자율 컴퓨팅을 적용한 SOA 서비스 결함 관리 기법

(A Method to Manage Faults in SOA using Autonomic Computing)

천 두 완 † 이 재 유 † 라 현 정 † 김 수 동 ††
(Du Wan Cheun) (Jae Yoo Lee) (Hyun Jung La) (Soo Dong Kim)

요 약 서비스 지향 아키텍처에서 서비스 제공자는 재사용 가능한 서비스를 개발하고 저장소에 배포하며, 서비스 사용자는 인터페이스를 통하여 블랙박스 컴포넌트 형태의 서비스를 사용한다. 저장소에 배포된 서비스는 시간이 지남에 따라 변경/진화될 가능성이 높고, 다양한 언어 또는 플랫폼을 사용하여 구현되는 이질성(Heterogeneity)을 가진다. 이런 이유로, 서비스 사용자는 서비스 내부 구조를 알기 힘들기 때문에, 서비스가 기능을 수행하는 도중 문제점이 발생하면 문제점을 식별하여 해결하는 등의 서비스 결함을 효과적으로 관리하는 것이 어렵다. 자율 컴퓨팅(Autonomic Computing, AC)은 사람의 개입을 최소화하고 시스템이 스스로의 결함을 관리하도록 설계하는 방식이다. AC는 시스템을 자율적으로 결함을 관리할 수 있는 주요 원칙들을 제안하고 있으므로, 서비스 결함 관리에 관한 기술적 이슈들은 AC의 기법들을 사용하여 해결될 수 있다. 본 논문에서는 SOA 환경에서 자율적으로 서비스의 결함을 관리하기 위한 이론적 모델인 Symptom-Cause-Actuator(SCA) 모델을 제시한다. SCA 모델은 의사가 환자를 치료하는 과정으로부터 유도된다. 먼저, 다섯 단계로 구성된 SCA 컴퓨팅 모델을 정의하고 SCA의 메타모델을 제안한다. 또한, SCA 모델의 저장소 역할을 하는 SCA 프로파일을 정의하고, SCA 프로파일에 저장되는 symptom, cause, actuator의 인스턴스와 이들 간의 의존 관계를 기계가 인식할 수 있는 형식으로 표현한다. 그리고, 서비스의 결함을 자율적으로 관리하는 컴퓨팅 모델의 다섯 단계를 수행하는데 필요한 알고리즘을 상세하게 기술한다. 마지막으로, SCA 모델의 실행 가능성을 보여주기 위하여, SCA 프로파일과 알고리즘을 구현한 프로토타입을 '비행기 예약 시스템'에 적용하는 사례 연구를 수행한다.

키워드 : 자율 컴퓨팅, 서비스 지향 아키텍처, 서비스 결함 관리

Abstract In Service-Oriented Architecture (SOA), service providers develop and deploy reusable services on the repositories, and service consumers utilize blackbox form of services through their interfaces. Services are also highly evolvable and often heterogeneous. Due to these characteristics of the service, it is hard to manage the faults if faults occur on the services. Autonomic Computing (AC) is a way of designing systems which can manage themselves without direct human intervention. Applying the key disciplines of AC to service management is appealing since key technical issues for service management can be effectively resolved by AC. In this paper, we present a theoretical model, Symptom-Cause-Actuator (SCA), to enable autonomous service fault management in SOA. We derive SCA model from our rigorous observation on how physicians treat patients. In this paper, we first define a five-phase computing model and meta-model of SCA. And, we define a schema of SCA profile, which contains instances of symptoms, causes, actuators and their dependency values in a

· 이 논문은 2006년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임(KRF-2006-005-J03803)

† 학생회원 : 숭실대학교 컴퓨터학과
dwcheun@otlab.ssu.ac.kr
jylee81@otlab.ssu.ac.kr
hjla@otlab.ssu.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학과 교수
sdkim@ssu.ac.kr

논문접수 : 2008년 6월 23일
심사완료 : 2008년 10월 20일

정보과학회논문지: 소프트웨어 및 응용 제35권 제12호(2008.12)

machine readable form. Then, we present detailed algorithms for the five phases that are used to manage faults the services. To show the applicability of our approach, we demonstrate the result of our case study for the domain of 'Flight Ticket Management Services'.

Key words : Autonomic Computing, Service-Oriented Architecture, Service Fault Management

1. 서 론

서비스 지향 아키텍처에서 서비스 제공자는 공통적인 특징들을 이용하여 서비스를 개발하고, 서비스 저장소에 배포한다. 서비스 사용자는 배포된 서비스를 실시간으로 발견하고, 그 중 필요한 서비스를 구독한다. 서비스 제공자와 사용자 사이의 낮은 결합도로 인해 SOA에서 서비스 관리는 중요하고 도전할만한 연구 목표이다[1,2]. 서비스 관리(Service Management)는 서비스 소비자가 요구하는 서비스 수준으로 서비스가 올바른 기능을 수행하기 위해 지원되는 일련의 행위들을 말하며, 여기에는 서비스 생명 주기 관리, 서비스 품질 관리, 서비스 결함 관리 등이 포함된다[3]. 이 중 서비스 결함 관리는 서비스의 문제점을 식별하여 이를 해결함으로써, 서비스가 올바르게 기능을 수행할 수 있도록 하는 일련의 행위들을 일컫는다. 저장소에 배포된 서비스들은 블랙박스처럼 인식되므로 서비스 사용자와 관리자는 서비스에 대해 제한된 가시성과 관리성을 가지게 된다. 또한, 서비스는 서비스 변경에 대한 알림없이 진화할 수 있고, 존재하는 서비스가 순간적으로 혹은 영속적으로 사라질 수도 있다. 이와 같은 SOA의 속성은 서비스가 결함없이 정확하게 수행하는지 관리하는 것을 매우 복잡하고 어렵게 만든다.

자율 컴퓨팅(AC)은 사람의 직접적인 조정없이 자율적인 방식하에 스스로가 결함을 관리하도록 시스템을 설계하는 방식이다[4]. AC는 시스템을 자율적으로 결함을 관리할 수 있는 주요 원칙들을 제안하고 있으므로, 전통적인 소프트웨어의 결함을 관리하는 기법으로 사용되어졌다. 자율적으로 시스템을 관리하는 기법은 일반 소프트웨어 뿐 아니라 서비스 결함 관리에도 적용될 수 있으며, AC 기법을 이용하여 서비스 결함 관리에 관한 기술적 이슈들을 해결하려는 시도가 행해졌다[5,6]. 이런 시도들은 서비스 결함 관리 작업의 대부분이 AC를 적용함으로써 자동화가 가능해질 수 있다는 가능성을 제시하였으나, 실제적으로 AC 기법이 어떻게 적용되어 구현될 수 있는지에 대한 가능성은 제시하지 않았다.

본 논문에서는 SOA에서 자율적인 서비스 관리를 위한 Symptom-Cause-Actuator (SCA)의 이론적 모델을 제시한다. SCA 모델은 의사가 환자를 치료하는 과정에 대한 정확한 관찰을 통하여 유도되며, MAPE(모니터(Monitor), 분석(Analyze), 계획(Plan), 실행(Execute))의 처리 방식에 적용 가능하다[4]. 그리고, 다섯 단계로 구성된 SCA 컴퓨팅 모델을 정의하고, SCA의 구조를 표현한 메타모델을 제안한다. 또한, SCA 프로파일의 스키마를 정의하며, 비정상 증상(symptom), 원인(cause), 액츄에이터(actuator)의 인스턴스와 이들 간의 의존 관계를 명시한다. 그리고, SCA 프로파일을 기반으로 서비스의 결함을 자율적으로 관리하기 위해, SCA 컴퓨팅 모델의 다섯 단계를 수행하는데 필요한 알고리즘을 상세하게 기술한다. 마지막으로, 제안한 SCA 모델과 알고리즘의 실행 가능성을 보여주기 위하여, SCA 프로파일과 알고리즘을 구현한 프로토타입을 통해 '비행기 예약 시스템'에서 문제가 있는 서비스를 식별하고, 원인을 분석하여, 이를 해결하는 사례 연구를 수행한다. 우리는 SCA 모델의 유효성을 보여주기 위해 준정형적인(semi-formal) 방식을 이용하지만, 자율적인 서비스 컴퓨팅에서 SCA 모델을 적용하기 위하여 실용적인 가이드라인을 제시한다.

2. Symptom-Cause-Actuator 모델

2.1 SCA의 결함 관리 프로세스

SCA의 결함 관리 프로세스는 의사가 환자를 치료하는 과정의 분석을 통하여 유도되었으며, 그림 1과 같은 다섯 개의 단계로 구성된다.

'비정상 증상 발견' 단계에서는 SOA 서비스가 실행(Invoke)되는 전반적인 과정을 감시하여, 비정상 증상을 식별한다. 비정상 증상이란 서비스를 식별, 조립, 실행하는 일련의 과정에서 감시된 값이 정상치 혹은 예상치와 차이를 나타낼 경우, 그 원인을 식별하고 궁극적으로 그 원인에 대한 Actuation(치료)을 해야 하는 어떤 SOA 요소항목의 상태에 대한 명세이다. SOA에서는 여러 개의 관련된 비정상 증상들이 동시에 발행될 수 있다.

'원인 결정' 단계에서는 SCA 프로파일을 기반으로 식별된 비정상 증상을 야기한 원인을 판단한다. SCA 프

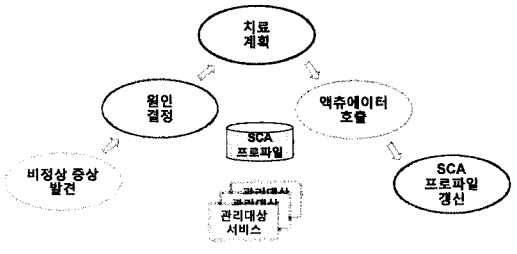

그림 1 SCA의 결함 관리 프로세스

로파일은 비정상 증상에 대한 진단 기록을 관리하고, 비정상 증상을 야기한 원인 판단 기록 및 비정상 증상과 원인 관계의 신뢰값을 관리한다. SCA 프로파일에 대한 자세한 사항들은 3장에서 다룬다.

'액츄에이터 계획' 단계에서는 결정된 원인을 해결할 수 있는 가장 효과적인 치료 방법을 결정하고, 치료 방법에 따른 액츄에이터의 실행 계획을 세운다. SCA 프로파일은 원인에 대한 치료 결정 기록 및 원인과 액츄에이터 관계의 신뢰값을 관리하고, 결정에 따른 액츄에이터 계획 기록을 관리한다. 액츄에이터의 실행 계획은 현재의 SOA 환경, 관리되는 서비스의 특징, 서비스의 현재 상태가 반드시 고려되어야 하므로 상당히 기술적인 작업이다.

'액츄에이터 호출' 단계에서는 서비스 혹은 서비스가 수행되고 있는 SOA 환경에서의 원인을 해결하기 위하여 액츄에이터를 실행한다. 치료는 관리되는 서비스들의 내부 또는 외부(즉, 환경)에서 실행된다. 내부적인 치료를 위해 관리 서비스는 치료을 지원하는 인터페이스를 구현해야 하며 외부적인 치료는 일반적으로 SOA 미들웨어 또는 엔터프라이즈 서비스 버스(Enterprise Service Bus) 인터페이스와 같은 서비스 버스를 통해 실행된다.

'SCA 프로파일 갱신' 단계에서는 비정상 증상과 원인, 원인과 액츄에이터, 비정상 증상과 원인과 액츄에이터와 같은 관계의 신뢰값(confidence)를 갱신한다. 이 관계는 3장 2절에서 상세하게 정의된다. SCA 프로파일은 치료를 통해 획득할 수 있는 새로운 기록과 측정값, 서비스 사용자에 의해 평가되는 만족도로 갱신되어야 한다. 이 프로파일을 갱신하는 것은 원인을 보다 정확하게 진단하고, 보다 효과적인 치료 방법을 결정하기 위해 매우 중요하다. 프로파일 갱신을 위한 효과적인 알고리즘은 4장에서 다룬다.

2.2 메타 모델

본 절에서는 그림 2와 같이 SCA의 결함 관리 프로세스를 위한 3가지 요소와 요소 간의 관계를 정의한다.

비정상 증상은 서비스를 식별, 조립, 실행 하는 일련의 과정에서 감시된 값이 정상치 혹은 예상치와 차이를

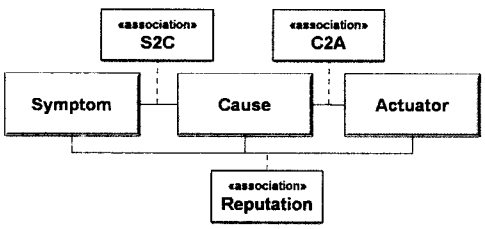

그림 2 SCA의 메타모델

나타낼 경우, 그 원인을 식별하고 궁극적으로 그 원인에 대한 치료를 해야 하는 어떤 SOA 요소항목의 상태로 정의된다. 비정상 증상의 예로는 '구글 맵 서비스 응답 없음,' 'BEA AquaLogic 서비스 레지스트리 응답 느림,' '기상 정보 검색 서비스를 위한 WSDL에서 예외 발생' 등이 있다. 발생할 수 있는 모든 비정상 증상의 집합을 *symptomSet*이라 하고, 집합 내의 한 비정상 증상을 $symptom_i$라고 정의하면, 다음과 같이 집합과 원소 간의 관계로 표현할 수 있다.

$$symptom_i \in symptomSet$$

원인은 비정상 증상을 직접 혹은 간접적으로 발생시킬 수 있는 원인으로 정의된다. 주어진 하나의 비정상 증상에 대하여 관련 있는 원인은 여러 개 존재할 수 있다. 또한, 하나의 원인은 하나 이상의 비정상 증상을 발생시킬 수 있다. 예를 들어, '기상 정보 검색 서비스를 위한 WSDL 예외 발생' 비정상 증상은 하나 이상의 원인인 '인증받지 못한 서비스 사용자의 유효하지 않은 호출,' '기상 정보 검색 서비스 갱신에 따른 WSDL이 갱신 되고, 기존 기상 정보 검색 서비스를 위한 WSDL과는 호환되지 않음'으로부터 발생될 수 있다.

비정상 증상의 원인이 될 수 있는 모든 원인의 집합을 *causeSet*이라하고, 집합 내의 한 원인을 $cause_i$라고 정의하면, 다음과 같은 관계로 표현할 수 있다.

$$cause_i \in causeSet$$

하나의 원인에 대한 액츄에이터는 원인이 더이상 관련된 비정상 증상을 발생시키지 않는 수준까지 원인을 효과적으로 치료하는 활동 또는 메카니즘을 수행하는 모듈로 정의된다. 하나의 원인에 대해 액츄에이터는 하나도 존재하지 않거나 한 가지 이상의 액츄에이터에 의해 치료될 수 있고, 하나의 액츄에이터가 여러 원인을 치료할 수도 있다. 원인이 발생한 구성 요소를 삭제하고 이를 대체할 수 있는 새로운 구성 요소를 추가함으로써 서비스 구성 요소의 배치(configuration)를 수정하거나, 구성 요소를 변경/진화시키는 활동들이 대표적인 형태의 액츄에이터라고 할 수 있다[7]. '기상 정보 검색 서비스 WSDL 갱신로 인해 기존 기상 정보 검색 서비스를 위한 WSDL과는 호환되지 않음' 이라는 원인을 해결할 수 있는 액츄에이터의 예로는 '새로운 기상 정보 검색 서비스 호출을 위한 WSDL의 시그내처 교체,' '새롭게 교체된 서비스 엔드포인트(Endpoint)로의 적응' 등이 있다.

원인을 해결할 수 있는 모든 액츄에이터들의 집합을 *actuatorSet*이라하고, 집합 내의 한 액츄에이터를 $actuator_i$라고 하면 다음과 같은 관계로 표현할 수 있다.

$$actuator_i \in actuatorSet$$

*S2C*는 비정상 증상과 원인 사이의 양방향 관계를 나타내고, 다음과 같이 (비정상 증상, 원인, 신뢰값)의 튜

플(tuple) 형태로 정의된다.

$$s2c = (symptom_x, cause_y, confidence(symptom_x, cause_y))$$

여기서 $symptom_x$과 $cause_y$는 $symptom_x \in symptomSet$, $cause_y$ $causeSet$를 만족시켜야 하고, confidence(symptom$_x$, cause$_y$)는 cause$_y$에 의해 symptom$_x$이 발생될 확률을 나타낸다.

*C2A*는 원인과 액츄에이터 사이의 양방향 관계를 나타내고, 다음과 같이 (원인, 액츄에이터, 신뢰값)의 튜플 형태로 정의된다.

$$c2a = (cause_x, actuator_y, confidence(cause_y, actuator_z))$$

여기서 $cause_x$와 $actuator_y$는 $cause_x$ $causeSet$, $actuator_y$ $actuatorSet$를 만족시켜야 하고, confidence는 cause$_x$가 actuator$_y$에 의해 치료될 확률을 나타낸다.

*평판(Reputation)*은 비정상 증상, 원인, 그리고 액츄에이터 사이의 3중 관계를 나타내고, 다음과 같이 (비정상 증상, 원인, 액츄에이터, 효율성(effectiveness))의 튜플 형태로 정의된다.

$$Reputation = (symptom_x, cause_y, actuator_z, effectiveness(symptom_x, cause_y, actuator_z))$$

여기서 $symptom_x$, $cause_y$, $actuator_z$는 symptom$_x$ ∈ symptomSet, cause$_y$ ∈ causeSet, actuator$_z$ ∈ actuatorSet를 만족시켜야 하고, effectiveness(symptom$_x$, cause$_y$, actuator$_z$)는 $symptom_x$가 $cause_y$에 의해 발생하며, $cause_y$가 $actuator_z$에 의해 성공적으로 치료될 수 있는 전반적인 확률을 나타낸다.

그러므로, 평판의 효율성은 s2c와 c2a의 신뢰값과 관련이 있다. 효율성을 사용하는 데에는 두 가지 주된 목적이 있다. 하나는 한 쌍의 신뢰값보다는 비정상 증상, 원인, 액츄에이터의 세가지 요소를 하나의 그룹으로 간주하여, 이 그룹에 대한 효율성을 표현하기 위한 것이고, 다른 하나는 효율성의 값은 두 가지 신뢰값과 서비스 사용자들에 의한 평가를 고려하여 결정하기 위한 것이다. 즉, 효율성은 세가지 요소에 대한 신뢰값만이 아닌 서비스들의 인식도, 서비스 사용자에 의한 평가도를 이용하여 표현된다. 즉, 평판은 서비스 제공자의 신뢰와 서비스 품질에 대한 서비스 사용자의 인식에 대한 포괄적인 지표이다. 신뢰값과 효율성 계산에 대한 세부적인 사항은 4장에서 설명된다.

2.3 MAPE와의 비교

MAPE는 IBM에서 정의한 자율 컴퓨팅에 대한 일반적인 모델로, AC의 일반적인 제어 활동을 보여준다[3]. SCA는 자율적인 서비스 컴퓨팅을 위한 모델이다. 따라서, 두 모델들은 프로세스 모델 측면에서 매우 유사하게 보이지만 중요한 몇가지 차이점이 존재한다. MAPE는 자율 컴퓨팅에 대한 일반적인 모델로 제안되었고 SCA는 자율적인 서비스 컴퓨팅을 위해 특화된 보다 구체적인 모델이므로, SCA는 MAPE의 특화된 형태라고 할 수 있다. 즉, MAPE의 중요 개념들과 단계들이 SCA에서 관찰되어진다. 하지만 SCA는 SOA 환경을 위한 추가적인 구조, 의미, 세부적인 부분에 대하여 구체적으로 정의한다.

프로세스 모델 측면에서 SCA의 '비정상 증상 발견' 단계는 MAPE의 '모니터' 단계와 '분석' 단계의 일부분과 일치한다. MAPE의 '분석' 단계의 나머지 부분은 SCA의 '원인 결정' 단계와 일치한다. SCA는 새롭게 계산된 확률값들과 평판에 대한 효율성을 SCA 프로파일에 갱신하기 위한 추가적인 단계를 포함한다. SCA 모델에서는 자율적인 서비스 관리 행위가 가능하기 위해 필요한 지식 베이스인 SCA 프로파일를 정의한다. 더욱이, 다섯 단계들의 수행에 필요한 특수한 알고리즘들을 제공한다. 특히, 우리는 신뢰값과 효율성을 계산하기 위한 지침과 공식들을 제공한다. 또한 SCA에서는 서비스 사용자 측면에서 액츄에이터들의 효율성을 계산하는 지침을 제공한다.

3. Symptom-Cause-Actuator 프로파일

3.1 SCA 프로파일 스키마

SCA 프로파일은 비정상 증상, 원인, 액츄에이터와 그들 간의 관계를 관리하고 있는 저장소이다. 그림 3은 메타 모델에서 정의한 요소에 속성을 기술함으로써 SCA 프로파일 스키마의 구조를 보여준다. 또한, 관계 엔티티를 나타내는 S2C, C2A, 평판은 신뢰값과 효율성을 관리하기 위한 속성을 기술하고 있다.

비정상 증상의 내용은 표 1과 같이 관리한다. 결정 기준(Decision Criteria)은 모니터된 결과가 비정상 증상인

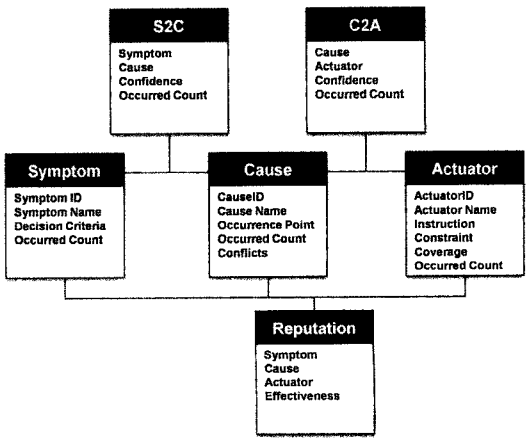

그림 3 구조적 관점에서의 SCA 프로파일

표 1 비정상 증상 스키마와 사용 사례

비정상 증상 ID	Symptom-017-22c
비정상 증상 이름	서비스 응답 없음
결정 기준	응답 시간 > 사용자가 기대한 응답 시간
발생 횟수	105

지 판단하기 위한 기준으로, 모니터된 결과와 한계값으로 이루어진 하나 이상의 명제 함수(propositional function)로 표현된다. 명제 함수를 구성하는 변수는 대상 요소(target element)의 매개 변수(parameter)를 모니터한 결과가 되고, 술어는 비교 연산자와 한계값 (threshold value)으로 표현된다. 비교 연산자는 =, <=, >= 과 같은 부등호를 이용하여 기술하고, 비교 연산자의 왼쪽에는 모니터된 결과, 오른쪽에는 한계값을 할당함으로써 서로의 값을 비교한다. 한계값은 서비스가 제공해야 하는 각 요소에 대한 최소한의 허용 가능한 값이며, default 값을 사용하거나 사용자가 정의하여 사용한다.

원인은 표 2와 같이 관리한다. 발생 지점은 원인이 발생한 위치를 의미하며, 이 속성의 값은 모니터된 결과의 문맥(context), 비정상 증상이 발견된 지점, 원인의 종류 등의 정보를 통합하여 획득한다. 충돌의 경우는 동시에 발생할 수 없는 원인 ID 리스트를 관리하기 위한 속성이다. 원인은 특정 발생 지점을 가지고 있는데, 이 발생 지점이 상이하게 다른 경우 동시에 발생할 수는 없다. 예를 들어 'Cause-122-334x'의 발생 지점은 WSDL이고, 'Cause-422-644z'의 발생 지점은 네트워크 연결(Network connection)이라고 하자. 이런 경우 원인들의 발생 지점이 상이하여 동시에 발생할 수 없으므로 충돌 항목에서 관리한다.

액츄에이터는 표 3과 같이 관리한다. 지침은 원인들을 치료하기 위해 수행되어야 일련의 단일 작업으로 이루어진다. 제약사항은 액츄에이터가 적용될 수 있는 상황을 기술한다. 적용 범위는 액츄에이터가 적용될 수 있는 기능적 범위를 표현한다. 문제 해결을 위해서 액츄에이터가 하나 이상 실행될 경우, 액츄에이터가 적용될 수 있는 제약사항끼리 충돌이 발생할 수 있고, 적용 범위가 겹쳐서 액츄에이터의 효용성이 떨어질 수 있다. 즉, 제약 사항 및 적용 범위에 기반하여 액츄에이터 간의 선택 및 실행 순서를 결정하여야 한다.

표 2 원인 스키마와 사용 사례

원인 ID	Cause-122-334x
원인 이름	서비스 인터페이스인 WSDL이 갱신되고, 기존 WSDL과 호환되지 않음
발생 지점	WSDL
발생 횟수	84
충돌	Cause-422-644z

3.2 SCA 프로파일의 내용

그림 4는 SCA 프로파일의 내용의 일부분을 보여준다. S2C는 비정상 증상과 원인, 이에 따른 신뢰값으로 설명된다. 예를 들어, *'Symptom-017-22c* 서비스 응답 없음'와 *'Cause-122-334x* 서비스 인터페이스인 WSDL이 갱신 되고, 기존 WSDL과 호환되지 않음'는 S2C 인스턴스에서 신뢰값 0.8로 선언되었다. 즉, *Symptom-017-22c*의 원인이 *Cause-122-334x*일 가능성은 80%라는 것을 의미한다. C2A는 원인과 액츄에이터, 이에 따른 신뢰값으로 설명된다. 예를 들어, '서비스 인터페이스인 WSDL이 갱신되고, 기존 WSDL과 호환되지 않음'와 *'Actuator-311-723b* 서비스 Endpoint 적응'는 C2A 인스턴스에서 신뢰값 0.5로 *Cause-122-334x*를 *Actuator-311-723b*가 치료할 가능성은 50%라는 것을 의미한다. 평판은 비정상 증상, 원인, 액츄에이터, 이들 간의 효율성으로 설명된다. 예를 들어, 평판 인스턴스에서 *Symptom-017-22c*, *Cause-122-334x*, *Actuator-311-723b*, 효율성이 0.9로 선언되어 있다면, 이는 *Symptom-017-22c*의 원인이 *Cause-122-334x*이고, *Cause-122-334x*가 *Actuator-311-723b*에 의해 치료할 확률 및 평가도는 0.9임을 의미한다.

4. SCA 컴퓨팅을 위한 알고리즘

본 장에서는 SCA 프로파일을 컴퓨팅 모델의 5개 단계에 적용하여, 각 단계를 어떻게 수행해야 하는지, 각 단계를 잘 수행하기 위해 중요한 요소인 신뢰값과 효율성 등을 포함한 SCA 프로파일을 어떻게 갱신하는지에

표 3 액츄에이터 스키마와 사용 사례

액츄에이터 ID	Actuator-311-723b
액츄에이터 이름	서비스 엔드포인트 적응
지침	1. 변수 endpoint_new를 선언하고, 새로운 엔드포인트 값을 이 변수에 할당 2. 변수 endpoint_older를 선언하고, 예전 엔드포인트 값을 이 변수에 할당 3. 요청된 엔드포인트 값이 endpoint_older에 할당된 값과 동일하면, endpoint_new 에 할당된 값으로 라우팅
제약 사항	새로운 엔드포인트와 요청된 엔드포인트는 유효하여야 함 WSDL 인터페이스는 서비스의 바인딩정보를 갖고 있어야 함
적용 범위	WSDL 사이의 엔드포인트 불일치
발생 횟수	42

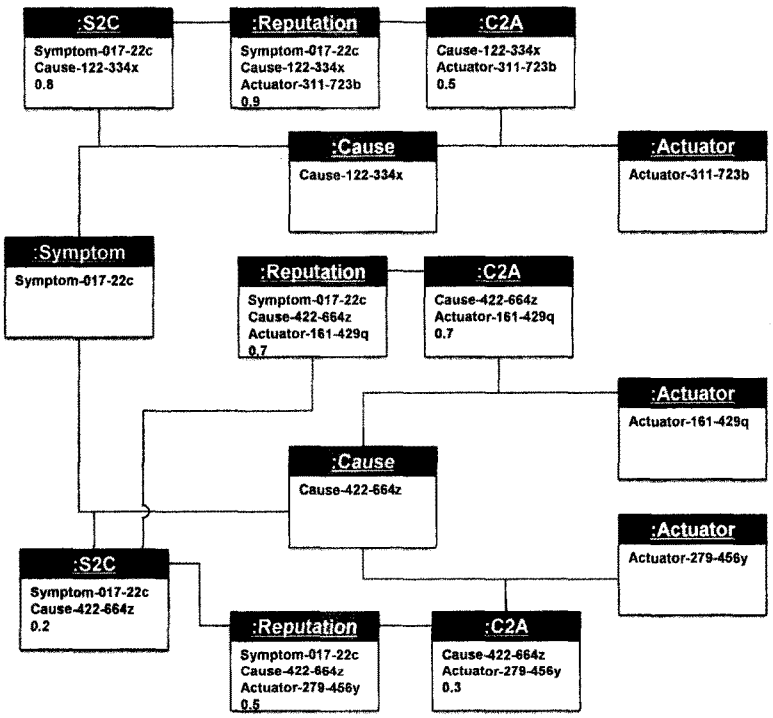

그림 4 SCA 프로파일의 단편

대한 알고리즘을 제안한다.

4.1 '비정상 증상 발견' 알고리즘

단계 1

단계 1은 모니터링 컴포넌트가 서비스 컴포넌트, 서비스 인터페이스, 엔터프라이즈 서비스 버스와 같은 SCA 컴퓨팅의 관리 대상 요소(target element)로부터 응답 시간, 가용성(availability) 등과 같은 서비스 호출 정보, 서비스의 상태 정보, 환경 변화 정보 등을 가져오는 것이다. 이 때 모니터링 컴포넌트는 요청-응답(Request-Respone)와 알림(Notification) 메커니즘을 사용할 수 있다[11]. 이와 같은 과정은 다음과 같은 함수로 정의한다.

$$NotifyMonitoredResult(monitoringComponent) = MonitoredResult_{parameter}^{targetElement}$$

$$getMonitoredResult(targetElement) = MonitoredResult_{parameter}^{targetElement}$$

모니터링 결과($MonitoredResult_{parameter}^{targetElement}$)는 관리 대상 요소로 구분되고, 모니터링 매개 변수와 이에 해당하는 모니터된 값(monitored value)으로 구성되어 있고, 다음과 같은 튜플로 정의한다.

$$MonitoringResult_{parameter}^{targetElement} = (parameter, monitoredValue)$$

매개 변수의 예로는 서비스 응답 시간(service response time), ESB로 들어오는 입력, 나가는 출력 메시지 개수 등이 있다. 모니터된 값은 매개변수에 대응되는데, 예를 들어 50ms는 서비스 응답 시간에 대응될 수 있다.

단계 2

측정한 값을 기반으로 비정상 증상 여부를 판단하기 위해 SCA 프로파일은 비교 연산자와 한계값(*thresholdValue*)으로 구성된 결정 기준을 관리하고 있다. 결정 기준에 따른 결과가 참 또는 거짓인지를 확인하고, 참이면 비정상 증상으로 판단한다.

단계 3

비정상 증상으로 판단된 결과에 한하여 대상 요소에 대한 *monitoredValue*와 *thresholdValue* 사이의 편차를 이용하여 치명적인 문제를 확인한다. 여기서 편차는 측정한 값과 서비스 사용자나 서비스 제공자의 기대치 사이의 차이를 확인하는 과정으로, 비정상 증상의 상태가 어떤지 판단하는 기준이 된다. 측정된 값이 한계값보다 높을 때 정상인 경우는 (1)과 같은 함수로 나타나고, 측정된 값이 한계값보다 낮을 때 정상인 경우는 (2)와 같은 함수로 나타낸다.

$$deviation\ (monitoredValue) = (thresholdValue - monitoredValue) \ / \ thresholdValue \quad (1)$$

$$deviation\ (monitoredValue) = (monitoredValue - thresholdValue) \ / \ thresholdValue \quad (2)$$

예를 들어, 비행기 예약 서비스 컴포넌트의 분당 처리량으로 미리 정의된 한계값이 100건이고 모니터된 값이 60건인 경우가 있다. 처리량의 경우, 모니터된 결과가 미리 정의된 한계값 보다 높은 값을 가져야 하므로 함수 (1)을 이용하여 편차를 계산하면 0.4라는 결과를 얻을 수 있다. 또한, 이 컴포넌트의 응답시간의 경우, 미리 정의된 한계값이 50ms이고 모니터된 값이 100ms으로 비정상 증상이 발생할 수 있다. 응답 시간의 경우, 모니터된 결과가 미리 정의된 한계값보다 낮은 값을 가져야 하므로 함수 (2)를 이용하여 편차를 계산하면 1이라는 결과를 얻을 수 있다.

deviation 함수를 통하여 모니터된 결과와 한계값의 차이나는 정도를 얻을 수 있으며, 계산된 deviation 값의 집합을 weight를 곱하여 다음과 같이 *criticality*(치명도)를 구할 수 있다;

$$criticality\ (parameter) = weight \times deviation\ (monitoredValue)$$

weight는 각 비정상 증상이 발생한 부분의 중요도를 의미하며, 이는 서비스 제공자나 관리자가 0부터 1사이의 값으로 할당한다. 편차와 중요도를 곱하면 감시 대상의 특정 상태가 전체 문제에 대해 얼마나 심각한 영향을 주고 있는지 판단할 수 있다. 즉, 각각의 치명도 값은 모니터된 값에 대해 문제가 있는 정도를 표현하는 지표가 될 수 있다. 예를 들어, 비행기 예약 서비스 컴포넌트의 분당 처리량과 응답 시간이 매개 변수이고, 각각의 중요도가 40%, 60%라고 하자. 위에서 계산된 편차를 이용하여 치명도를 구하면, 분당 처리량의 치명도는 0.16이고 응답 시간의 치명도는 0.6이다. 즉, 비행기 예약 서비스의 경우, 응답 시간이 좀더 문제가 있다고 판단할 수 있다.

단계 4

단계 3에서 계산된 criticality 값들을 이용하여 모니터된 대상의 심각도(Severity)를 계산한다. 특정 관리 대상 요소로부터 모니터된 값에 대한 criticality를 모두 합하면 감시 관리 대상 요소의 상태가 전체 문제에 대해 얼마나 심각한 영향을 주고 있는지 판단하는 기준이 될 수 있다. 계산을 위한 함수는 다음과 같이 표현할 수 있다.

$$\forall targetElement, severity(targetElement) = \sum criticality(parameter)$$

심각도 함수를 통하여 두 가지를 알 수 있는데, 감시 대상의 심각도와 감시 관리 대상 간의 우선 순위를 알 수 있다[11,12]. 비정상 증상 간의 우선순위는 먼저 치료할 대상을 결정하는 데에 있어 판단 기준으로 사용될 수 있다. 일반적으로 의존성 있는 비정상 증상 여러 개가 동시에 발생하고 여러 관리 대상 요소에서 발생하기 때문에 우선 순위에 따라 비정상 증상의 집합을 순서화 할 필요가 있다.

4.2 '원인 결정' 알고리즘

단계 1

먼저, S2C에 있는 정보를 기반으로 발견된 $symptom_i$를 발생시킬 수 있는 모든 원인을 찾는다. SCA 프로파일에 있는 S2C에는 서로 관련있는 비정상 증상과 원인의 ID를 저장하고 있는데, 발견된 $symptom_i$의 ID를 이용하면 관련있는 모든 원인의 집합, *possibleCauseSet*을 얻을 수 있다. 이 집합에서 하나의 가능한 원인을 $possibleCause_i$라고 정의하면, 다음과 같은 관계로 표현할 수 있다.

$$possibleCause_i \in possibleCauseSet,\ possibleCauseSet \subset causeSet$$

*possibleCauseSet*에 있는 원인 중에 정의되지 않은 *(unknown)* 원인이 있다면, 이를 제거하고 SCA 프로파일 갱신할 내역으로 로그에 저장한다. 여기서 정의되지 않은 원인이란 하나 이상의 비정상 증상들을 발생시키는 원인이지만 SCA 프로파일에서 미리 정의하여 관리하지 않은 원인을 의미한다. '정의되지 않은 원인'는 도메인에 따라 원인이 가변적으로 나타날 수 있어서 미리 정의해 놓는 것이 비효율적인 경우와 무엇이 원인인지 정확한 정보가 없는 경우에 정의된다. 즉, 정의되지 않은 원인은 SCA 프로파일을 갱신하기 전에는 해당 원인에 대한 정보가 없는 경우를 의미한다.

$symptom_i$을 발생시킬 수 있는 원인에 대해 기술하지 않고 $unknownCause_i$로 선언해 놓은 경우, 이런 원인을 모아놓은 집합을 *unknownCauseSet*이라 정의하고, 가능한 원인의 집합인 *possibleCauseSet*에서 *unknownCauseSet*을 제거한 집합을 *refinedCauseSet*이라 정의하면 다음과 같은 관계로 표현할 수 있다.

$$unknownCause_i \in unknownCauseSet,\ unknownCauseSet \subset causeSet$$
$$refinedCauseSet = possibleCauseSet - unknownCauseSet$$

단계 2

단계 2에서는 *refinedCauseSet*에 있는 $cause_j$ 중에 해결 방법이 없는 원인을 찾는 것이 목적이다. 즉, 이번 단계에서는 SCA 프로파일의 평판 정보를 이용하여 해결할 수 없는(insoluble) 원인을 제거한다. 여기서 해결할 수 없는 원인이란 원인에 대한 정보는 있으나 이를 치료할 수 있는 액츄에이터가 없는 원인을 의미한다. 즉, *refinedCauseSet*에 있는 원인의 ID를 이용하여 평판에 있는 액츄에이터의 ID가 없는 경우를 찾아서 최종 후보 원인 집합을 만드는 것이다.

*refinedCauseSet*에 있는 원인 중에 액츄에이터와의 관계가 없는 해결 불가능한 원인의 집합을 *insoluble-CauseSet*이라 하고, 집합 내의 하나의 해결 불가능한 원인을 $insolubleCasue_i$라고 하며, $symptom_i$에 대한 후보 원인의 집합을 *candidateCauseSet*이라고 정의하면 다음과 같은 관계로 표현할 수 있다.

$$insolubleCause_i \in insolubleCauseSet,$$
$$insolubleCauseSet \subset causeSet$$
$$candidateCauseSet = refinedCauseSet - insolubleCauseSet$$

단계 3

후보 원인 집합 중에서 가능성이 가장 높고, 서비스 사용자에게 평가가 가장 높은 것을 선택한다. 즉, 이 단계의 목적은 S2C에 저장되어 있는 비정상 증상과 원인 간의 관계에 대한 신뢰값과 원인이 성공적으로 해결될 수 있는지에 대한 효율성을 계산하고, 가장 높은 값을 갖는 원인을 결정하는 것이다.

$confidence(symptom_i, cause_j)$는 $cause_j$가 $symptom_i$를 발생시킬 확률을 가리키는 신뢰값이다. 신뢰값은 SCA 프로파일에 저장되어있는 통계를 기반으로 하며 0에서 1까지의 값을 가진다. $effectiveness(symptom_i, cause_j, actuator_k)$는 $symptom_i$가 $cause_j$에 의해 발생 가능하고 $cause_j$는 $actuator_k$에 의해 성공적으로 치료가 가능한 전반적인 확률을 표현하는 효율성이다. 효율성은 SCA 프로파일에 저장되어 있는 통계를 기반으로 하며 0에서 1까지의 값을 가진다.

효율성을 원인 결정에 사용하기 위해서는, 우선 $symptom_i$와 $cause_j$는 고정 시키고 $cause_j$를 해결할 수 있는 *actuator*만 변화시켜서 나온 효율성을 모두 합한다. 발견된 $symptom_i$와 가능한 $cause_j$에 대한 효율성의 총합을 통하여 우리는 두가지 요소를 응용할 수 있다. 첫째, 얼마나 성공적으로 $cause_j$가 치료될 수 있는지에 대한 판단 기준으로 사용할 수 있다. 또한, 얼마나 많은 서비스 사용자들이 발견된 결과에 만족하는지에 대해 간접적인 판단 기준으로 사용할 수 있다.

발견된 $symptom_i$와 가능한 $cause_j$를 통하여 알아낼 수 있는 효율성의 합을 *sumOfCauseRating*이라하면 다음과 같은 수식으로 표현할 수 있다. 이 식에서 $actuator(symptomi,cause_j)$는 $cause_j$가 $symptom_i$를 발생시키는 경우 이를 치료할 수 있는 액츄에이터를 의미한다.

$$sumOfCauseRating = \sum effectiveness(symtom_i, cause_j, actuator(symptom_i, cause_j))$$

그런 다음, 주어진 $confidence(symptom_i, cause_j)$와 *sumOfCauseRating*의 평균값을 곱한다. *causeAccuracy*의 결과값이 클수록 $cause_j$가 $symptom_i$를 발생시킬 확률이 높고 하나 이상의 액츄에이터에 의해 치료될 확률이 높음을 의미한다.

$$causeAccuracy = confidence(symptom_i, cause_j) \times (sumOfCauseRating \,/\, n)$$

이 식에서 n은 $cause_j$가 $symptom_i$ 를 발생시키는 경우 이를 치료할 수 있는 액츄에이터의 개수를 의미한다. (예. 그림 5의 경우 n은 3개가 됨) *causeAccuracy*의 결과가 같은 경우 SCA 프로파일 에서 관리하고 있는 충돌 항목의 원인 목록들을 기반으로 동시에 발생할 수 없는 원인을 분류하여 가장 알맞은 원인을 결정한다.

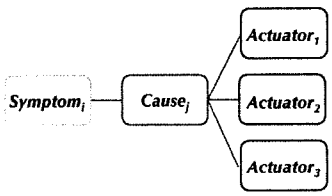

그림 5 $Cause_j$가 $Symptom_i$을 발생시키는 경우 치료할 수 있는 액츄에이터의 예

4.3 '치료 계획' 알고리즘

단계 1

먼저, C2A에 있는 정보를 기반으로 결정된 $cause_j$를 해결할 수 있는 모든 액츄에이터를 찾는다. SCA 프로파일에 있는 C2A에는 관련있는 원인과 액츄에이터의 ID를 저장하고 있는데, 결정된 $cause_j$의 ID를 이용하면 관련있는 모든 액츄에이터의 집합 *candidateActuator Set*를 얻을 수 있다. 이 집합에서 하나의 가능한 액츄에이터를 $candidateActuator_i$라고 하면 다음과 같은 관계로 표현할 수 있다.

$$candidateActuator_i \in candidateActuatorSet,$$
$$candidateActuatorSet \subset actuatorSet$$

단계 2

후보 액츄에이터 집합 중에서 가능성이 가장 높고, 서비스 사용자의 평가가 가장 높은 것을 선택한다. 즉, 이 단계의 목적은 C2A에 저장되어 있는 원인과 액츄에이터 간의 관계에 대한 신뢰값과 액츄에이터의 치료 성공 효율성을 계산하고, 가장 높은 신뢰성과 치료성공율을 갖는 액츄에이터를 결정하는 것이다. 여기서 $actuator_k$의 치료 성공 효율성(*sumOfActuatorRating*)을 구하려면, $actuator_k$를 고정시키고 관련 있는 *cause* 및 *symptom*을 변화시켜 다음과 같은 수식으로 계산한다.

$$sumOfActuatorRating = \sum effectiveness(symtom(cause(actuator_k)), cause(actuator_k), actuator_k)$$

이 식에서 $cause(actuator_k)$는 $actuator_k$가 치료한 원

인을 의미하고, *symptom(cause(actuator$_k$))*는 이런 원인에 의해 발생한 비정상 증상을 의미한다. 그런 다음, 주어진 *confidence(cause$_j$, actuator$_k$)*을 *sumOfActuator Rating*의 평균값과 곱한다. *actuatorSuitability*의 값이 높을수록 *actuator$_j$*가 적절히 원인을 치료할 수 있음을 의미한다.

$$actuatorSuitability = confidence(cause_j, actuator_k) \times (sumOfActuatorRating / n)$$

이 식에서 *n*은 *actuator$_k$*가 해결한 비정상 증상의 개수를 의미한다. (예. 그림 6과 같은 경우 n은 3이 됨) *actuatorSuitability*가 같은 경우 SCA 프로파일의 액츄에이터에서 관리하고 있는 제약 사항 항목에 기술되어진 내용을 기반으로 같이 발생할 수 없는 액츄에이터를 분류하고 적용 범위의 기능 범위가 가장 알맞은 액츄에이터 집합을 결정한다.

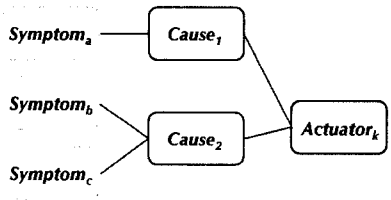

그림 6 Actuator$_k$가 치료할 수 있는 원인과 원인이 발생시킨 비정상 증상의 예

4.4 '액츄에이터 호출' 알고리즘

'비정상 증상 발견' 알고리즘에서 정의한 세 가지 단계, '원인 결정'에서 정의한 세 가지 단계, '치료 계획'에서 정의한 단계 1과 2를 통해서 비정상 증상, 원인, 액츄에이터로 이루어진 경로를 여러 개 얻을 수 있고, 이를 다음과 같이 튜플로 표현할 수 있다.

$$SCAPath_{ijk} = (symptom_i, cause_j, actuator_k)$$

여기서 symptom$_i$ ∈ symptomSet,cause$_j$ ∈ possible CauseSet, actuator$_k$ ∈ candidateActuatorSet를 만족해야 한다. 주어진 경로들의 목록($SCAPath_{111}$, $SCAPath_{112}$...$SCAPath_{ijk}$) 중에서 각 비정상 증상의 심각도를 기반으로 액츄에이터 호출 순서를 결정한다. 결정된 SCAPath의 순서에 따라 액츄에이터의 제약 사항, 적용 범위 등을 고려하여 호출한다. 또한 병렬적으로 호출해야하는 경우, 액츄에이터 간의 의존 관계를 고려하여 호출한다. 이 알고리즘의 경우는 액츄에이터의 종류에 따라 가변적인 상황이 많이 발생하고 이를 일반화하기 어렵기 때문에 자세하게 기술하지는 않겠다.

4.5 'SCA 프로파일 갱신' 알고리즘

단계 1

'액츄에이터' 단계 수행 후 두가지 요소들을 기반으로 선택된 *SCAPath*의 유효성을 입증한다. 첫 번째, 치료 이전 *monitoredValue* 값과 이후 *monitoredValue* 값을 비교하여 특정 대상 요소에서 발생한 비정상 증상을 해결하기 위해 SCAPath에서 결정된 액츄에이터들의 실행 결과에 따른 향상 정도를 의미한다. 측정된 값이 한계값보다 높을 때 정상인 경우는 (1)과 같은 함수로 나타나고, 측정된 값이 한계값보다 낮을 때 정상인 경우는 (2)와 같은 함수로 나타낸다.

$$DegreeOfEnhancement(targetElement) = \frac{monitoredValue(t) - monitoredValue(t-1)}{thresholdValue - monitoredValue(t-1)} \quad \cdots\cdots (1)$$

$$DegreeOfEnhancement(targetElement) = \frac{monitoredValue(t-1) - monitoredValue(t)}{monitoredValue(t-1) - thresholdValue} \quad \cdots\cdots (2)$$

여기서 *monitoredValue(t)*는 치료 이후의 특정 대상 요소의 매개변수에 대해 모니터된 값을 의미하고 *monitoredValue(t-1)*은 비정상 증상이 발생했을 시(치료 이전) 모니터된 값을 의미한다. *DegreeOfEnhancement (targetElement)*의 결과가 1 보다 작은 경우, *SCAPath*가 유효하지 않았음을 의미한다.

두 번째, 액츄에이터의 실행 결과에 대한 서비스 사용자들의 만족도에 기반하여 결정된 등급들을 획득한다. 서비스 사용자의 평가 항목에 대한 만족도는 $satisfaction_{serviceConsumer}(raitingItem)$으로 표현할 수 있다.

특정 액츄에이터의 실행 결과에 대해 서비스 사용자들에 의해 획득된 등급 항목들의 만족도의 평균을 구하는 함수를 *rating(Actuation)*라고 하면 다음과 같이 표현할 수 있다. 여기서 *Actuation*은 SCAPath 목록 중 하나의 결과로서 특정 대상 요소의 매개 변수를 위한 액츄에이터의 실행 결과를 의미한다. 여기서 *maxRaitingTotal*은 한 명의 서비스 사용자가 치료의 결과에 대해 가장 높은 만족도를 부여할 때의 합계이고, n은 서비스 사용자의 수를 의미한다. 만약 raiting(Actuation) 결과가 0이라면, *SCAPath*는 유효하지 않음을 의미한다.

$$rating(Actuation) = \frac{\sum satisfaction_{serviceConsumer}(ratingItem)}{max\,RaitingTotal \times n}$$

단계 2

causeAccuracy, *actuatorSuitability*, *DegreeOfEnhancement*의 값들을 기반으로 신뢰값과 효율성을 갱신한다. 먼저 신뢰값을 계산하기 위한 식은 다음과 같다.

$$confidence(symptom_i, cause_j) = OccurredCount(symptom_i, cause_j) / Count(symptom_i)$$

$$confidence(cause_j, actuator_k) = OccurredCount(cause_j, actuator_k) / Count(cause_j)$$

여기서 $OccurredCount(symptom_i, cause_j)$는 $symptom_i$가 발생했을 경우 $cause_j$가 원인이었던 횟수를 의미하며, $OccurredCount(cause_j, actuator_k)$는 원인이 $cause_j$인 경우 $actuator_k$가 치료한 횟수를 의미한다. 또한, $Count(symptom_i)$는 $symptom_i$가 발생한 횟수를 의미하며, $Count(cause_j)$는 $cause_j$가 원인인 횟수를 의미한다.

치료의 결과에 *causeAccuracy*, *actuatorSuitability*의 값이 각각 영향을 끼치므로, 두 가지 값의 크기를 고려하여 각각의 신뢰값에 반영한다. 또한, *DegreeOfEnhancement* 값을 기반으로 S2C와 C2A의 발생횟수(OccurredCount)를 늘리고 신뢰값을 갱신한다. 여기서 *DegreeOfEnhancement*의 결과에 따라 다음과 같은 경우로 나뉜다. 첫째, *DegreeOfEnhancement*의 값이 1 이상으로 치료의 결과가 만족스러운 경우, 다음과 같이 신뢰값을 계산한다.

$$confidence_{new}(symptom_i, cause_j) = (OccurredCount_{old}(symptom_i, cause_j) + 1) / (Count(symptom_i) + 1)$$

$$confidence_{new}(cause_j, actuator_k) = (OccurredCount_{old}(cause_j, actuator_k) + 1) / (Count(cause_j) + 1)$$

둘째, *DegreeOfEnhancement*의 값이 1 보다 작고 0 보다 큰 경우, 대상 요소의 상태가 좋아졌지만 치료의 결과가 만족스럽지 않기 때문에 SCAPath를 결정하는데 더 많은 영향을 끼친 S2C 혹은 C2A의 신뢰값을 다음과 같이 신뢰값을 계산한다. 단, 영향력이 같은 경우는 S2C, C2A의 신뢰값을 모두 갱신한다.

$$causeAccuracy > actuatorSuitability,\ confidence_{new}(symptom_i, cause_j) = OccurredCount_{old}(symptom_i, cause_j) / (Count(symptom_i) + 1)$$

$$causeAccuracy < actuatorSuitability,\ confidence_{new}(cause_j, actuator_k) = OccurredCount_{old}(cause_j, actuator_k) / (Count(cause_j) + 1)$$

셋째, *DegreeOfEnhancement*의 값이 0 보다 작은 경우, 대상 요소의 상태가 나빠지고 치료의 결과가 만족스럽지 않으므로 다음과 같이 신뢰값을 계산한다.

$$causeAccuracy > actuatorSuitability,\ confidence_{new}(symptom_i, cause_j) = OccurredCount_{old}(symptom_i, cause_j)\ 1 / (Count(symptom_i) + 1)$$

$$causeAccuracy < actuatorSuitability,\ confidence_{new}(cause_j, actuator_k) = OccurredCount_{old}(cause_j, actuator_k)\ 1 / (Count(cause_j) + 1)$$

주어진 평가도와 갱신된 신뢰값들을 이용하여 *SCA-Path* 의 효율성을 다음과 같은 수식을 이용하여 갱신한다. 이 식에서 W_{sc}, W_{ca},와 W_{sca}는 각 부분에 대한 가중치 값이고 총합은 1이다. 이 가중치는 관리 대상, 서비스가 사용되는 도메인에 따라 다르기 때문에 서비스 관리자 혹은 서비스 제공자가 정의한다.

$$effectiveness(symptom_i, cause_j, actuator_k) = confidence(symptom_i, cause_j)W_{sc} + confidence(cause_j, actuator_k)W_{ca} + rating(Actuation_{ijk})W_{sca}$$

5. 관련 연구 및 평가

5.1 관련 연구

Bittenham의 연구는 계층 개념을 이용하여 자율 컴퓨팅의 주요 컴포넌트와 컴포넌트 간의 관계를 표현하는 자율 컴퓨팅 아키텍처를 제안한다[5]. 또한, 이 아키텍처를 이용한 IT 서비스 관리 방법을 기술한다. 먼저, 문제-결정(Problem-Decision) 시나리오를 이용하여 문제와 관련된 이벤트의 예상 순서에 대해 개략적으로 기술하고, 기술된 순서에 기반하여 자율 컴퓨팅의 주요 컴포넌트를 수행함으로써 서비스를 관리한다. 그러나 시나리오는 이벤트의 예상 순서에 대한 개략 기술이므로, 아키텍처를 구성하고 있는 컴포넌트의 상세 설계 수준이나 구현 수준을 다루기 힘들다. 또한, 문제-결정 방법이 시나리오 및 이벤트에 특화되어 있기 때문에 좀더 일반적인 모델로 표현될 필요가 있다.

Brüning의 연구는 서비스의 신뢰성을 유지하고 관리하기 위하여 신뢰도에 영향을 줄 수 있는 서비스 결함의 분류 체계를 제시한다[8]. 이 분류 체계는 서비스 요청과 관련된 5 단계의 프로세스를 기반으로 한다. 이 프로세스는 1) 서비스 제공자가 서비스를 배포하고, 2) 서비스 사용자가 배포된 서비스 중 필요한 서비스를 검색하고, 3, 4) 검색된 서비스를 서비스 사용자 요구에 따라 조합하고 바인드하며, 5) 바인드된 서비스를 실행하는 단계로 구성된다. 이에 따라 결함을 배포 시에 발생하는 결함, 발견 시에 발생하는 결함, 결합 시에 발생하는 결함, 바인딩 시에 발생하는 결함, 실행시에 발생하는 결함의 다섯 가지로 분류한다. 그리고, 다섯 가지 단계에서 발생할 수 있는 세부적인 결함을 식별한다. 서비스 관리는 일반적으로 문제 발견, 원인 분석, 원인 해결의 단계로 이루어지고, 원인 해결을 하기 위해서는 SOA의 특정 대상 요소에 대한 기술이 필요하지만, 이 연구에서는 결함을 서비스 요청과 관련된 프로세스와

연관 지음으로써 원인 해결의 대상을 기술하기 어렵다.

IBM의 연구에서는 시스템 운영시에 발생할 수 있는 문제를 비정상 증상으로 정의하고, 이를 기술하기 위한 구조를 제안한다[9]. 비정상 증상을 기술하기 위해서는 먼저 비정상 증상에 대해 정의하고, 발견된 이벤트를 비정상 증상으로 판단하기 위한 조건과 비정상 증상이 발생하였을 때 시스템에 미칠 수 있는 영향을 정의하였다. 그리고 이런 비정상 증상을 기계적으로 처리하는데 필요한 비정상 증상 엔진을 제안한다. 또한, 비정상 증상에 대해 기술한 내용을 기계적으로 처리하기 위하여 XML 스키마을 사용하며, 이는 비정상 증상을 발견하고, 발견된 비정상 증상 인스턴스를 표현하며, 개발하는데에 사용된다. 그리고, 이 연구는 '보안성', '운용성', '가용성', '서비스 품질', '정의하지 않음'의 다섯 가지 비정상 증상 카테고리를 제안한다. 이와 같은 구조와 비정상 증상 카테고리를 이용하면 비정상 증상과 관련된 정보를 체계적으로 유지하고 관리할 수 있다. 그러나 비정상 증상과 문제의 원인을 구분하지 않았고, 비정상 증상 자체에만 초점을 맞추고 있다.

Martín의 연구는 서비스를 관리하기 위해 온톨로지와 정책 기반의 패러다임을 이용하여, 관리할 서비스를 정의하고, 이를 특정 환경을 고려하여 특화시키며, 그 환경에 배치시켜 운영한다[10]. 또한, 관리 서비스 간의 상호 운용성을 증가시키고, 제어할 수 있는 기능성을 높이기 위하여 문맥(context) 정보를 이용하고 있다. 마지막으로 이를 지원할 수 있는 프레임웍을 제공함으로써 미들웨어 수준에 필요한 기술을 제안하고 있다. 서비스 관리에서 중요한 점은 서비스 사용자의 관점에서 관리해야 할 문제를 정의해야 한다는 것이다. 그러나 일반적인 미들웨어에 특화된 기술을 제시함으로써 SOA의 특화된 부분에 관해서는 초점을 맞추지 않는 경향이 있다.

5.2 평 가

본 절에서는 5.1절에서 조사한 관련 연구와 본 연구의 차이점을 분석한다. 평가 기준은 효과적으로 서비스의 결함을 관리하는데 필요한 네 가지 조건을 기반으로 도출하였다.

"자율 관리 기능"은 서비스의 제한된 관리성을 해결하기 위해서는 서비스가 자율적으로 결점을 발견하여 해결할 수 있는 기법이 필요하기 때문에 평가 기준으로 도출되었다. "SOA 환경과의 적합성"은 일반 시스템과는 다른 SOA만의 고유한 특징을 고려하여 서비스 관리가 이루어져야 하기 때문에 두번째 평가 기준으로 도출되었다. "서비스 관리 대상"은 각 논문에서 제시한 기법을 이용하여 관리될 수 있는 대상을 기술하는 항목으로, 서비스가 기능을 수행하면서 발생하는 결함은 서비스 자체의 결함외에도 관련 요소들의 영향도 있으므로, 세번째 평가 항목으로 도출되었다. "자율 서비스 관리 기능 구현 가능성 제시"는 제안된 기법이 이론상으로만이 아닌 실제 구현이 가능하여 서비스가 결함없이 제공될 수 있어야 하기 때문에 마지막 평가 항목으로 도출되었다.

표 4는 위에서 도출한 항목을 기준으로 본 연구와 관련 연구를 비교/분석한 결과를 보여준다.

Bittenham의 연구는 일반적인 자율 컴퓨팅 아키텍처가 IT 서비스 관리에 어떻게 적용될 수 있는지를 제안하였고 적용 가능한 SOA 관련 표준들을 나열하지만, 이 표준들이 어떻게 자율 컴퓨팅 아키텍처에 적용되는지와 SOA 환경에 특화된 서비스 관리 대상을 구체적으로 제시하지 않았다. Brüning의 연구는 서비스가 운영되는 과정을 따라서 발생할 수 있는 서비스 결함을 식별하였지만, 이 결함들이 어떻게 관리될 수 있는지는 구체적으로 언급하지 않았다. IBM의 연구는 서비스 운영시 발생할 수 있는 비정상 행위를 비정상 증상으로 정의/분류하고 비정상 증상을 XML 스키마를 이용하여 기술함으로써 구현 가능성을 제시하였지만, 비정상 증상은 SOA 환경에 특화된 대상을 기술한 것이 아니라 일반적인 시스템을 대상으로 기술하였다. Martin은 온톨로지와 정책 기반으로 서비스를 관리하는 기법을 제안하여 구현 가능성을 제시하였지만, 자율적인 관리 기법을 이용하지 않았으며 미들웨어 외의 다른 수준에서의 서비스 관리 기법은 다루어지지 않았다.

이에 비해, 본 논문에서는 SCA 프로파일에 저장된 비정상 증상, 원인, 액츄에이터 정보를 이용하여 자율적으로 비정상 증상을 식별하고, 식별된 비정상 증상에 대한 원인을 결정하며, 원인을 해결할 수 있도록 치료 계

표 4 본 연구와의 비교 (√ : 제공)

비교 항목 \ 비교 대상	Bittenham [5]	Brüning [8]	IBM [9]	Martín [10]	본 논문의 모델
자율 관리 기능	√		√		√
SOA 환경과의 적합성	√	√		√	√
서비스 관리 대상	관리대상 구체적으로 언급 없음	SOA 구성요소에 대한 결함 도출	일반 시스템 구성요소 대상	서비스 미들웨어 관리	SOA 핵심 구성요소 관리
자율 서비스 관리 기능 구현 가능성 제시	√		√	√	√

획을 만들고 이에 따른 액츄에이터를 실행하는 알고리즘을 제안하였다. 또한, SCA 프로파일을 자율적으로 업데이트할 수 있도록 SCA 프로파일 갱신 알고리즘을 제안하였다. 이 알고리즘에서는 통계에 기반한 신뢰값 이외에도 서비스 사용자의 평가를 고려하여 효율성을 계산함으로써 사용자의 요구사항에 맞춘 SCA 프로파일을 구성할 수 있도록 제안되었다. 이는 모두 서비스를 자율적으로 관리하기 위한 기법이다. 그리고, SCA 프로파일에는 서비스 뿐 아니라 서비스를 운영하는데 필요한 서비스 미들웨어에 대한 비정상 증상, 원인, 액츄에이터 정보를 기술함으로서 SCA 모델을 이용하여 서비스와 관련 구성 요소까지도 관리 대상이 될 수 있음을 나타내었다. 그리고, 사례연구를 수행함으로써 SCA 모델이 SOA 환경과 어떻게 연동되는지와 이론적인 SCA 모델이 구현이 될 수 있음을 보여주었다.

6. 사례 연구

제안된 SCA 프로파일과 컴퓨팅 알고리즘의 적용 가능성을 보이기 위해, '비행기 예약 시스템'에 적용한 사례연구를 수행한다. '비행기 예약 시스템'은 좌석 조회(Flight Inquiry) 서비스, 좌석 예약(Flight Reservation) 서비스, 결제(Flight Payment) 서비스로 이루어진다. 세 가지 서비스는 좌석 조회 서비스 컴포넌트, 좌석 예약 서비스 컴포넌트, 결제 서비스 컴포넌트로 구현되고, BEA WebLogic 9.2 서버에 배치되어 실행가능한 상태이다. 또한, 서비스들의 정보는 BEA AquaLogic 서비스 레지스트리(Service Registry)에 등록되어 사용자가 검색하여 사용가능하다. 그림 7은 이와 같은 '비행기 예약 시스템'의 전반적인 구성 현황을 보여주고 SCA 컴퓨팅 모듈의 관리 대상을 보여준다.

결함 주입 방법을 이용하여, 좌석 조회 서비스의 응답 시간을 사용자가 원하는 응답 시간보다 높게 조정한다[13]. SCA 프로파일에 저장되어 있는 결정 기준 중 응답 시간에 해당하는 행에 있는 한계값에 30을 할당하고, 좌석 조회 서비스의 응답 시간을 70초로 할당한다. 이로 인해 '비행기 예약 시스템'은 동작하지 않는 것으로 인식된다.

먼저, 비정상 증상 발견을 위하여 SCA 컴퓨팅 모듈을 이용하여 감시할 대상이 배치되어 있는 서버 혹은 미들웨어를 선택한다. 그림 8의 A처럼 BEA WebLogic 9.2 서버, BEA AquaLogic 서비스 레지스트리, BEA AquaLogic Service Bus, Others라는 옵션이 있다. 여기서 결함이 주입되어 있는 좌석 조회 서비스를 선택하기 위해 이 서비스가 배치되어 있는 BEA WebLogic 9.2 서버를 선택한다. 그러면 현재 BEA WebLogic 9.2 서버에 배치되어 있는 서비스 이름, 서비스 상태, 서비스 엔드포인트를 요약하여 그림 8의 B처럼 보여준다. 또한, 좌석 조회 서비스가 비정상 상태임을 알 수 있는데, 자세한 결과를 보면 응답 시간이 70, 분당 처리량은 8, 시간 당 입력 메시지 수가 1500, 출력 메시지 수는 950임을 알 수 있다. 즉, 좌석 조회 서비스의 응답 시간이 70, 응답 시간에 대한 한계값이 30임으로 70 > 30이 성립하게 되어 비교 연산자를 만족하게 된다. 그러므로 결정 기준 'DC01'과 관련 있는 비정상 증상이 발견된다. DC01의 내용은 3.1절의 표 1과 관련 있으므로 그림 8처럼 'Symptom-017-22c'이 비정상 증상으로 발견된다.

발견된 비정상 증상의 원인을 찾기 위해 SCA 프로파일에 저장되어 있는 s2c 테이블의 내용을 참조하면, 그

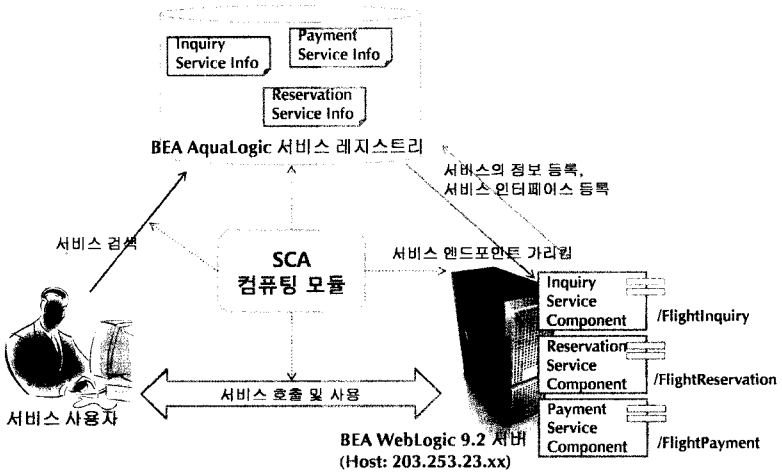

그림 7 비행기 예약 시스템의 구성에 따른 SCA 컴퓨팅 모듈의 관리 대상

그림 8 좌석 조회 서비스 감시 결과 및 비정상 증상 발견

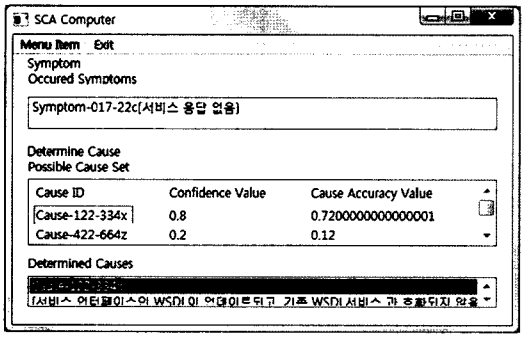

그림 9 Symptom-017-22c에 대한 원인 결정 과정

림 9와 같이 'Cause-122-334x: 서비스 인터페이스인 WSDL이 갱신 되고, 기존 WSDL과 호환되지 않음'과 'Cause-422-664z: BPEL/WSDL 불일치'라는 두 가지 원인이 가능한 원인 집합 (possibleCauseSet)으로 선택된다. 이는 그림 4에 있는 프로파일 내용을 기반으로 하고 있다.

Symptom-017-22c와 Cause-122-334x의 관계에 대해 그림 4 SCA 프로파일에 저장되어 있는 신뢰값은 0.8이고, Cause-123-334x와 관련 있는 모든 평판의 효율성을 합하여 계산하면 0.9이다. 이를 알고리즘에서 제안한 수식에 대입하여 원인 정확도 수치를 구하면 다음과 같다.

$$causeAccuracy = confidence(symptom_i, cause_j) \times (sumOfCauseRating / n) \; (0.8 \times 0.9 / 1) = 0.72 \quad (1)$$

Symptom-017-22c와 Cause-422-664z의 관계에 대해 SCA 프로파일에 저장되어 있는 신뢰값은 0.2이고, Cause-422-664z와 관련 있는 모든 평판의 효율성의 합은 1.2이다. 이 값을 알고리즘에서 제안한 식을 적용하여 원인 정확도 수치를 구하면 다음과 같다.

$$causeAccuracy = confidence(symptom_i, cause_j) \times (sumOfCauseRating / n) = 0.2 \times (1.2/ 2) = 0.12 \quad (2)$$

Symptom-017-22c에 따른 원인의 정확도 수치는 Cause-122-334x의 경우가 더 높으므로, Symptom-017-22c를 발생시킨 원인으로 Cause-122-334x이 결정된다. 이렇게 결정된 원인을 해결할 수 있는 액츄에이터를 SCA 프로파일에 저장되어 있는 c2a 테이블 내용을 참조하여 보면, 'Actuator-311-723b: 서비스 Endpoint 적용'이 검색된다. 그림 10과 같이 Cause-122-334x와 Actuator-311-723b의 관계에 대해 SCA 프로파일에 저장되어 있는 신뢰값은 0.5이고, Actuator-311-723b와 관련 있는 모든 평판의 효율성의 합은 0.9이다. 이 값을 알고리즘에서 제안한 수식에 대입하여 액츄에이터 적합성 수치를 구하면 다음과 같다.

$$actuatorSuitability = confidence(cause_j, actuator_k) \; (sumOfActuatorRating / n) = 0.5 \times (0.9 / 1) = 0.45 \quad (3)$$

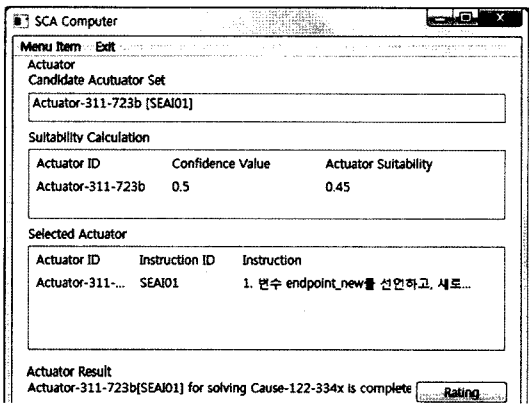

그림 10 Cause-112-334x에 대한 치료 계획 및 실행 결과

Cause-122-334x를 치료할 수 있는 액츄에이터로는 Actuator-311-723b 만이 검색되어 있으므로, 추가적인 비교과정이 없이 해당 액츄에이터를 실행한다. 선택된 액츄에이터는 SCA 프로파일에 저장되어 있는 지침, 제약 사항, 적용 범위에 따라 실행된다.

액츄에이터의 수행 결과는 SCA 프로파일 갱신를 통하여 비정상 증상, 원인, 그리고 액츄에이터 간의 의존성을 나타내는 s2c와 c2a의 신뢰값과 해당 *SCAPath*의 효율성 값을 갱신한다. Actuator-311-723b 수행 결과 목표 서비스의 응답시간이 70에서 30으로 낮아지고, 이를 알고리즘에서 제시한 수식에 대입하여 향상 정도를

계산하면 다음과 같다.

$$DegreeOfEnhancement(Actuator\text{-}311\text{-}723b) = (70 - 30) / (70 - 30) = 1 \quad (1)$$

향상 정도가 1이고, 치료의 수행 결과에 따라 갱신될 신뢰값을 구하기 위해 알고리즘에서 제시된 수식을 이용하면 다음과 같다.

$$confidence_{new}(symptom_i, cause_j) = (84 + 1) / (105 + 1) = 0.80 \quad (2)$$

$$confidence_{new}(cause_j, actuator_k) = (42 + 1) / (84 + 1) = 0.50 \quad (3)$$

Actuator-311-723b 수행 결과에 따른 사용자의 만족도를 효율성 값에 적용하기위해 그림 11과 같이 세 가지 항목 총평(general review), 효율성, 만족도에 대해서 rating을 실시하고, 알고리즘에서 제시한 수식에 대입하여 rating 값을 구하면 다음과 같다.

$$Rating(Actuation_{Symptom\text{-}017\text{-}22cCause\text{-}122\text{-}334xActuator\text{-}311\text{-}733b}) = (1 + 2 + 1) / (9\ 1) = 0.44 \quad (4)$$

마지막으로 지금까지 계산된 값들을 이용하여 수행된 *SCAPath*에 대한 효율성 값을 구하기위해 (2)부터 (4)까지의 가중치를 0.3, 0.4, 0.3로 설정하고 이를 알고리즘에서 제시한 수식에 대입하면 다음과 같다.

효율성 = 0.80 0.3 + 0.50 0.4 + 0.44 0.3 = 0.57 (5)

본 논문에서 제안된 서비스 관리 도구는 모델기반의 추론 모델을 기반으로 설계되었기 때문에, 대상 시스템에 대한 모델링의 완전성과 모니터링을 수행하는 모니터의 정확성에 따라 수행 결과의 정확도가 판단된다. 즉, 모델링된 시스템의 상태에 대해서는 완전한 추론 및 갱신이 가능하지만 그 외의 상태에 대해서는 관리할 수 없다. 따라서 SCA 관리도구의 실행도중 새로운 Symptom의 발생 시, SCA 관리도구는 새로운 symptom에 대한 정보를 관리자에게 통보하고 관리자는 이를 기반으로 새로운 symptom과 관계된 causes, actuators 를 SCA 프로파일에 추가하여 SCA 관리도구의 유효성를 높인다. 즉, SCA의 실행 결과에 대한 유효성은 관리대상을 모니터링하는 모니터의 정확성과 대상 시스템의 전체 상태 중에서 실제 모델링된 결함 모델의 비율로 계산된다;

ValidityOfSCA = CorrectnessOfMonitor × (Numof ModeledSystemState/NumofPossibleSystemState)

위의 식에서 ValidityofSCA는 SCA 실행 결과의 유효성을 나타내며, CorrectnessofMonitor는 관리 대상을 모니터링하는 모니터의 정확성, NumofModeledSystemState는 대상 시스템의 모델링된 상태의 수, 그리고 NumofPossibleSystemState는 대상 시스템의 가능한 모든 상태의 수를 나타낸다. 따라서 SCA의 유효성에 대한 측정 결과는 결함 모델의 완전성에 따라 변화되며, 결함 모델은 SCA 실행에 따른 새로운 symptom의 발견과 관리자의 관련된 모델 추가로 점진적으로 완성된다.

7. 결 론

서비스들의 블랙박스 특성 때문에 서비스 사용자들과 관리자들은 제한된 관리성을 가진다. 또한 서비스들은 변화에 대한 알림없이 발전할 수 있다. 한편, 자율 컴퓨팅은 직접적인 사람의 조정없이 스스로 관리가 가능한 시스템을 설계하는 방법이다. 자율 컴퓨팅의 주요 기법들을 이용하면, 서비스 결함 관리의 대부분 활동들이 자율적으로 행해질 수 있으므로 현재 서비스 결함 관리가 가지는 문제들을 해결할 수 있게 된다.

본 논문에서는 SOA 환경에서 자율적인 서비스 결함 관리가 가능한 SCA의 이론적 모델을 제안하였다. SCA 모델은 의사가 어떻게 환자들을 치료하는지에 대한 정확한 관찰을 통하여 유도되었다. 그리고, SCA의 다섯 단계로 구성된 컴퓨팅 모델과 메타모델을 정의하였고, SCA 모델의 저장소 역할을 하는 SCA 프로파일의 스키마를 정의하였고 사용 예를 SOA에 특화된 요소를 기반으로 보여주었다. SCA 프로파일에 저장된 비정상 증상, 원인, 액츄에이터 정보를 이용하여 자율적으로 비정상 증상을 식별하고, 식별된 비정상 증상에 대한 원인을 결정하며, 원인을 해결할 수 있도록 액츄에이터를 결정하는 알고리즘을 제안하였다. 제안된 알고리즘에서는 S2C, C2A에 저장된 신뢰값과 평판에 저장된 효율성을 이용하여, 가장 확실한 원인을 결정하고 결정된 원인을 가장 효과적으로 해결할 수 있는 액츄에이터를 결정하였다. 또한, SCA 프로파일을 통계에 기반한 신뢰값 이

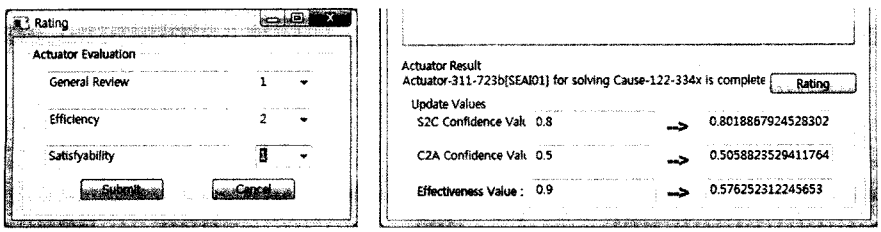

그림 11 SCA 프로파일 갱신 실행 결과

외에도 서비스 사용자의 평가를 고려하여 자율적으로 업데이트할 수 있도록 SCA 프로파일 갱신 알고리즘을 제안하였다. 우리는 SCA 모델의 정당성을 보여주기 위해, 준정형적인 방식을 사용하였지만 자율적인 서비스 컴퓨팅의 SCA 모델을 적용하기 위한 실용적인 가이드라인을 소개하였다. 또한, 실용적인 가이드라인을 기반으로 한 프로토타입 및 '비행 예약 시스템' 적용 사례 연구를 통하여 SCA 모델 및 알고리즘의 구현 가능성 및 유효성을 보여 주었다.

참 고 문 헌

[1] Erl, T., SOA Principles of Service Design, *Prentice Hall*, July, 2007.

[2] Manes, A., "The Elephant Has Left the Building," at URL: http://eai.ittoolbox.com/documents/industry-articles/the-elephant-has-left-the-building-3633, *Intelligent Enterprise*, July, 2005.

[3] Pulier, E. and Taylor, H., *Understanding Enterprise SOA*, Manning Publications, 2005.

[4] Kephart, O. and Chess, M., "The Vision of Autonomic Computing," *IEEE Computer*, IEEE Computer Society Press, Vol. 36, No. 1, pp. 41-50, Jan., 2003.

[5] Brittenham, P., Cutlip, R., Draper, C., Miller, B., Choudhary, S., and Perazolo, M., "IT service management architecture and autonomic computing," *IBM Systems Journal*, IBM, Vol. 46, No, 3, 2007.

[6] Cutlip, R., and Zabeu, C., "Autonomic Computing: Strengthening Manageability for SOA Implementations," *Autonomic Computing White Paper*, IBM, Dec., 2006.

[7] Fugini, M., Mussi, E., "Recovery of Faulty Web Applications through Service Discovery," *In proceedings of the 1st International Workshop on Semantic Matchmaking and Resource Retrieval: Retrieval: Issues and Perspectives (SMR 2006)*, pp. 67-80, 2006.

[8] Brüning, S., Weißleder, S., and Malek, M., "A Fault Taxonomy for Service-Oriented Architecture," *10th IEEE High Assurance Systems Engineering Symposium (HASE'07)*, Nov., 2007.

[9] IBM Research Center, "Symptoms Reference Specification, Version 2.0," *IBM Autonomic Computing Symptom Specification*, IBM, 2006.

[10] Martín, S., Joan, S., John, S., Greg, C., Ray, C., and Mícheál, Ó., "Services Management Using Context Information, Ontologies and the Policy-Based Management Paradigm: Towards Integrated Management in Autonomic Communications," *IEEE Workshop on Autonomic Communications and Network Management (ACNM' 07)*, May, 2007.

[11] Tewari, V. and Milenkovic, M., "Standards for Autonomic Computing," *Intel Technology Journal*, Intel, Vol. 10, No. 04, 9th Nov., 2006.

[12] DMTF, *CIM Schema: Version 2.8.2*, DMTF Standard, DMTF, Jan., 2005.

[13] Arlat, J., Aguera, M., Amat, L., Crouzet, Y., Fabre, J., Laprie, J., Martins, E., and Powell, D., "Fault Injection for Dependability Validation: A Methodology and Some Applications," *IEEE Transaction on Software Engineering*, IEEE, Vol. 16, No. 02, Feb., 1990.

천 두 완

2005년 숭실대학교 컴퓨터학부 공학사
2006년 숭실대학교 컴퓨터학과 공학석사
2006년~현재 숭실대학교 컴퓨터학과 박사과정. 관심분야는 서비스 지향 아키텍처(SOA), 자율 컴퓨팅(AC)

이 재 유

2007년 홍익대학교 컴퓨터정보통신학과 공학사. 2007년~숭실대학교 컴퓨터학과 석사과정. 관심분야는 서비스 지향 아키텍처(SOA), 자율 컴퓨팅(AC)

라 현 정

정보과학회논문지 : 소프트웨어 및 응용 제 35 권 제 3 호 참조

김 수 동

정보과학회논문지 : 소프트웨어 및 응용 제 35 권 제 3 호 참조