기반의 프레임워크로서 MAPE-K에 해당되는 컴포넌트들로 구성된다[13]. 자가 적응을 수행하기 위한 요구사항들은 모델 관리기(Model Manager), 적응 관리자(Adaptation Manager)에 의해 관리된다. 모델 관리기는 아키텍처 모델링 언어인 ACME로 표현된 시스템의 구조, 시스템 현재 상태에 대한 정보를 관리한다. 적응 관리자는

자가 적응 언어인 Stitch를 이용하여 적응 전략에 대한 정보를 관리하며, 시스템의 상태에 문제가 있다고 판단되었을 경우 적합한 적응 전략을 선택한다.

CASA는 자원의 식별과 동적으로 적응 행동을 수행하기 위한 통합적인 프레임워크이다[14]. 자가 적응을 위한 요구사항들을 CSL(Contract Specification Language)로 표현된 계약서(Contract)를 이용하여 명시한다. 응용 프로그램에 따라 각기 다른 계약서가 정의되며 이 계약서는 자원 관리기(Resource Manager)에 의해 관리된다.

하지만 이러한 자가 적응 소프트웨어 연구들은 자가 적응을 위한 요구사항 및 지식 관리 측면에서 기존에 정의된 지식들을 공유하거나 재사용하기 위한 방법을 고려하고 있지 않다. 이는 새로운 자가 적응 시스템을 개발하거나 기존의 비자가 적응 시스템을 자가 적응이 가능한 시스템으로 개발하고자 할 때 매번 새로 정의해야 하는 문제를 지닌다.

2.2 지식 재사용 기법

지식 재사용을 위한 기법들은 주로 지식을 표현하는 온톨로지를 재사용하기 위한 연구들로 이루어진다. 이러한 연구들은 문서들로부터 온톨로지를 추출하고 각 온톨로지 간 비교를 통하여 유사한 온톨로지를 추출하는 방법을 이용한다.

[15]는 지식의 공유를 위해 온톨로지 간의 유사도를 측정하는 방법을 제안한다. Levenshtein 거리[16]에 기반하여 문자열 매칭을 통한 두 온톨로지 간의 어휘적 유사도를 계산한다. 측정된 유사도는 기존 온톨로지에서 재사용 가능한 외부 온톨로지를 찾기 위해 사용된다.

CORE는 온톨로지의 재사용 및 평가를 위한 아키텍처이다[17]. 문서로부터 추출된 단어를 워드넷의 동의어, 상위어, 하위어 관계를 통해 확장하며, 확장된 단어와 지식베이스에서 관리하고 있는 온톨로지와의 어휘적, 구조적 유사도를 계산한다. 최종적으로 유사도순으로 온톨로지를 추천한다.

[18]은 지식의 재사용을 지원하는 아키텍처이다. 먼저 Stemming, POS-tagging, Stopword 제거 등 문서의 전처리를 수행한 후 클래스, 속성들을 분류하여 온톨로지 형태로 변환한다. 변환된 온톨로지와 지식 저장소에서 관리되고 있는 온톨로지를 문자열, 그래프 분석, 시맨틱 분석을 사용하여 비교하며 최종적으로 기존의 온톨로지와 가장 부합하는 온톨로지를 지식베이스로부터 추출한다.

이러한 연구들은 외부 온톨로지를 재사용함으로써 기존의 온톨로지를 확장한다. 하지만 이러한 기법들은 유사도 비교를 통하여 온톨로지 전체를 추천하기 때문에 온톨로지를 구성하는 클래스, 속성, 관계들 중 구체적으로 재사용할 수 있는 요소들까지는 추천하지 않는다.

2.3 Rainbow 프레임워크

이 논문은 자가 적응 시스템에 연관된 지식을 온톨로지 스키마로 정의하기 위해 Rainbow 프레임워크를 참조한다. 그 이유는 Rainbow 프레임워크는 적응 전략과 시스템 구성 요소들 간의 관계가 명확하고 직관적으로 구성되어있으며 재사용을 위해 적응 전략을 개념 요소 단위로 분류하기에 적합하기 때문이다.

Rainbow 프레임워크는 MAPE-K 모델을 이용하며 감시, 분석, 계획, 실행 과정을 통해 시스템에 적응 행동을 수행한다. 또한 Rainbow 프레임워크는 적응 전략을 표현하고 적응 행동을 수행하기 위해 ACME와 Stitch를 사용한다. ACME는 아키텍처 기술 언어로서, 타깃 시스템을 구축하기 위한 설계 단계에서 사용된다[19]. 시스템을 구성하는 요소(Component), 구성 요소들의 상호 작용을 가능하게 하는 연결기(Connector), 구성 요소의 속성값(Property) 등으로 구성되어있다. Rainbow에서는 시스템의 제약사항이 위반된 것을 평가할 수 있는 구성 요소의 속성값들을 참조한다.

Stitch는 적응 행동을 기술하기 위해 사용되는 언어로서, 적응 전략(Strategy), 적응 전술(Tactic), 오퍼레이터(Operator)로 구성되어있다. ACME에 의해 표현된 타깃 시스템의 속성값을 참조하며, 위반되었을 경우 수행할 적응 전략을 표현한다. 즉 Stitch는 시스템 제약 사항의 위반을 판단하는 조건(Condition), 위반되었을 경우 수행할 행동(Action), 그리고 행동을 수행한 후의 기대 효과(Effect)를 기술한다. 이와 같이 Stitch는 자가 적응을 수행하기 위한 단계를 개념 요소 단위로 분류하여 적용하므로 적응 전략과 시스템 구성 요소들 간의 관계를 명확하고 직관적으로 표현할 수 있다.

이 논문에서는 시스템의 구조를 기술하는 언어인 ACME와 적응 전략을 기술하는 언어인 Stitch를 개념 단위로 분류하기 위해 온톨로지로 정의한다. 정의된 온톨로지는 적응 전략 및 전술의 개념 단위로 재사용된다. 적응 전략에 대한 온톨로지는 3.1절에서 자세히 기술한다.

3. 적응 전략 추출 시스템

이 절에서는 자가 적응의 요구사항을 나타내는 적응 전략 온톨로지, 그리고 타깃 시스템의 구조적 정보를 나타내는 타깃 시스템 온톨로지를 정의한다. 정의된 두 온톨로지를 기반으로 타깃 시스템에서 재사용 가능한 적응 전략을 추출하는 프로세스를 기술한다.

제안 시스템은 Fig. 1과 같이 지식베이스(Knowledge Base), 타깃 시스템(Target system), 적응 전략 추출기(Adaptation Strategy Extractor)로 구성된다.

타깃 시스템의 타깃 시스템 온톨로지(Target System Ontology)는 개발하고자 하는 시스템의 구조적인 정보를 나타낸다. 지식베이스는 기존의 자가 적응 시스템들이 지니는 적응 전략을 지식화하여 관리한다. 지식베이스에서 관리되는 정보들은 적응 전략 온톨로지(Adaptation Strategy Ontology)로 구성된다. 타깃 시스템 온톨로지와 지식베이스의 적응 전략 온톨로지는 적응 전략 추출기를 거치게 된다. 적응 전략 추출기는 두 온톨로지의 특정 요소 간의 문자열 비교를 통해, 재사용 가능한 전략들을 추출하여, 재사용 가능 전략 리스트(Reusable Strategy List)로 제공한다.

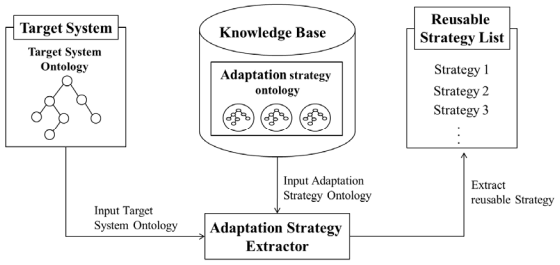

Fig. 1. An adaptation strategy extraction system architecture

3.1 지식베이스

이 절에서는 제안 시스템에서 정의한 지식베이스에 대해 설명한다. 지식베이스는 적응 전략들을 저장 및 관리하기 위하여 이용된다. 기존의 자가 적응 시스템에서 사용되고 있는 적응 전략 및 전략을 유발시키는 상태 등을 개념단위로 분류하여 온톨로지 형태로 정의하여 관리한다.

지식베이스의 적응 전략 온톨로지는 Rainbow의 적응 행동 기술 언어인 Stitch에 기반한다. Stitch는 적응 전략, 전술, 오퍼레이터, 적응 전략 수행조건 등 적응 행동에 필요한 모든 요소들 간의 관계가 명확하고 직관적으로 구성되어있으며, 이러한 요소들에 대한 개념은 다양한 자가 적응 시스템에도 공통적으로 적용할 수 있도록 설계되어있다. 따라서 Stitch는 재사용을 위해 적응 전략을 개념 요소 단위로 분류하기에 적합하다.

Fig. 2는 Stitch에 기반한 적응 전략 온톨로지를 나타낸다. 유틸리티(Utility)는 감시된 시스템의 현재 상태로서, 예를 들면 평균 응답시간(uR, Average Response Time), 평균 콘텐트의 질(uF, Average Fidelity), 평균 서버 비용(uC, Average Server Cost) 등이 해당된다. 유틸리티는 적응 전략 적용 가능성을 판단하는 척도로 각 유틸리티에 대한 직접적인 계산은 시스템의 프로브(Probe)에 의해 수행된다. 오퍼레이터(Operator)는 시스템의 상태를 변화시키는 명령으로서, 메소드 또는 함수의 집합을 의미한다. 단일 오퍼레이터 그 자체로 기능을 수행할 수 있으며 'triggerOp' 속성으로 연결된 또 다른 오퍼레이터를 호출하여 기능을 수행할 수도 있다.

전술(Tactic)은 시스템에 문제가 발생했을 때 해당 문제를 해결하기 위한 방법을 나타낸다. 전술을 유발시키는 조건(TCondition)은 유틸리티에 대한 임계값으로 표현된다. 임계값은 사용자가 현재 시스템에 맞게 직접 설정하게 되며 유틸리티가 설정된 임계값에 위반될 경우 해당 전술(Tactic)이 수행되며, 오퍼레이터를 통해 시스템을 변화시킴으로써 문제를 해결한다.

전략(Strategy)은 시스템의 문제를 해결하기 위한 방법들의 집합으로서, 다수의 전술로 구성된다. 전략을 유발시키는 조건(SCondition)은 해당 전략에서 호출되는 전술을 유발시키는 조건들에 대한 임계값으로 표현된다. 정의된 조건들이 임계값을 위반했을 경우 해당 전략이 수행되며, 다음으로 전술을 호출하는 조건(TTCondition)을 확인한다. 만약 첫 번째 전술을 수행한 후에도 문제가 해결되지 않을 경우 다음 조건을 만족하는 전술을 호출한다.

3.2 타깃 시스템

이 절에서는 제안 시스템에서 정의한 타깃 시스템에 대해 설명한다. 타깃 시스템은 자가 적응 시스템에서 스스로 적응하기 위한 대상을 의미하며 적응 전략이 실제로 수행하여 변화된 환경에 적응하기 위한 목표이다. 이 절에서는 타깃 시스템을 개념화하여 타깃 시스템 온톨로지를 정의하였다. 타깃 시스템 온톨로지는 시스템의 구조적인 정보를 나타내는 온톨로지로서, 아키텍처 기술 언어인 ACME에 기반한다. ACME는 시스템의 구성 요소, 속성, 오퍼레이터 등 시스템을 나타내는 요소를 명확하게 구분하여 표현하기 때문에, 타깃 시스템을 개념 요소 단위로 분류하기에 적합하다.

Fig. 3은 타깃 시스템의 구조적인 정보를 나타내는 온톨로지이다. 타깃 시스템은 적응 전략 온톨로지와 동일하게 시스템을 변화시키는 메소드인 오퍼레이터와 시스템이 감지하는 상태값인 유틸리티를 가진다. 오퍼레이터는 단일로 수행되어 작용할 수 있으며, 'triggerOp' 속성으로 연결된 또 다른 오퍼레이터를 호출하여 기능을 수행할 수 있다. 같은 기능을 수행하는 오퍼레이터는 개발자에 따라 각기 다른 이름으로 정의되어 모호성을 지닐 수 있다. 시스템의 상태값인 유틸리티 또한 다른 이름으로 정의될 수 있다. 이러한 점을 해결하기 위해 해당 오퍼레이터와 유틸리티를 설명하는 메타데이터를 'comment' 속성으로 표현한다. 컴포넌트(Component)는 해당 시스템이 작업을 수행하기 위해 구성하는 요소로서, 데이터베이스, 서버, 클라이언트 등이 해당된다. 속성값(Property)은 특정 타깃 시스템과 타깃 시스템을 구성하고 있는 컴포넌트 및 오퍼레이터에 사전에 설정된 값을 의미한다. 또한 컴포넌트들은 연결기(Connector)에 연결되어 작업을 수행한다.

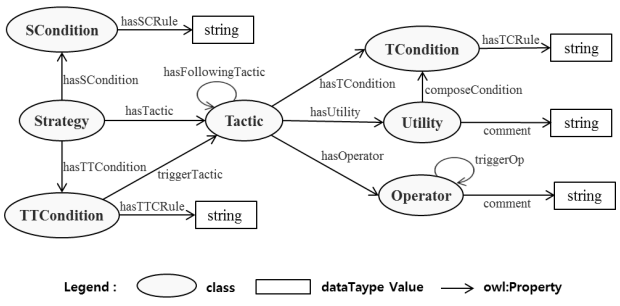

Fig. 2. Adaptation Strategy Ontology

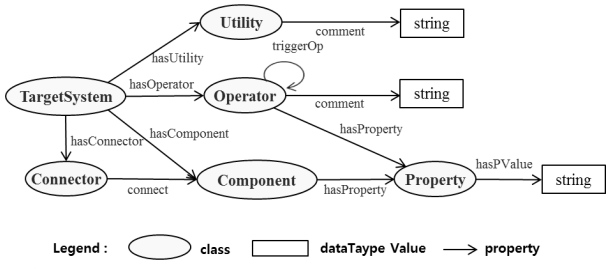

Fig. 3. Target System Ontology

3.3 적응 전략 추출기

이 절에서는 지식베이스로부터 적응 전략을 추출하기 위한 적응 전략 추출기에 대해 설명한다. 적응 전략 추출기는 적응 전략 온톨로지와 타깃 시스템 온톨로지의 비교를 통해 타깃 시스템에서 재사용 가능한 전략을 추출하는 모듈로서, Fig. 4와 같이 6단계로 나누어 적응 전략을 추출한다.

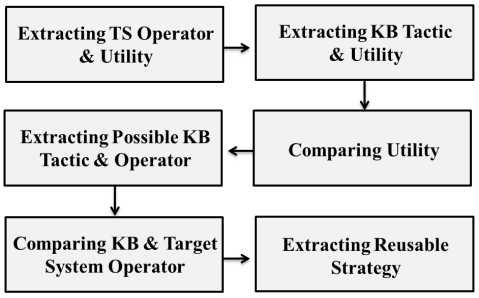

Fig. 4. Process of Extracting Adaptation Strategy

적응 행동의 수행 여부는 시스템의 상태를 나타내는 유틸리티에 의해 결정되고, 전술에서 오퍼레이터를 호출하여 최종적으로 시스템을 변화시킴으로써 문제를 해결한다. 즉, 유틸리티와 오퍼레이터 요소에 의해 전술이 만들어지며, 다수의 전술에 의해 전략이 구성된다. 따라서 타깃 시스템이 지니고 있는 유틸리티, 오퍼레이터를 기존에 관리되고 있는 지식베이스의 적응 전략 온톨로지의 유틸리티, 오퍼레이터와 비교를 하여 재사용 가능한 전술 및 전략을 추출한다.

▪ **타깃 시스템 유틸리티 및 오퍼레이터 추출 :** 유틸리티와 오퍼레이터는 전술을 구성하는 최소 단위로서, 지식베이스의 적응 전략 온톨로지와의 비교를 위하여 타깃 시스템의 유틸리티 및 오퍼레이터를 추출한다. 이를 위하여 새로 개발하고자 하는 시스템의 정보를 타깃 시스템 온톨로지로 정의하고 이를 입력받는다. 그 결과 SPARQL 쿼리를 통하여 타깃 시스템이 지니는 유틸리티 및 오퍼레이터를 추출하고 배열로써 저장한다.

▪ **지식베이스 전술 및 유틸리티 추출 :** 타깃 시스템에서 추출된 유틸리티와 유틸리티 비교를 통해 재사용 가능성이 있는 전술을 추출하기 위해 지식베이스의 전술, 그리고 해당 전술이 지니고 있는 유틸리티를 추출한다. 이를 위해 지식베이스에 정의되어 관리되고 있는 적응 전략 온톨로지를 입력받는다. 적응 전략 온톨로지의 'Strategy' 클래스가 지니는 전술을 추출하고 해당 전술의 유틸리티를 추출하여 새로운 배열로 저장한다.

▪ **유틸리티 비교 :** 타깃 시스템과 지식베이스의 유틸리티를 비교하여 재사용 가능성이 있는 전술을 추출한다. 이를 위해 이전 단계에서 추출된 타깃 시스템의 유틸리티 배열, 지식베이스의 유틸리티 배열을 입력받는다. 타깃 시스템의 유틸리티 항목과 지식베이스의 유틸리티 항목 이름에 대한 문자열, 그리고 해당 유틸리티에 'comment' 속성으로 연결된 유틸리티에 대한 설명에 대한 문자열을 비교한다. 문자열 비교를 위해 Levenshtein[16], Second String[20] 기법을 사용하며, 문자열 비교 방식은 3.3.2절에서 자세히 기술한다. 문자열 비교를 통해 유사한 것으로 판정된 타깃 시스템의 유틸리티 항목이 지식베이스의 유틸리티 항목을 포함할 경우 해당 전술을 재사용 가능성이 있는 것으로 판단한다. 그 이유는 타깃 시스템이 전술의 수행 여부를 판단하는 유틸리티를 가지고 있는 경우에만 해당 전술을 사용할 수 있기 때문이다. 최종적으로 재사용 가능성이 있는 전술을 추출하여 배열로 저장한다.

▪ **지식베이스의 오퍼레이터 추출 :** 지식베이스에서 재사용 가능성이 있는 것으로 추출된 전술이 지니고 있는 오퍼레이터를 추출한다. 유틸리티 비교를 통해 추출된 전술 중 실제로 재사용이 가능한 것을 오퍼레이터 비교를 통해 추출하기 위해 수행되는 단계이다. 이를 위해 이전 단계에서 추출된 재사용 가능성이 있는 전술의 배열을 입력받는다. 재사용 가능성이 있는 전술이 지니고 있는 오퍼레이터를 추출하여 배열로 저장한다.

▪ **오퍼레이터 비교 :** 타깃 시스템과 지식베이스의 오퍼레이터를 비교하여 재사용 가능한 전술을 추출한다. 이를 위해 1단계에서 추출된 타깃 시스템의 오퍼레이터 배열과, 4단계에서 추출된 지식베이스의 오퍼레이터 배열을 입력받는다. 타깃 시스템의 오퍼레이터 항목과 지식베이스의 오퍼레이터 항목 이름에 대한 문자열, 'comment' 속성으로 연결된 해당 오퍼레이터 설명에 대한 문자열, 해당 오퍼레이터가 호출하는 또 다른 오퍼레이터 항목에 대한 문자열을 비교한다. 문자열 비교를 위해 Levenshtein, SecondString 기법을 사용하며, 문자열 비교 방식은 3.3.2절에서 자세히 기술한다. 문자열 비교를 통해 유사한 것으로 판정된 타깃 시스템의 오퍼레이터 항목이 지식베이스의 오퍼레이터 항목을 포함할 경우 해당 전술을 재사용할 수 있는 것으로 판단한다. 그 결과, 재사용 가능한 전술을 추출하여 배열로 저장한다.

▪ **재사용 가능 전략 추출 :** 지식베이스에서 재사용 가능한 전술을 사용하는 전략을 추출한다. 오퍼레이터 비교 과정을 통해 추출된 재사용 가능한 전술 배열을 입력받는다. 적응 전략은 문제에 대한 해결 방법인 다수의 전술로 구성되어있기 때문에 그중 하나라도 재사용이 가능한 경우 주어진 문제는 해결 가능한 것으로 판단하며 해당 전략은 재사용 가능한 것으로 간주한다. 최종적으로 재사용 가능한 전략이 출력된다.

적응 전략 추출기를 통해 최종적으로 재사용 가능한 전략이 추출되며, 타깃 시스템은 해당 전략을 재사용하여 적응을 위한 요구사항 개발 시 소요되는 시간을 단축시킬 수 있으며, 재사용된 적응 전략은 실제 타깃 시스템의 문제 해결에 도움을 줄 수 있다.

1) 적응 전략 및 전술 추출 알고리즘

Table 1은 3.2절의 오퍼레이터 비교, 재사용 가능 전략 추출 단계에 해당되는 알고리즘으로서, 재사용 가능한 전술 및 전략을 추출한다. 유틸리티 비교를 통해 저장된 재사용 가능성이 있는 전술 배열에서 각 전술이 지니고 있는 오퍼레이터를 SPARQL 질의를 통해 추출한다. 다음으로 지식베

Table 1. Algorithm for Extracting Reusable Tactic and Strategy

[Algorithm] Extracting reusable tactic and strategy

```
Input : TSO[] // Target system operator list
        availableTactic[] //Tactic list which has probability
                          //to be reused.
Output : possibleStrategyList[] //Strategy list which can
                                //to be used
Initialize : i=0, j=0, k=0, u=0
TSO[] ← Query(queryString_op) //get operator list of
                              //target system
avaiableTactic[] ← extractAvailableTactic(TSO[])
//save available tactic of the target system to the array
FOR ∀a∈sCount DO
   //while I is below the number of strategy
   kbt_queryResults ← Query(kbt_queryStringList[i])
   //get tactic list of the KB
   IF kbt_queryResults.hasNext THEN
      tacticList[k] ← kbt_queryResults.nextSolution()
      //save the tactic to the string array[KB]
      k ← k+1
   ENDIF
   FOR ∀i∈q DO
      FOR ∀j∈k DO
         IF availableTactic[i] = tacticList[j] THEN
         //compare tactic list
           count ← count+1
         ENDIF
         INCREMENT k
      ENDFOR
      INCREMENT q
   ENDFOR
   IF count!=0 THEN
      possibleStrategyList[u] ← KBSL[i]
      //save possible strategy to the array
      u ← u+1
   ENDIF
   INCREMENT i
ENDFOR
RETURN possibleStrategyList[]
```

이스와 타깃 시스템 오퍼레이터 간의 문자열 비교 과정을 거치게 되며 타깃 시스템의 오퍼레이터 항목이 지식베이스의 오퍼레이터 항목을 포함할 경우 해당 전술은 재사용이 가능한 것으로 판단하여 추출한다. 전략은 문제 해결 방법인 다수의 전술로 구성되어있기 때문에 추출된 전술 중 하나라도 재사용이 가능한 경우 해당 전략 또한 재사용이 가능한 것으로 판단하여 추출한다.

2) 온톨로지 유사도 비교

이 절에서는 재사용 가능한 전술 및 전략을 추출하기 위해 수행되는 3.2절의 유틸리티 비교, 오퍼레이터 비교 단계에서 사용하는 문자열 비교 기법에 대해 설명한다. 유틸리티 비교를 위해 유틸리티 이름, 해당 유틸리티에 대한 설명을 고려하며, 오퍼레이터 비교를 위해 오퍼레이터의 이름, 호출된 오퍼레이터 리스트, 해당 오퍼레이터에 대한 설명을 고려한다.

유틸리티 및 오퍼레이터 이름에 대한 유사도를 비교하기 위해 Levenshtein 거리[16]에 기반한 유사도 측정 기법[14]을 사용한다. 유틸리티 및 오퍼레이터의 이름은 공백이 없는 한 단어로 정의되며, 따라서 이 기법은 단어의 유사도를 비교하는 데 적합하다. Equation (1)은 두 유틸리티 또는 두 오퍼레이터 i, j 의 이름 l_i, l_j 이름의 유사도를 비교하는 수식이다.

$$\mathrm{SM}(l_i, l_j) = \max\left(0, \frac{\min(|l_i|,|l_j|) - \mathrm{ed}(l_i,l_j)}{\min(|l_i|,|l_j|)}\right) \in [0,1] \quad (1)$$

비교하는 두 오퍼레이터가 호출하는 오퍼레이터의 항목을 비교하기 위해 Levenshtein에 기반한 유사도 기법을 사용한다. 두 오퍼레이터가 각각 m, n개의 오퍼레이터를 호출하며 호출된 오퍼레이터의 집합을 L^m, L^n로 가정하면 Equation (2)에 의해 호출된 오퍼레이터들의 평균 유사도가 계산된다.

$$\overline{\mathrm{SM}}(L^m, L^n) = \frac{1}{|L^m|}\sum_{l_i \in L^m}^{k} \max_{l_j \in L^n} \mathrm{SM}(l_i, l_j) \quad (2)$$

유틸리티 및 오퍼레이터에 'comment' 속성으로 연결되어 해당 유틸리티 및 오퍼레이터를 설명하는 문자열을 비교하기 위해 Second String 유사도 측정 기법[20]을 사용한다. 유틸리티 및 오퍼레이터에 대한 설명은 공백을 포함한 문장 단위의 문자열로 이루어져 있기 때문에 문장의 유사도를 비교하는 Second String 기법을 사용하였다. 두 유틸리티 또는 두 오퍼레이터 i, j의 'comment'에 대한 문자열이 $i_s = a_1 \ldots a_k$, $j_s = b_1 \ldots b_l$이면 유사도는 Equation (3)에 의해 계산된다.

$$\mathrm{sim}(i_s, j_s) = \frac{1}{k}\sum_{i=1}^{k}\sum_{j=1}^{l} \max(\mathrm{sim}'(a_i, b_j)) \quad (3)$$

최종적으로 유틸리티 i, j의 유사도는 다음 Equation (4)에 의해 계산되며, 유틸리티의 유사도는 유틸리티 이름의 유사도, 유틸리티에 대한 설명의 유사도 각각에 부여된 가중치에 따라 값이 유동적으로 바뀐다.

$$\mathrm{Similarity}(i,j) = \alpha * \mathrm{SM}(l_i, l_j) + \beta * \mathrm{sim}(i_s, j_s) \quad \alpha + \beta = 1 \quad (4)$$

최종적으로 유틸리티 i, j의 유사도는 Equation (5)에 의해 계산되며, 오퍼레이터의 유사도는 오퍼레이터 이름, 오퍼레이터에 대한 설명, 호출된 오퍼레이터 리스트 이름의 유사도 각각에 부여된 가중치에 따라 값이 유동적으로 바뀐다.

$$\mathrm{Similarity}(i,j) = \alpha * \mathrm{SM}(l_i, l_j) + \beta * \overline{\mathrm{SM}}(L^m, L^n) + \gamma * \mathrm{sim}(i_s, j_s) \quad \alpha + \beta + \gamma = 1 \quad (5)$$

4. 구현 및 평가

이 절에서는 제안 시스템을 구현하고 실험을 통해 제안 시스템의 재사용률을 평가한다. 또한 추출된 적응 전략을 실제 자가 적응 시스템에 적용시킴으로써 제안 시스템을 검증한다.

4.1 제안 시스템 구현

이 절에서는 Fig. 1의 제안 시스템 구조와 같이 타깃 시스템 온톨로지를 입력하였을 때 해당 타깃 시스템에 적용 가능한 전략을 추출하는 시스템을 구현한다. 온톨로지 구축을 위해 온톨로지 생성 및 편집 툴인 Protégé[21]를 사용하며, 온톨로지 추출을 위해 Jena 프레임워크[22]를 사용한다.

실험에서 지식베이스는 기존의 자가 적응 뉴스 시스템인 Znn.com의 적응 전략을 온톨로지화 하여 이용한다. Znn.com은 슬래시닷 효과(Slashdot effect)로 인해 시스템에 문제가 발생한 경우, 가용한 서버의 수를 늘리거나, 웹 서비스에서 제공하는 콘텐트의 질을 낮추는 등의 적응 전략을 사용하여 시스템 스스로 적응함으로써 문제를 해결한다[23]. Tdisk.com은 새롭게 구축하고자 하는 타깃 시스템으로 Znn.com과 유사한 도메인의 가상 뉴스 시스템이며, Tdisk.com의 구조적인 정보는 타깃 시스템 온톨로지로 정의된다. Znn.com과 마찬가지로 Tdisk.com 역시 슬래시닷 효과로 인해 시스템이 다운되는 문제가 발생하고 사용자가 데이터를 제공받지 못하는 문제가 발생하며, 제안 시스템을 통해 지식베이스로부터 재사용 가능한 적응 전략을 추출하여 Tdisk.com을 구축하고자 한다. 하지만 Tdisk.com은 Znn.com과 다르게 평균 콘텐트의 질을 모니터링 할 수 있는 'uF' 유틸리티를 지니고 있지 않으며, 제공하는 콘텐트의 질을 변경하는 'setFidelity', 접속자의 권한을 설정하는 'setPermission', 그리고 위협이 되는 접속자에게 경고하는 'warnUser' 오퍼레이터는 지니고 있지 않다. 이 실험은 이러한 데이터들을 이용하여 Tdisk.com에서 재사용될 수 있는 적응 전략들이 제대로 추출되는지를 검증하고자 한다.

Fig. 5는 3.1절, 3.2절에 정의된 적응 전략 온톨로지, 타깃 시스템 온톨로지에 따라 표현된 인스턴스들을 나타낸다. Fig. 5의 상단 부분은 기존의 자가 적응 시스템인 Znn.com의 적응 전략 온톨로지의 인스턴스를 나타내며, 하단 부분의 새로운 타깃 시스템 Tdisk.com의 타깃 시스템 온톨로지의 인스턴스를 나타낸다. 또한 Table 2는 이러한 인스턴스들의 의미를 기술한다.

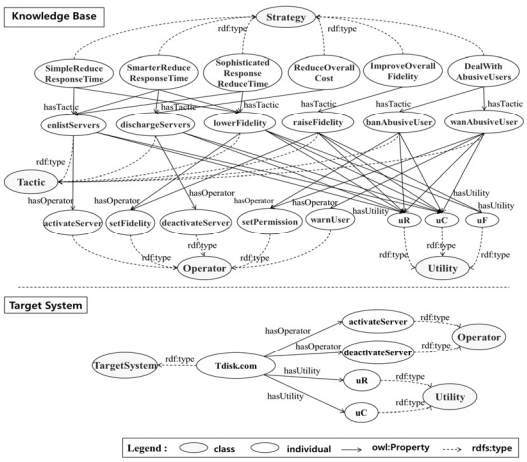

Fig. 5. Individuals of Adaptation Strategy Ontology from Knowledge Base and Target System Ontology

Table 2. Operator, Tactic, Strategy, Utility of Znn.com

Type Name	Functional Description
Operator	
avtivateServer	Activate Server instance
deactivateServer	Deactivate Server instance
setFidelity	Set the server content fidelity to the level identified by the input parameter
setPermission	Set client permission to the level identified by the input parameter
warnUser	Warn the client about violation
Tactic	
enlistServers	Enlist n free servers into server pool
dischargeServers	Deactivate n servers from server pool into free pool
lowerFidelity	Lowers fidelity by integral steps for percent of requests
raiseFidelity	Raise fidelity by integral steps for percent of requests
banAbusiveUser	Ban abusive client to access the web-site
warnAbusiveUser	Set abusive client permission to some degree, and warn the client about violation
Strategy	
SimpleReduce ResponseTime	While it encounters high experienced response time, this simple strategy first enlists one server, then lowers fidelity one step, then quits
SmarterReduce ResponseTime	This strategy looks for a percentage of clients with anomalous experienced response time, in which case it enlists a few servers in sequence, then lowers fidelity a few step, then starts delaying clients
SophisticatedReduce ResponseTime	This experimental strategy has the sophistication of reducing fidelity for a percentage of requests depending on percentage of unhappy clients
ReduceOverallCost	Triggered by total server costs above threshold, it reduce number of active servers
ImproveOverall Fidelity	Triggered by overall fidelity below threshold, it raises server fidelity
DealWithAbusive Users	This strategy looks for access rate of each clients, if the bandwidth exceed some degree, it ban the abusive client, then lowers permission and warn the client about violation
Utility	
uR	Average Response Time, client experienced response time(ms), float arch property
uF	Average Fidelity, server content fidelity level, int arch property
uC	Average Server Cost, server cost in unit/hr, average of float arch property

Fig. 6은 타깃 시스템의 입력화면을 나타낸다. 제안 시스템의 적응 전략 추출기는 입력값으로 받은 타깃 시스템 온톨로지를 지식베이스의 적응 전략 온톨로지와 비교하여 재사용 가능한 전술 및 전략을 추출한다. Fig. 7은 제안 시스템을 통해 추출된 적응 전술 및 전략들의 리스트를 나타낸다. 타깃 시스템은 슬래시닷 효과로 인해 과부하가 발생하여 서버가 다운되고 원활한 서비스를 제공하지 못하는 문제가 발생할 경우 가용 가능한 서버의 수를 늘리거나 서버가 제공하는 콘텐트의 질을 낮춤으로써 문제를 해결하는 'SophisticatedReduceResponseTime', 'SmarterReduceResponseTime', 'SimpleReduceResponseTime', 그리고 문제가 해결된 경우 다시 서버의 수를 줄여 비용을 절감시키는 'ReduceOverallCost' 전략 및 'dischargeServers', 'enlistServers' 전술을 재사용할 수 있다.

4.2 비교 평가

이 절에서는 제안 시스템으로부터 추출된 적응 전략의 재사용률을 평가한다. CORE는 재사용에 관련된 연구로서, 두 온톨로지 인스턴스들 사이의 유사도를 측정하고, 측정된 유사도에 기반하여 재사용 가능한 온톨로지를 추출하여 제안 시스템과 재사용률을 평가할 비교 대상으로 선정한다.

유사도 측정 실험에서 제안 시스템의 유틸리티 유사도 측정을 위하여 3.2.2절 수식 4의 α,β를 0.6, 0.4로 설정한다. 또한 오퍼레이터의 유사도 측정을 위하여 Equation (5)의 α,β,γ를 각각 0.5, 0.3, 0.2로 설정하였다. Table 3은 CORE와 제안 시스템의 유사도 측정 기법을 통해 도출된 지식베이스의 각 적응 전략의 유사도를 나타낸다.

Table 3. Similarity of Adaptation Strategy with CORE and Proposed System

Strategy	Similarity	
	CORE	Proposed System
SimpleReduce ResponseTime	0.63365	0.88499
SmarterReduce ReponseTime	0.63365	0.88499
SophisticatedReduce ResponseTime	0.63365	0.88499
ReduceOveallCost	0.87500	0.82249
ImproveOverallFidelity	0.59038	0.00142
DealWithAbusiveUsers	0.44285	0.00133

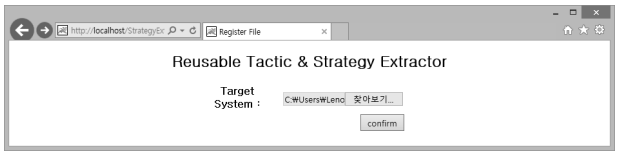

Fig. 6. Target System Ontology Input Screen

Fig. 7. List of the Reusable Tactic and Strategy

CORE의 유사도의 평균은 0.63486, 표준편차는 0.12677이며, 제안 시스템의 유사도의 평균은 0.580035, 표준편차는 0.40977이다. 표준편차가 낮은 CORE는 각 적응 전략의 유사도가 평균값에 근접해있으며, 따라서 추출을 위한 유사도의 임계치를 설정하기가 힘들다. 반면에 제안 시스템의 유사도는 CORE보다 높은 표준편차를 지니므로, 추출을 위한 임계치를 CORE에 비해 명확히 설정할 수 있다.

결과값의 정확성을 높이기 위해 두 시스템 모두 유사도의 임계치를 0.8로 설정하여 추출한다. CORE는 지식베이스를 구성하는 총 6개의 적응 전략 중 1개의 적응 전략 'ReduceOverallCost'를 추출하며, 제안 시스템은 총 6개의 적응 전략 중 4개의 적응 전략 'SimpleReduceResponse time', 'SmarterReduceResponseTime', 'SophisticatedReduceResponseTime', 'ReduceOverallCost'를 추출한다. 즉, 유사도에 기반해 온톨로지를 추출할 경우 CORE의 경우 지식베이스의 적응 전략 온톨로지의 16.66%를 재사용할 수 있으며, 제안 시스템의 경우 66.66%를 재사용할 수 있다.

CORE는 유사도 측정을 위해 모든 인스턴스들을 비교하는 기법으로서, 재사용에 불필요한 요소들까지 비교 대상에 속하게 된다. 또한 CORE는 클래스의 구조, 메타데이터 등은 고려하지 않는다. 이와 같은 요인들로 인해 CORE는 낮은 재사용률이 측정되었다. 반면에 제안 시스템은 오퍼레이터, 유틸리티 비교에 기반하여 재사용을 위해 필요한 요소만을 비교하고, 클래스의 구조 및 'comment' 속성으로 표현된 메타데이터 역시 고려하여 문자열 비교를 수행하기 때문에 CORE에 비해 높은 재사용률을 확인할 수 있었다.

4.3 추출 결과 평가

이 절에서는 제안 시스템에서 추출된 결과를 실제 자가적응 시스템에 적용하여 설계하였을 때 정확히 동작하는지를 평가한다. 이를 위해 제안 시스템과 CORE를 통해 4.2절에서 유사도 비교를 통하여 추출된 적응 전략을 타깃 시스템인 Tdisk.com에 각각 적용시켜 서버 부하 테스트를 진행하였다. 실제 다수의 사람이 동시에 접속하여 테스트하기에는 한계가 있기 때문에 서버 부하테스트 시뮬레이터인 JMeter[24]를 사용하였다.

Fig. 8은 제안 시스템이 추출한 적응 전략이 적용된 Tdisk.com과 CORE가 추출한 적응 전략이 적용된 Tdisk.com을

JMeter를 이용하여 시뮬레이션 한 테스트 결과를 보인다. 이 실험에서 동시 접속자 수(# samples)는 2000명으로 설정하며 이는 실제 Tdisk.com에 슬래시닷 효과를 유도한다. CORE가 추출한 적응 전략 'ReduceOverallCost'만이 적용된 타깃 시스템의 경우 슬래시닷 효과에 따른 문제를 해결하지 못하고 응답시간이 증가함을 확인할 수 있다.

CORE는 78,722ms의 최대 응답시간(Max)을 기록하며, 17,869ms의 평균 응답시간(Average), 그리고 59.40%의 오류율(Error %)을 보인다.

반면 제안 시스템의 경우 슬래시닷 효과로 인한 문제 발생 시, 가용한 서버 수를 증가시키는 'SimpleReduce ResponseTime', 'SmarterReduceResponseTime', 'Sophisti catedReduceResponseTime'의 세 가지 적응 전략을 이용하면 27,339ms의 최대 응답시간, 평균적으로 8,180ms의 응답시간, 10.85%의 오류율을 보인다. 그 결과, 제안 시스템에서 추출한 적응 전략이 적용된 타깃 시스템의 경우 CORE에 비해 46% 빠른 응답시간과 49% 적은 오류율을 확인할 수 있다.

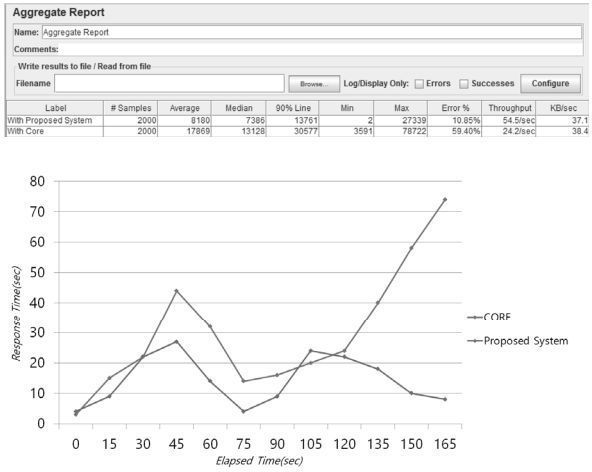

Fig. 8. A Simulation Result of Server Load Test using JMeter

5. 결 론

최근 자가 적응 시스템의 개발을 지원하기 위해 다양한 연구들이 진행되었다. 하지만 기존의 연구는 지식을 공유하고 재사용하는 기능을 제공하지 않는다. 따라서 적응을 위한 요구사항을 자가 적응 시스템 개발 단계에서 매번 다시 정의해야 하며, 그로 인한 비용이 발생하는 문제가 있다.

이 논문은 이러한 문제를 해결하기 위해 자가 적응 시스템을 개발할 때 기존에 정의된 적응 전략 중 재사용 가능한 것을 추출하는 시스템을 제안하였다. 이를 위해 적응 전략에 연관된 지식을 관리할 수 있는 적응 전략 온톨로지와 타깃 시스템 온톨로지를 정의하였다. 정의된 온톨로지를 기반으로 지식이 관리될 때 유틸리티, 오퍼레이터 비교를 통해 재사용 가능한 전략을 추출하는 단계를 기술하였다. 또한 비교 평가를 통해 제안 시스템이 기존 연구에 비해 더 높은 재사용률을 지님을 확인하였으며, 추출된 적응 전략을 실제로 재사용하여 적용시킴으로써 문제를 해결할 수 있음을 보였다.

자가 적응 시스템은 모든 산업에서 이용될 수 있다. 특히 자가 적응 시스템 개발 시 요구사항 정의를 위해 소요되는 시간은 전체 시스템 구축에서 큰 비중을 차지한다. 제안 시스템은 4절에서 언급된 Znn.com과 Tdisk.com 예제와 같이 동일한 영역에서 활용된다면 높은 재사용률로 자가 적응 시스템 개발의 비용을 줄일 수 있다. 또한 동일한 영역이 아닌 다른 영역에서 사용될 경우 지식베이스로부터 추출되는 적응 전략은 동일한 영역에 비해서는 낮은 재사용률을 지니지만, 한 분야의 도메인 전문가가 생각하지 못한 문제를 다른 영역의 적응 전략을 적용함으로써 해결할 수 있다는 점을 고려하였을 때 큰 시너지 효과를 지닌다.

현재 구축되어있는 자가 적응 시스템의 수는 제한적이기 때문에 제안 시스템에서 사용할 수 있는 적응 전략은 한정적이다. 하지만 향후 제안 시스템의 지식베이스에 자가 적응 시스템의 적응 전략에 대한 정보가 축적됨에 따라 타깃 시스템은 다양한 적응 전략을 추출하여 적용시킬 수 있을 것이다. 또한, 적응 전략 추출 프로세스를 통해 추출되는 전략의 재사용률을 높이기 위해 타깃 시스템의 도메인 간의 유사도를 비교하는 연구가 요구된다.

References

[1] M. Salehie, L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems(TAAS)*, Vol.4, No.2, 2009.

[2] B.H. Cheng, R. De Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, and B. Becker et al, "Software engineering for self-adaptive systems: A research roadmap," In Software engineering for self-adaptive systems, pp.1-26, 2009.

[3] S. R. White, J. E. Hanson, I. Whalley, D. M. Chess, and J. O. Kephart, "An architectural approach to autonomic computing," In Autonomic Computing, International Conference on, *IEEE Computer Society*, pp.2-9, 2004.

[4] M. Parashar, S. Hariri, "Autonomic computing: An overview," In Unconventional Programming Paradigms, Springer Berlin Heidelberg, pp.257-269. 2005.

[5] M. C. Huebscher, J. A. McCann, "A server of autonomic computing-degrees, models, and applications," *ACM Computing Surverys(CSUR)*, Vol.40, No.3, 2007.

[6] V. E. S. Souza, A. Lapouchanian, K. Angelopulos, and J. Mylopolos, "Requirements-driven software evolution," *Computer Science -Research and Development*, Vol.28, No.4, pp.311-329, 2013.

[7] D. Garlan, S.W. Cheng, A.C Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *Computer*, Vol.37, No.10, pp.46-54, 2004.

[8] DiVA Project Consortium, "A Model-based Approach for Construction and Run-time Management of Adaptive Systems: DiVA practices and Lessons Learned," DiVA White Paper, 2011.

[9] S. Lombardo, “Mobility and Adaptation enabling Middleware: Final Report,” MADAM final report, 2007.
[10] B. W. Boehm, “A spiral model of software development and enhancement,” *Computer*, Vol.21, No.5, pp.61-72, 1988.
[11] S. W. Cheng, D. Garlan, “Stitch: A language for architecture-based self-adaptation,” *Journal of Systems and Software 85*, No.12, pp.2860-2875, 2012.
[12] V. E. S. Souza, J. Mylopoulos, “Requirements-based software system adaptation,” PhD Thesis, University of Trento, Italy, 2012.
[13] D. Garlan, B. Schmerl, and S. W. Cheng, “Software architecture-based self-adaptation,” *In Autonomic computing and networking*, Springer, pp.31-55. 2009.
[14] A. Mukhija, M. Glinz, “A framework for dynamically adaptive applications in a self-organized mobilie network environment,” In Distributed Computing Systems Workshops, 2004, Proceedings. 24th International Conference on, pp. 368-374, 2004.
[15] A. Maedche, S. Staab, “Measuring Similarity between Ontologies,” In Knowledge engineering and knowledge management: Ontologies and the semantic web, Springer Berlin Heidelberg, pp.251-263, 2002.
[16] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions and reversals,” *In Soviet physics doklady*, Vol.10. pp.707, 1966.
[17] M. Fernandez, I. Cantador, and P. Castells, “CORE: A tool for collaborative ontology reuse and evaluation,” The 4th Int. Workshop on Evaluation of Ontologies for the Web, at the 15th Int. World Wide Web Conference, pp.23-26, 2006.
[18] M. Gaeta, F. Oriuoli, S. Paolozzi, and S. Salerno, “Ontology extraction for knowledge reuse: The e-learning perspective,” Systems, Man and Cybermetics, Part A: System and Humans, *IEEE Transactions on*, Vol. 41, No. 4, pp.798-809, 2011.
[19] D. Garlan, M. Robert, and W. David, “Acme: an architecture description interchange language,” In CASCON First Decade High Impact Paper, IBM Corp, pp.159-173, 2010.
[20] W. Cohen, P. Ravikumar, and S. Fienberg, “A comparison of string metrics for matching names and records,” *In KDD Workshop on Data Cleaning and Object Consolidation*, Vol.3. pp.73-78, 2003.
[21] J. H. Gennari, M. A. Musen, R. W. Fergerson, W. E Grosso, M. Crubezy, H. Eriksson, and S. W. Tu, “The evolution of Protégé: an environment for knowledge-based systems development,” I*nternational Journal of Human computer studies*, Vol.58, No.1, pp.89-123, 2003.
[22] A. Jena[Internet], https://jena.apache.org/
[23] E. Yuan, S. Malek, B. Schmerl, D. Garlan, and J. Gennari, “Architecture-based self-protecting software systems,” In proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures, pp.33-42, 2013.
[24] JMeter[Internet], http://jmeter.apache.org/

남 정 식

e-mail : melonam@korea.ac.kr
2013년 고려대학교 컴퓨터정보학과(학사)
현 재 고려대학교 컴퓨터·전파통신공학과 석사
관심분야 : 자가 적응 시스템, 지식 재사용, 온톨로지

이 석 훈

e-mail : leha82@korea.ac.kr
2009년 고려대학교 전자 및 정보공학부(학사)
2011년 고려대학교 컴퓨터·전파통신공학과 (공학석사)
현 재 고려대학교 컴퓨터·전파통신공학과 박사과정
관심분야 : 자가 적응 시스템, 온톨로지, 데이터 마이닝, 메타데이터 레지스트리, 자율 컴퓨팅

백 두 권

e-mail : baikdk@korea.ac.kr
1974년 고려대학교 수학과(학사)
1977년 고려대학교 산업공학과(석사)
1983년 Wayne State Univ. 전산학과(석사)
1985년 Wayne State Univ. 전산학과(박사)
현 재 고려대학교 컴퓨터·전파통신공학과 /융합소프트웨어전문대학원 교수
1989년~1991년 고려대학교 전산과학과 학과장
1990년~1991년 미국 Arizona대학교 객원 교수
1991년~2013년 ISO/IEC JTC1/SC32 전문위원회 위원장
1993년~1999년 한국과학기술원 객원책임연구원
1993년~1999년 한국DB진흥센터 표준연구위원
1996년~1997년 고려대학교 컴퓨터과학기술연구소(초대소장)
1997년~1998년 고려대학교 정보전산원 원장
1998년~1999년 한국정보과학회 전산교육연구회 운영위원장
1999년~2001년 정보통신진흥협회 데이터기술위원회 의장
2002년~2004년 고려대학교 정보통신대학(초대학장)
2002년~2003년 한국시뮬레이션학회 회장
2003년~현 재 정보통신부 컴퓨터프로그램보호위원회 위원
2004년~2005년 한국정보처리학회 부회장
2005년~2008년 한국소프트웨어진흥원 이사
2009년~2010년 고려대학교 정보통신대학 학장
관심분야 : 메타데이터, 소프트웨어공학, 데이터공학, 컴포넌트기반 시스템, 메타데이터 레지스트리, 프로젝트 매니지먼트 등