韓國컴퓨터情報學會 論文誌
第14卷 第3號, 2009. 3.

2009-14-3-4-3

비즈니스 서비스간의 오류 정제를 위한 데이터 제약조건 자동 설정 기법

이 정 원*

An Automatic Setting Method of Data Constraints for Cleansing Data Errors between Business Services

Jung-Won Lee*

요 약

본 논문에서는 SOA(Service-Oriented Architecture)를 기반으로 서비스 간에 상호 작용하는 데이터의 품질 관리를 위한 오류 정제 서비스를 대상으로 데이터 제약조건 설정 시 인간 개입을 최소화하기 위한 기법을 제안한다. 단, 실세계에서 통용되는 일반적인 데이터를 모두 다루는 것은 불가능하므로 비즈니스 도메인에서 자주 사용되는 CRM(Customer Relationship Management)과 ERP(Enterprise Resource Planning) 서비스와 같이 고객 주문 정보 및 처리에 관련된 데이터를 대상으로 한다. 이를 위해, 컴포지션 되는 서비스간의 상호 작용하는 데이터를 의미적으로 확장하여 확장-엘리먼트 벡터를 생성하고 이를 기반으로 의사결정 트리(decision tree) 학습 방법을 적용하여 제약조건 설정을 자동화 하기위한 규칙 기반 시스템을 구축한다. 이 시스템을 오류 정제 서비스에 삽입한 결과, 비즈니스 분야의 공개된 서비스로부터 데이터 학습을 통해 제약조건 설정을 41% 넘게 자동화 할 수 있음을 보였다.

Abstract

In this paper, we propose an automatic method for setting data constraints of a data cleansing service, which is for managing the quality of data exchanged between composite services based on SOA(Service-Oriented Architecture) and enables to minimize human intervention during the process. Because it is impossible to deal with all kinds of real-world data, we focus on business data (i.e. customer order, order processing) which are frequently used in services such as CRM(Customer Relationship Management) and ERP(Enterprise Resource Planning). We first generate an extended-element vector by extending semantics of data exchanged between composite services and then build a rule-based system for setting data constraints automatically using the decision tree learning algorithm. We applied this rule-based system into the data cleansing service and showed the automation rate over 41% by learning data from multiple registered services in the field of business.

▸ Keyword : 서비스-지향 구조(Service-Oriented Architecture), 데이터 정제(Data Cleansing), 데이터 품질(Data Quality), 데이터 제약조건(Data Constraints), 의사결정트리(Decision Tree)

• 제1저자 : 이정원
• 투고일 : 2009. 1. 23, 심사일 : 2009. 2. 24, 게재확정일 : 2009. 3. 10.
* 아주대학교 정보통신대학 전자공학부 교수
※ 이 연구는 2008학년도 아주대학교 교내연구비 지원(20083770)에 의하여 이루어졌음.

Ⅰ. 서 론

서비스 지향 구조 (Service-Oriented Architecture, 이하 SOA)는 느슨하게 결합된 서로 협력하는 서비스들로 시스템을 개발하는 아키텍처링 방법을 제안하는 패러다임이다[1]. SOA는 기존의 컴포넌트 기반의 소프트웨어 개발 방법과는 달리 통신 프로토콜에 상관없이 서비스 작성(composition) 및 조립(assemble)을 용이하게 하여 기업 내부의 시스템은 물론 B2B 시스템에 이르기까지 엔터프라이즈 수준의 시스템 통합을 가능하게 한다는 점에서 각광 받고 있다. 특히 웹 서비스와 ESB(Enterprise Service Bus)와 같은 구현 기술에 힘입어 실시간 엔터프라이즈 (Real-Time Enterprise, RTE)를 실현하기 위한 비즈니스 분야에 급속도로 확산되고 있다[2].

웹서비스나 SOA 기반으로 서비스를 컴포지션할 때, 전체 시스템의 동작을 시뮬레이션 함으로써 서비스들 간의 연결 상태를 검증하는 것이 가능하다. 그러나 이때, 각 서비스 요구사항에 맞는 입출력만을 검증하는 것은 가능하나 의미적으로 옳은 데이터가 사용될 것인지에 대한 검증은 불가능하다. SOA의 목적이 서비스의 품질(Quality of Service, 이하 QoS)을 보장하면서 서비스들을 컴포지션하고 통합하며 관리하는 것이지만 실행시간에 교환되는 데이터의 품질을 보장하는 것은 여전히 개발자의 몫으로 남겨져 있다. 현재 대부분의 QoS는 서비스 자체에 대한 품질로서 성능, 유효성, 응답시간 등에 초점을 두고, 컴포지션시 최상의 성능을 낼 수 있는 서비스를 탐색하거나[3], 최소의 서비스 컴포지션으로도 컴퓨팅의 목적을 달성될 수 있는 척도 개발[4] 등에 초점을 두고 있다. 그러나 정작 서비스 컴포지션 후, 실제 교환되는 서비스간의 데이터에 대한 품질 보장에 대한 연구는 미비한 실정이다.

이러한 서비스간의 데이터 품질 보장을 위해, 이미 우리는 [5, 6] 연구에서 사용자가 명시한 데이터 제약조건에 따라 탐지 규칙이 설정되고 오류 데이터를 걸러낼 수 있는 방법을 제안하고 SOA기반의 데이터 정제 서비스를 개발한 바 있다. 따라서 어떠한 목적을 가진 서비스건 간에 두 서비스를 컴포지션할 때 데이터의 의미 변형을 가능하게 함으로써 상호작용하는 데이터의 제약조건 조정만으로 서비스의 재사용성과 컴포지션 성질을 극대화 시킬 수 있음을 보였다.

그러나 지금까지의 가장 큰 문제점은 사용자가 일일이 개입하여 데이터의 제약조건을 설정하도록 설계되어 있다는 데에 있다. SOA에서 서비스를 사용하는 개발자는 본인이 직접 개발해 놓은 서비스보다 이미 출판된(published) 그리고 개방된(open) 서비스를 사용하는 경우가 대부분이다. 따라서 데이터 제약조건 설정 자체도 데이터의 의미를 파악하고 이전에 사용된 데이터 혹은 이미 공개된 서비스의 데이터를 학습함으로써 제약조건 설정 시 사용자의 개입을 최소화 할 수 있어야 한다.

본 논문에서는 SOA를 기반으로 서비스 간에 상호 작용하는 데이터의 품질 관리를 위한 오류 정제 서비스를 대상으로 데이터 제약조건 설정 시 인간 개입을 최소화하기 위한 기법을 제안한다. 단, 실세계에서 통용되는 일반적인 데이터를 모두 다루는 것은 불가능하므로 비즈니스 도메인에서 자주 사용되는 CRM(Customer Relationship Management)과 ERP(Enterprise Resource Planning) 서비스와 같이 고객 주문 정보 및 처리에 관련된 데이터를 대상으로 한다. 이를 위해, 컴포지션 되는 서비스간의 상호 작용하는 데이터의 제약조건을 의미적으로 확장하여 확장-엘리먼트 벡터를 생성하고 이를 기반으로 의사 결정 트리(decision tree) 학습 방법을 적용하여 제약조건 설정을 위한 규칙 기반 시스템을 구축한다. 이 시스템을 오류 정제 서비스에 삽입한 결과, 학습을 통해 제약조건 설정을 41% 넘게 자동화 할 수 있음을 보였다.

Ⅱ. 관련 연구

데이터 품질 문제는 데이터가 시스템 사양에 맞지 않는 경우, 데이터의 복잡도 나 메타데이터의 결함으로 사용자가 데이터를 완전히 이해하지 못하는 경우, 혹은 사용자가 생각하는 데이터가 아닌 경우 등에서 발생한다[7]. 데이터 품질문제를 다루는 연구들은 데이터베이스의 한 테이블 내에 중복된 값들을 찾는 문제[8], 스키마 매핑 시 이름이나 구조상 충돌을 일으키는 데이터를 정제하는 문제[9] 등으로 주로 데이터베이스의 오류 데이터를 찾아내는데 초점을 맞추고 있다. 특히 데이터 마이닝 분야에서는 정제된 데이터가 필수적이기에 데이터 마이닝을 기초로 하는 ERP, CRM과 같은 다양한 어플리케이션에서는 데이터 정제 도구를 필요로 하고 있으며, 이러한 이유로 MonArch, SLAAM, ZipIt, The AscentialTM Enterprise Integration Suite, HummingBird와 같은 데이터 정제를 수행하는 다양한 도구들이 개발되었다[6]. 그러나 이미 데이터가 모두 수집된 데이터베이스 혹은 데이터 마이닝을 위한 저장소 내에서의 데이터 정제 기법만을 고려한다면 웹 서비스를 이용하는 SOA기반 시스템들에서 상호 작용하는 데이터의 품질을 고려할 수 있는

방법이 없다. 또한 ETL(Extraction, Transformation, Loading) 도구는 자체적으로 데이터 정제를 할 수 있는 내장 기능은 제공하고 있지 않지만, 개발자로 하여금 API를 사용하여 정제 기능을 명세화 할 수 있도록 하고 있다. 그러나 대부분 데이터 오류나 비일관성 등을 자동으로 검지할 수 있는 분석 기능을 제공하고 있지는 못한다[10]. 즉, 개발자의 노력 여하에 정적인 데이터 정제가 가능한 현실이다.

한편, SOA를 기반으로 서비스를 합성할 때, 두 서비스의 입력과 출력 데이터를 연결할 필요가 있다. 이러한 입출력 데이터 연결 방법에는 1대 1 매핑 방법과 여러 개의 데이터를 연결시켜서 하나의 데이터에 연결하거나 불필요한 데이터를 삭제하는 테일러링 방법이 있다. 이러한 연구는 컴포넌트 기반의 소프트웨어 개발 방법론에서, 컴포넌트의 I/O를 어떻게 합성할 것인가 하는 문제로 언급하고 있다[11]. 기존의 대부분의 SOA 도구들은 두 서비스간의 데이터 연결을 위해 데이터 매핑과 테일러링 할 수 있는 방법은 제공하고 있다. 주로 XML 을 이용하여 서비스 A의 출력 데이터 집합과 서비스 B의 입력 데이터 집합을 한눈에 보여 주고 선으로 연결할 수 있는 도구를 제공하고 있다.

그러나 불특정 다수를 위해 개발된 서비스에서 사용되는 데이터는 단순히 매핑이나 테일러링만으로 데이터를 형식적/의미적으로 연결하기 어렵다. 즉, 데이터가 동일한 이름과 타입을 가지더라도 실제 서비스가 사용될 때, 데이터의 제약조건이 수시로 변화 할 수 있기 때문이다. 예를 들어, CRM과 같은 시스템이 서비스로 등록되어 있다고 하자. 이 등록된 서비스를 SOA를 기반으로 재사용하고자 한다고 가정하면 직원 100명 이하의 기업과 직원 100,000명 이상의 대규모 기업에서는 CRM 자체의 기능은 그대로 사용할 지라도 데이터의 크기나 데이터가 처리해야 할 범위가 달라질 수 있다. 즉, 적용하고자 하는 비즈니스 프로세스의 규모에 따라 여러 가지로 데이터의 설정을 바꾸어 사용할 필요가 있다. 특히 시장 상황에 따라 오퍼레이션이 다 각도로 변화할 수 있는 B2B 시스템에 이용될 경우에는 오늘 사용 가능한 데이터가 내일은 사용될 수 없는 범위에 놓일 수 있다. 이렇게 다변화 하는 데이터 의미의 변형 문제를 해결하지 않는다면 항상 서비스의 내부를 건드려야 하고, 등록된 서비스는 재설계될 필요가 있다.

한편, 우리의 사전 연구인 [5, 6]은 데이터 정제의 범위를 실시간 서비스로 확장하여, SOA기반의 서비스 컴포지션시 부적절한 데이터를 탐지하고 이를 정제할 수 있는 기법을 제안하고 SOA 지원 도구인 Business Integration Suite[12]를 이용하여 서비스 오류 데이터 정제 서비스를 개발한 바 있다. 이 서비스는 실제 상품주문을 위해 9개의 서비스로 이루어진 전자상거래 시스템에 독립적인 서비스로 컴포지션되어 고객의 주문정보 중, 오류 데이터를 탐지하여 서비스가 사용되기 전보다 오류 데이터를 30% 이상 줄이는 효과를 보였다. 그러나 이 서비스에서 오류 데이터를 탐지하기 위한 규칙을 개발자가 일일이 설정해야 하는 비효율적인 면이 여전히 존재하였다.

III. 오류 정제 서비스

이 장에서는 본 논문의 데이터 학습을 설명하기 위해 사전 연구인 [5]의 오류 데이터의 정의 및, 탐지 규칙, 그리고 이를 구현한 서비스를 간략하게 소개한다.

서비스간의 상호 교환되는 데이터 제약조건을 위반하는 데이터를 오류로 보고 이를 탐지하여 걸러내는 작업은 두 가지 목적을 갖는다. 하나는 데이터의 품질을 보장하기 위한 것이고 다른 하나는 서비스의 컴포지션과 재상용성을 증강시키는데 있다. 즉, 서비스와는 독립적으로 데이터의 의미 변형, 예를 들어 데이터 값의 범위의 변화, 데이터 카테고리의 변화 등을 가능하게 하기 위한 것이다. 각각의 오류 타입에 따른 비즈니스 도메인의 문제들을 데이터 품질 보장 측면과 서비스 컴포지션과 재사용성을 보장하기 위한 목적으로 나누어 예시하면 다음 표 1과 같다. [5]에서 제시한 33가지의 오류 타입 중 비즈니스 데이터와 직접 관련 있는 6가지의 오류 타입 (1, 2, 3, 5, 11, 21번)만을 선정하였는데, 이는 서비스 개발자로부터 제약조건을 설정하여 탐지될 가능성이 있으며 서비스 재사용성을 높이는 데에 기여 할 수 있는 타입들이다.

표 1을 중심으로 탐지 알고리즘을 설계하고 서비스간의 입출력을 XML 기반으로 정의하여 다음 그림 1과 같은 SOA기반의 오류 정제 서비스를 구현하였다.

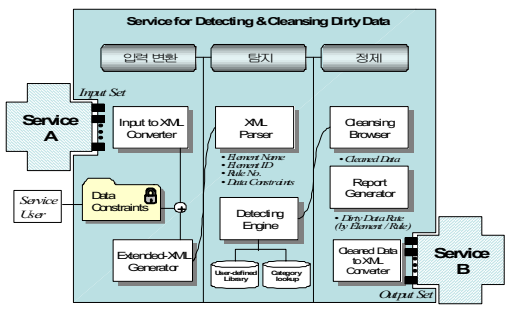


그림 1. 오류 정제 서비스
Fig. 1. Data Cleansing Service

표 1 비즈니스 문제 중심으로 바라본 데이터 오류
Table 1. Data Error according to the View of Business Problems

No.	Error Type	Business Analysis
1	오류 타입 및 제약조건	"missing data without the constraint of not allowing null" (Constraints) not empty and allowing null
	의미	Service A에서 B로 입력되는 데이터들 중 empty data는 missing data로 보고 오류 처리한다. 이때 'null' 입력은 허용한다.
	예	Service A에서 사용되던 "employee's supervisor" data가 선택적(optional)으로 사용되었다고 하자. 즉 이 data는 empty일 수도 있고 'John Smith'와 같은 값을 가져도 되는 데이터이다. 그러나 다른 개발자가 이 서비스를 재사용하려고 할 때, 이 데이터의 성질을 반드시 값이 존재하는 것으로 변경하려면 이 제약조건을 줌으로써 service A의 내부를 변경하지 않고 그대로 재사용할 수 있다. 즉 empty value이면 representative value인 null로 교체함으로써 새로운 개발자가 service A를 그대로 사용할 수 있게 된다.
2	오류 타입 및 제약조건	"missing data with the constraint of not allowing null" (Constraints) not empty and not allowing null
	의미	Service A에서 B로 입력되는 데이터들 중 empty data는 missing data로 보고 오류처리 한다. 이때 null을 허용하지 않는다.
	예	Service A에서 사용되던 "wedding anniversary"는 data value가 empty이더라도 운영되었다. 그러나 고객의 결혼기념일 정보를 매우 중요시 여기는 웨딩을 전문으로 하는 회사가 service A를 사용하고자 한다면 이 data는 empty도, null value도 허용하지 않도록 제약조건을 설정하고 service A를 재사용할 수 있다.
3	오류 타입 및 제약조건	"wrong data type" (Constraints) not violating data type constraint
	의미	"age" data는 200이라는 값을 가질 수 없는 것처럼 데이터 값의 범위(range)에 위배되는 데이터를 걸러 낸다.
	예	Service A에서 사용되던 "# of Order"는 데이터 타입이 정수이기만 하면 되었다. 그러나 다른 개발자가 각 분기별로 'order'의 양을 조절하고 싶다면 이 데이터의 값의 범위를, 주문 한 건당 최대 혹은 최소 수량을 설정하여 조정할 필요가 있다. 즉, 데이터 값의 범위를 조정함으로써 service A를 그대로 재사용할 수 있다.
5	오류 타입 및 제약조건	"duplicated data (violating non-null uniqueness constraint)" (Constraints) not duplicated and not allowing null
	의미	네트워크 장애나 사용자의 중복 주문 실수 등으로 인한 중복 데이터를 걸러낸다.
	예	가격 경쟁을 하는 경매 등과 같은 service의 고객 데이터 주문 정보는 중복을 허용하지 않을 수도 있다. 중복을 허용하지 않는 데이터 값이 필요한 경우, 이 제약조건을 줌으로써 service를 재사용할 수 있다.
11	오류 타입 및 제약조건	wrong categorical data (e.g., out of category range data) (Constraints) within category range
	의미	일반적으로 service의 데이터 타입은 integer, string, float 등 정도로만 입력할 수 있다. 이 외에도 데이터의 카테고리에 따라 새로운 타입을 정의 할 수 있으며 이에 위배 되는 데이터를 걸러 낼 수 있다.
	예	Service A에서 사용되는 "postal zip code"는 미국을 비롯한 전 세계의 모든 데이터를 다 받아들인다. 그러나 Canada의 시스템 개발자가 이 service를 Canada지역의 주문만 배송하려고 한다면 postal zip code의 처리 범위를 조정할 필요가 있다. 즉 zip code의 형식만을 변형함으로써 service를 재사용 가능하게 한다.
21	오류 타입 및 제약조건	"Ambiguous data because of using abbreviation" (Constraints) identifying the abbreviation
	의미	약어(Abbreviation) 혹은 복합어(compound word) 사용을 허가한 경우, 이것이 본래 어떤 의미로 사용된 것인지 파악하여 오류인지 아닌지 결정한다.
	예	예를 들어 "person's title"인 경우 abbreviation을 허가한다면 Dr.과 Doctor를 동일한 data로 처리할 수 있어야 한다. 또 다른 목적의 service 재사용 시 그 데이터의 abbreviation을 불허할 수도 있다.

오류 정제 서비스는 앞단의 서비스 출력을 받아 들여, 변환 과정, 오류 탐지 과정 및 정제 과정을 거쳐 뒷단의 서비스에 적절한 데이터를 걸러서 내보낸다. 이 때, 모든 데이터 입출력은 XML을 기반의 요구 명세에 데이터 제약조건과 결합되어 서비스간의 계약(service contract)을 만든다. 즉 서비스 A와 서비스 B사이에 정제 서비스가 담당해야 할 계약을 확장-XML이라고 하며 이 문서는 탐지 과정의 기초가 된다. 앞서 표 1에 제시된 6가지의 오류 타입에 따른 탐지 알고리즘에 의

해 데이터와 오류 원인이 함께 정제를 위한 브라우저에 제시되고 오류에 대한 통계를 보고서로 출력한다. 이 서비스는 정적인 데이터베이스를 검색하여 통계적인 방법으로 오류 데이터를 걸러내는 것이 아니라 제약조건에 위배되는 데이터를 실시간으로 탐지함으로써 서비스 사용자에게 즉각적인 피드백이 가능하다. 특히, 서비스 내부를 변경하지 않고 서비스 외부에서 제약 조건을 조정할 수 있는 방법을 개발함으로써 어떠한 목적을 가진 서비스이건 간에 두 서비스를 합성할 때 데이터의 의미 변형을 융통성 있게 도와 줄 수 있다. 즉, 상호작용하는 데이터 제약조건의 조정만으로 서비스의 재사용성과 컴포지션 가능성을 극대화시킬 수 있다.

지금까지의 가장 큰 문제점은 사용자가 일일이 개입하여 데이터 제약조건을 설정하도록 설계되어 있다는 데에 있다. 그러나 SOA에서 서비스를 사용하는 개발자는 본인이 직접 개발해 놓은 서비스보다 이미 출판된(published) 그리고 개방된(open) 서비스를 사용하는 경우가 대부분이다. 따라서 제약조건 설정자체도 데이터의 의미를 파악하고 이전에 사용된 데이터 혹은 이미 개방된 서비스의 데이터를 학습함으로써 제약조건 설정 시 사용자의 개입을 최소화 할 수 있어야 한다. 다음 장에서는 상호 작용 데이터의 제약조건을 학습함으로써 다음 서비스 사용 시 제약 조건을 완전히 새로 설정하지 않고 반자동으로 설정할 수 있는 방법을 설명한다.

IV. 데이터 제약 조건 학습

실세계에서 통용되는 데이터를 모두 다루는 것은 불가능하다. 본 논문에서는 비즈니스 영역에서 자주 사용되는 CRM과 ERP 시스템과 같이 고객 주문 정보와 주문 처리에 관련된 데이터로 제한을 둔다.

1. 데이터 의미 확장

데이터 설정을 자동화 할 수 있는 방법은 다음과 같다.

- 단순 저장: 현재 사용되는 서비스의 데이터와 제약 조건들을 모두 저장한다. 차후 동일한 이름의 데이터가 사용되면 저장된 제약 조건을 자동으로 설정한다.
- 의미 확장: 개방된 서비스는 다수의 개발자가 여러 가지 목적으로 개발하여 등록해 놓은 것이므로 반드시 하나의 용어가 한 가지 의미로 사용됐으리라는 보장이 없다. 즉 어떤 서비스 개발자는 'price'를 'cost'로, 또 'product'를 'OrderItems' 등과 같이 다양한 용어를 사용하여 동일한 의미를 표현하게 된다. 따라서 데이터를 유사어(synonyms)로 확장함으로써 유사 데이터에 대한 제약조건의 설정도 가능하다.

단순 저장 방식은 특별한 방법 없이 데이터 통계에 의존하면 되므로 학습의 의미가 없다. 따라서 본 논문에서는 의미 확장 방법을 제시한다. 데이터의 의미를 확장하기 위해 하나의 단어가 유사어를 가지도록 다음과 같은 과정을 거쳐 확장-엘리먼트 벡터(extended-element vectors)를 생성한다.

- 오류 정제 서비스의 제약 조건을 명시한 XML 기반의 서비스 계약 문서를 파싱하여 엘리먼트를 추출하고 DOM(Document Object Model) 트리를 생성한다.
- 공백, 하이픈, 밑줄 등의 구분자들을 걸러 내고 토큰만을 추려 낸 뒤, 불용어(stop-list)를 제거한다.
- 토큰의 어근(stem)을 추출한다. 어근으로 만들어진 엘리먼트를 WordNet시소러스와 사용자-정의 용어 사전을 이용하여 유사어, 복합어, 그리고 생략어 순으로 확장한다. 예를 들어 'quantity'는 (QT, amount)를, 'OrderID'는 (Order, Ordering, purchase, OID, ID)로 확장될 수 있다.

여기에서 확장-엘리먼트 벡터를 형식적으로 다음과 같이 정의한다.

(Definition 1.) 확장-엘리먼트 벡터

// SynVect_WN: synonyms, abbreviation, compound words that are found from WordNet

// MaxSize : maximum length of an extended-element vector

// 〈T〉 is an element, and t is a term in an extended-element vector

$$DataSet_i = (\langle T^i_1\rangle, \langle T^i_2\rangle, \cdots, \langle T^i_m\rangle),$$
$$T^i_p = (t^i_{p1}, t^i_{p2}, \ldots, t^i_{pa})$$

when a ≤ MaxSize and p: random number,

$t^i_{ma} \in$ SynVect_UDL or SynVect_WN

즉, I/O 명세로부터 입력되는 각각의 데이터 엘리먼트는 〈Tin〉으로 정의되고 또 그 〈Tin〉은 다시 Tin→(synonym1, synonym2, …) 로 확장되는 것이다. 예를 들어 어떤 서비스의 입력이 DataSet = (〈Order〉, 〈Price〉, 〈Items〉)이었다면, Order=(OrderID, Ordering), Price = (Pricing, Cost), Items = (OrderItems, Product, ProtuctItems)로 예시할 수 있다.

2. 의사결정 트리를 이용한 제약 조건 학습

이제 확장-엘리먼트 벡터를 기반으로 각각의 데이터에 대한 제약조건을 학습할 재료가 준비가 되어 있다. 본 논문에서는 본래의 용어에 대한 제약 조건만을 기억하는 단순 암기 학습이 아닌 벡터안에 있는 유사어와의 관계를 추론해서 가장 적절한 규칙을 설정해 줄 수 있도록 의사결정 트리 학습 알고리즘을 적용하였다. 결정 트리는 교사 학습(supervised learning) 방법 중 하나로 이산 값을 가지는 타겟 함수를 근사하는 널리 사용되는 귀납(inductive) 추론 방법이다 [13]. 교사 학습 방법에는 SVM(Support Vector Machine), NN(Neural Network), Naive Bayes, KNN(K-Nearest Neighbor) 등의 대표적인 알고리즘이 있다 [14]. SVM은 입력을 n-차원 공간에 두 개의 벡터 집합으로 보고 두 집합의 간격(margin)을 거리화 하여 최대화 하는 방법이다. NN은 비선형 확률 모델로 단계별로 변화하는 적응형 시스템(adaptive system)에 적합한 알고리즘이다[15]. Naive Bayes 분류 방법은 대상의 속성이 매우 다양한 경우, Bayes의 정리를 이용한 확률 모델이다. 반면, 결정 트리는 속성-값으로 쌍이 되는 사례에 가장 적합하며[16] 우리의 문제도 데이터-제약조건의 쌍으로 표현할 수 있기 때문에 적합한 학습방법이다. 특히 루트 노드로부터 단말 노드에 이르는 트리의 정렬에 의해 인스턴스가 속하는 클래스를 분류할 수 있기 때문에, 본 논문에서 제약 조건 학습 후, 규칙 기반 시스템으로 학습 알고리즘을 구현하기에 매우 적합하다. 다음 그림 2는 데이터 제약 조건을 학습하기 위해 설계한 결정 트리이다.

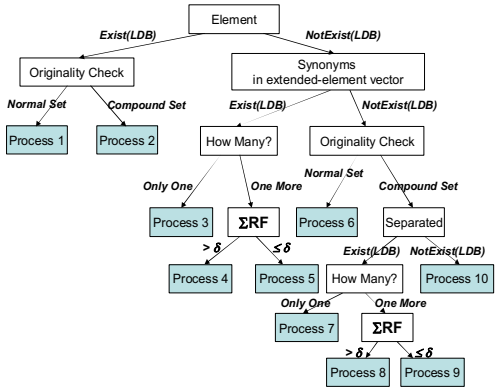


그림 2. 제약조건 학습을 위한 결정 트리
Fig. 2. Decision Tree for Learning Data Constraints

트리의 가지(branch)는 인스턴스의 값의 테스트를 의미하고 노드는 그 테스트 속성을 만족하는 클래스를 가리킨다. 최종 단말 노드는 분류된 10개의 프로세스(Process 1~10)로 각각의 프로세스는 루트 노드로부터 단말 노드에 이르기까지의 정보를 정합하여 학습을 위한 규칙을 정의 할 수 있다. 각 프로세스별 자세한 처리 규칙은 다음 3절에서 명제 논리(propositional logic)로 설명한다. 트리의 가지에 사용된 속성들은 크게 3가지 분류로 첫째, 학습된 데이터인가 혹은 아닌가에 따라 분류되고 두 번째는 일치하는 유사어가 하나뿐인가 아니면 다수인가에 따라서 분류되며, 마지막으로 주어진 값이 일정 임계치를 넘는가아닌가에 따라 분류된다. 이러한 테스트 과정은 단말 노드에 도달할 때까지 반복되며 사용된 속성을 자세하게 살펴보면 다음 표 2와 같다.

표 2. 결정 트리의 노드와 가지 속성의 의미
Table 2. The Meanings of Internal Nodes and Branches

Node	Attributes & Meaning
Element	Exist(LDB): 엘리먼트가 이미 학습된 적이 있을 때. (학습 데이터베이스인 LDB를 참조)
	NotExist(LDB): 엘리먼트가 한 번도 학습된 적이 없어서 LDB에 존재하지 않음.
Originality Check	Normal Set: 엘리먼트는 복합어도 생략어도 아님.
	Compound Set: 엘리먼트가 복합어 혹은 생략어임.
Synonyms in the extended element vector	Exist(LDB): 확장-엘리먼트 벡터의 유사어들이 학습된 적이 있어서, 유사어의 학습 정보를 LDB에서 찾을 수 있음.
	NotExist(LDB): 확장-엘리먼트 벡터의 모든 유사어들이 학습된 정보가 LDB에 없음.
How Many?	Only One: 하나의 유사어의 학습 정보만이 LDB에 있음.
	One More: 하나 이상의 유사어의 학습 정보가 LDB에 있음.
Separated	Exist(LDB): 복합어로부터 파생된 용어가 학습된 적이 있음.
	NotExist(LDB): 복합어의 파생어도 학습된 정보가 없음.
RF	$>\delta$: 정의 2의 RF의 합이 일정 임계치(δ)보다 클때. 이는 본래의 엘리먼트가 LDB에 있지 않을 때, 임계치를 넘는 유사어로 본래의 엘리먼트의 학습정보를 대치하고자 하는 목적으로 사용.
	$\leq\delta$: RF가 δ 를 넘지 못할 때. 이는 본래의 엘리먼트를 대치할 만한 유사어의 학습정보가 없음을 의미.

여기에서 exist(LDB) 속성은 상위 노드가 본래의 엘리먼

트이건 아니면 엘리먼트의 파생된 벡터 안의 유사어이건 이미 학습된 정보를 가지고 있는 지를 검사하는 속성이다. 이때 RF(Rule Frequency)라 하여 학습된 엘리먼트의 규칙 빈도수를 정의하여 만약 하나의 엘리먼트 혹은 유사어가 학습된 정보를 다수(multiple) 가지는 경우, 각각의 RF를 검사하여 가장 빈도수가 큰 학습 정보를 이용하게 하였다. 다음은 엘리먼트 및 이를 포함한 벡터의 RF를 정의한 것이다.

(Definition 2.) 확장-엘리먼트 벡터의 RF
// RF는 규칙-빈도수, 총 6개의 규칙별 계수기 포함

$$RF(E) = (rf^i_{p1}, rf^i_{p2}, \cdots, rf^i_{p6})$$

이때, $E = T^i_p$ (original element) or t^i_{pn} (one of synonyms)

여기에서 고려해야 할 사항은 동일한 데이터라 하더라고 조립되는 시스템의 목적에 따라 데이터의 제약 조건은 달리 설정될 수 있다는 것이다. 예를 들어 'OrderID' 에 대해 규칙 2(not null), 3(wrong data type), 5(uniqueness)를 모두 적용하기도 하고 때로는 값의 범위를 주지 않고 규칙 2와 5만 적용할 수도 있다. 즉, 각 엘리먼트 T는 rule 1, 2, 3, 5, 11, 21에 대한 RF를 누적하여 저장한 6개의 계수기를 가지게 된다. 예를 들어 'OderID'의 경우, RF('OrderID') = (0, 20, 18, 20, 0, 0)을 가질 수 있다. 이때 각 데이터의 규칙 별 rf 값이 높을수록 가장 높은 설정 우선순위를 갖는다. 따라서 'OrderID'는 rule 2, rule 5, 그리고 rule 3순으로 설정할 수 있는 근거를 가진다. 한편, 특정 엘리먼트가 학습된 적은 없지만 유사어중 다수가 학습된 것으로 판단될 때, RF를 가지고 우선순위를 결정하게 된다. 즉 두 단어 중 학습 빈도가 더 높았던 것을 비즈니스 영역에서 더 자주 사용되어 학습된 것으로 보고 엘리먼트의 규칙을 추천받기 위해 RF가 높은 단어의 rule 정보를 이용하게 된다.

다음 그림 3은 하나의 데이터에 대한 확장-엘리먼트 벡터와 각각의 RF를 저장하고 있는 데이터 사례(OrderID)의 테이블의 자료 구조로 앞에서 언급한 LDB의 자료 구조에 해당한다. 그림 3에서 보듯이, 'OrderID'는 LDB를 보면 학습된 적이 없으며(N/A) synonyms로 (Order, Ordering, Purchase, OID, ID)를 갖는다. 이때, Normal Set은 WordNet을 통해 유추 가능한 일반적인 유사어들이고 Compound Set은 단어의 합성 원리로 쪼갰을 때, 쪼개진 단어를 통해 유추된 유사어를 의미한다. 'Sim' 필드는 유사성(similarity) 값으로 normal set의 'Sim'은 온전히 1이고 compound set의 'Sim'값이 0.5로 두었다. 이는 normal set의 매칭에 더 우선순위를 두기 위한 것이나 변경 가능하다. 'Tno.' 는 RF 테이블 인덱스이며 인덱스 10, 11, 98, 104, 105를 탐색하면 각각의 규칙 번호(1, 2, 3, 5, 11, 21)에 따른 RF값 및 부가적인 제약 조건 정보들이 담겨져 있다. 여기에서 RF(Order) = 48, RF(Ordering) = 8, RF(ID) = 360으로 한 번도 학습된 적이 없는 OrderID에 대한 규칙 설정을 위해서 ID → Order → Ordering 순으로 추천된다.

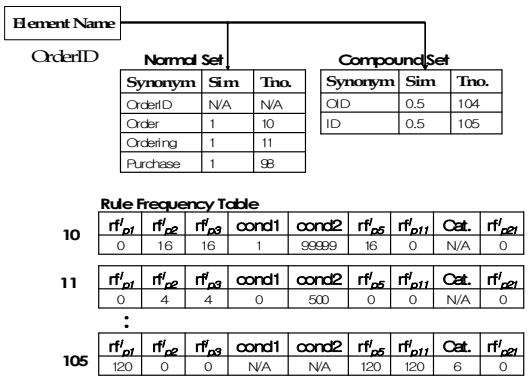


그림 3. LDB의 학습된 데이터의 RF 구조
Fig. 3. RF Structure of Data Learned in LDB

3. 제약 조건 규칙 생성

이제 앞 절에서 결정 트리의 단말 노드인 프로세스 1~10까지를 규칙으로 모두 정의한다. 트리의 루트로부터 단말까지의 노드와 속성을 따라 조합한 10개의 프로세스는 크게 5가지 종류의 rule로 구분되며 데이터의 포함관계를 명확히 할 수 있는 명제 논리로 정의한다. 명제 논리로 정의된 규칙은 구현 시, 규칙 기반 시스템의 지식 베이스로 자연스럽게 표현이 가능하다.

- LearnedElement Rule: 엘리먼트가 학습된 적이 있을 때. (for Process 1)

$$\forall x, (Element(x) \wedge ExistLDB(x) \wedge OriginalityCheckNormal(x)) \supset LearnedElement(x)$$

입력으로 들어 온 엘리먼트가 compound word가 아니면서 완전 일치하는 학습 정보가 LDB에 있을 때, 그 RF를 자동으로 설정하게 된다. 예를 들어 엘리먼트 집합에 {CustomerID, Email, Order, Price, BillAmount}가 있고 각각의 엘리먼트에 대하여 LDB를 검색한다. ExistLDB(x) 결과, {Email, Price, BillAmount}를 검색했

고 compound word인지 아닌지를 검사해서 최종적으로 {Email, Price}를 얻으면 이 두 엘리먼트는 이미 학습된 정보가 있으므로 학습된 규칙으로 설정된다.

- LearnedDerivedElement Rule: 엘리먼트가 복합어로서 학습된 적이 있을 때. (for Process 2)

$$\forall\ x,\ (\ Element(x)\ \wedge\ ExistLDB(x)\ \wedge\ \neg\ OriginalityCheckNormal(x)\ )\ \supset\ LearnedDerivedElement(x)$$

엘리먼트가 compound word이면서 완전 일치되는 학습정보가 LDB에 있을 때, 그 RF를 자동으로 설정하게 된다. 위에서 {BillAmount} 가 이에 해당한다.

- LearnedSynonym Rule: 엘리먼트 자체의 학습정보는 없으나 벡터의 유사어가 학습된 적이 있을 때. (for Process 3, 4, 5)

$$\forall\ x,\ y,\ (\ Element(x)\ \wedge\ \neg\ ExistLDB(x)\ \wedge\ Synonyms(x,\ y)\ \wedge\ ExistLDB(y))\ \supset\ LearnedSynonym(x,\ y)$$

엘리먼트가 완전 일치되는 학습정보가 LDB에 없는 경우다. 이때, 확장-엘리먼트 벡터로 확장하고 벡터의 유사어 리스트에 있는 엘리먼트들을 검색한다. 예를 들어 위의 경우 LDB에 없는 {CustomerID, Order}를 CustomerID→{customer, customer number}와 Order→{order of magnitude, ordering, orderliness, order items}로 확장하고 CustomerID의 모든 유사어들과 Order의 모든 유사어들을 LDB에서 검색한다. 그리고 하나 이상의 학습된 유사어 정보가 발견되면 6개의 계수기 중 가장 높은 혹은 일정 임계치를 넘는 RF를 가지는 유사어가 가지는 제약 조건이 선택된다.

- LearnedDrivedSynonym Rule: 엘리먼트의 확장 벡터에 있는 유사어들로부터 파생된 엘리먼트들만 학습된 적이 있을 때. (for Process 7, 8, 9)

$$\forall x,\ y,\ z,\ (\ Element(x)\ \wedge\ \neg\ ExistLDB(x)\ \wedge\ Synonyms(x,\ y)\ \wedge\ \neg\ ExistLDB(y)\ \wedge\ \neg\ OriginalityCheckNormal(y)\ \wedge\ DerivedType(y,\ z)\ \wedge\ ExistLDB(z)\ )\ \supset\ LearnedDerivedSynonym(x,\ z)$$

입력 엘리먼트도, 그 엘리먼트의 유사어도 모두 학습된 적이 없을 때, 만약 그 엘리먼트가 compound word이면 혹시 그 엘리먼트를 쪼개어 각각의 단어가 LDB에 있는지 검색한다. 이때 검색 결과가 다수 있다면 우선순위별 혹은 일정 임계치 이상의 RF를 가지는 파생된 엘리먼트가 선택된다. 예를 들어 CustomerID{customer, ID}로 쪼개지고 LDB, 검색 결과 ID에 대한 학습 정보가 있다면 ID의 RF가 선택된다. 이 규칙은 조금 위험할 수도 있다. 즉 학습된 ID는 Customer ID일 수도 있고, 물품 ID일수도 있기 때문이다. 그러나 추천받은 데이터 제약조건을 완전 자동으로 설정하는 것이 아니라 서비스 사용자가 최종적으로 재검토하게 되므로 이러한 경우를 배제 시킬 수 있다.

- NotLearnedElement Rule: 엘리먼트 및 파생어들이 전혀 학습된 적이 없을 때 (for Process 6, 10)

$$\forall x,\ y,\ z,\ ((Element(x)\ \wedge\ \neg\ ExistLDB(x)\ \wedge\ Synonyms(x,\ y)\ \wedge\ \neg\ ExistLDB(y)\ \wedge\ \neg\ OriginalityCheckNormal(y))\ \vee\ (Element(x)\ \wedge\ \neg\ ExistLDB(x)\ \wedge\ Synonyms(x,\ y)\ \wedge\ \neg\ ExistLDB(y)\ \wedge\ \neg\ OriginalityCheckNormal(y)\ \wedge\ DerivedType(y,\ z)\ \wedge\ \neg\ ExistLDB(z)))\ \supset\ NotLearnedElement(x)$$

엘리먼트의 유사어, 유도 타입 모두 학습된 적이 없으므로 규칙 설정을 자동화 할 수 없다.

지금까지 서비스간에 상호 작용하는 데이터의 오류를 탐지하고 제약 조건을 학습함으로써 자동으로 오류 데이터를 거를 수 있는 규칙을 설정할 수 있는 방법을 제안하였다. 다음 장에서는 제안한 방법론을 SOA를 기반으로 독립된 오류 정제 서비스에 적용한 사례를 제시한다.

V. 구현 및 실험 결과

이미 사전 연구[6]에서 Java 2 Platform, Enterprise Edition 1.4.2로 구현된 오류 정제 서비스를 구현한 바 있다. 또한 이 서비스를 Fiorano Business Integration Suite[12]라는 SOA 모델링 및 실행 지원 도구를 기반으로 여러 가지 서비스와 합성하여 오류 정제의 개선율이 증대되는 것을 보였다. 그러나 합성시 서비스 간에 상호작용하는 데이터의 제약조건을 일일이 설정하게 함으로써 사용자에게 불편을 줄 뿐만 아니라 데이터의 의미를 파악해야만 오류 정제 서비스를 사용할 수 있었다. 따라서 본 논문에서 제시한 비즈니스 서비스 데이터 학습에 의한 데이터 제약조건 자동 설정 방법을 JESS[17]로 구현하여 오류 정제 서비스에 추가하였다. 그리고 새로운 오류 정제 서비스를 다음과 같은 동일한 서비스 합성에 적용해 보았다. 다음 그림 4의 시스템 목적은 고객의 주문 정보가 정확하지 않거나 ERP의 재고 관리 정보에 의해 주문이 적절하게 처리 되지 않을 가능성이 있는 경우, 자동으로 필터링 해주고 이 사실을 고객에게 알려 주기 위한 것이다. 이 CRM과 ERP사이에 상호 교환되는 데이터의 제약조건을 자동으로 설정해주고 이 조건을 기반으로 오류 탐지 및 정제가 가능해 진다.

그림 5는 CRM으로부터 'Data Elements' 필드에

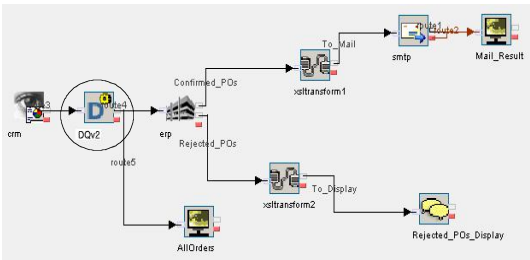


그림 4. CRM과 ERP 서비스의 합성
Fig. 4. Composition of CRM and ERP

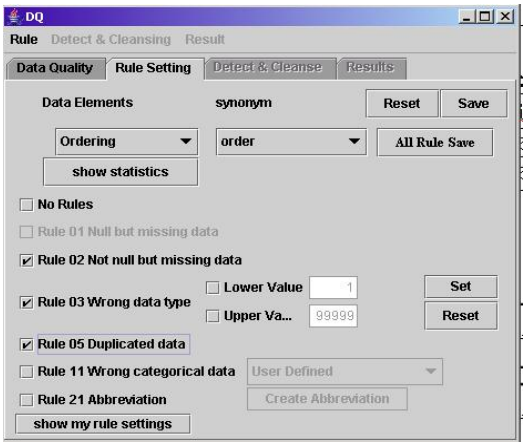


그림 5. 규칙 자동 설정 예 ('ordering')
Fig. 5. An Example of Rule Setting ('ordering')

'Ordering'이라는 데이터가 ERP로 입력되어야 할 때, 과거에 'ordering'과 유사한 데이터(synonym 필드)로서 'order'라는 추천된 제약조건을 수정하고 싶다면 화면의 다른 checkbox 메뉴를 설정함으로써 조정 가능하다. 만약 새로운 규칙을 설정 했다면 'All Rule Save'를 선택하여 LDB의 학습 정보를 갱신하게 된다. 또한 'show statistics'를 선택하여 현재습된 규칙(Rule 02, Rule 03, Rule 05)을 자동으로 설정해 놓은 화면을 보여 주고 있다. 물론 서비스 사용자가 추천된 제약조건을 수정하고 싶다면 화면의 다른 checkbox 메뉴를 설정함으로써 조정 가능하다. 만약 새로운 규칙을 설정 했다면 'All Rule Save'를 선택하여 LDB의 학습 정보를 갱신하게 된다. 또한 'show statistics'를 선택하여 현재 'ordering'의 모든 학습정보를 열람할 수도 있다. 이러한 구현을 바탕으로 기존에 21개의 공개 서비스들을 본 서비스에 연결하여 입력 데이터 집합이 지식 베이스인 LDB를 얼마나 많이 그리고 정확하게 학습시킬 수 있는가를 실험 하였다. 실험은 다음 두 가지 파라미터 측정을 목표로 한다.

- 학습율(Learning Rate): 서비스의 데이터가 입력되면 본 논문에서 제시한 학습 방법으로 데이터를 학습하여 LDB에 저장한다. 이때 학습으로 인한 데이터 수 증가율은 LDB의 학습 데이터 수/순수 입력 데이터 수로 평가한다. 현재 훈련 데이터의 수가 매우 적으나 이 비율의 증가율을 통하여 이 서비스의 실질적인 사용 가능성을 제시할 수 있다.
- 자동화율(Automation Rate): 서비스의 데이터가 입력될 때, 학습된 데이터 제약 조건으로 인해 자동으로 규칙 설정이 가능해 진 경우를 센다. 이때 자동화율은 자동으로 규칙이 설정된 데이터의 수/입력 데이터 수로 평가한다.

1	Email	1	(To,…,SmtpServerUrl?)	
2	To	2	#PCDATA	2, 11
3	From	2	#PCDATA	2, 11
4	CC	2	#PCDATA	1, 11
5	BCC	2	#PCDATA	1, 11
6	ReplyTo	2	#PCDATA	1, 11
7	Subject	2	#PCDATA	1
8	Body	2	#PCDATA	2
9	Attachment	2	#PCDATA	1, 11
10	SmtpServerUrl	2	#PCDATA	2, 11
11	SMTP	1	(Result, Reason?)	
12	Result	2	#PCDATA	2
13	Reason	2	#PCDATA	1
14	Email	1	(To,…,Body)	
15	To	2	#PCDATA	2, 11
16	From	2	#PCDATA	2, 11

31	Order	1	(Item+,TotalPrice)	
32	TotalPrice	2	#PCDATA	2, 3
33	Item	2	(OrderID,…,OrderDate)	
34	OrderID	3	#PCDATA	2, 3, 5
35	CustomerID	3	#PCDATA	2, 3
36	Email	3	#PCDATA	2, 11
37	ShipDetails	3	(Mode,…,PhoneNumber)	
38	Product	3	#PCDATA	2, 11, 21
39	Quantity	3	#PCDATA	2, 3
40	Price	3	#PCDATA	2, 3
41	BillAmount	3	#PCDATA	2, 3
42	OrderDate	3	#PCDATA	2, 11
43	Mode	4	#PCDATA	2, 11, 21
44	Address	4	#PCDATA	2
45	Country	4	#PCDATA	2, 11, 21
46	PostalCode	4	#PCDATA	2, 11
47	PhoneNumber	4	#PCDATA	2, 11
48	ChatMessage	1	(Sender?,Message)	
49	Sender	2	(Name?,Email?)	
50	Name	3	#PCDATA	2, 21
51	Email	3	#PCDATA	2, 11
52	Message	2	#PCDATA	2

85	Mail-List	1	(Mail)+	
86	Mail	2	(Body,Attachment+)	
87	Body	3	#PCDATA	2
88	Attachment	3	#PCDATA	2, 11
94	Order	1	(Manufacture+)	
95	Manufacturer	2	(Product+)	
96	Product	3	(Name,Cost,Discount)	
97	Cost	4	#PCDATA	2, 3
98	Discount	4		1, 3
99	Order	1	(Product+)	
100	Product	2	(Manufacture,DiscountPrice)	
101	Manufacturer	3	#PCDATA	2, 21
102	DiscountPrice	3	#PCDATA	1, 3
103	InputPO	1	(POHeader,Address+,Item,To	
104	POHeader	2	#PCDATA	1
105	Address	2	(ContactType,…,Country)	
106	Item	2	#PCDATA	2, 11, 21
107	Total	2	#PCDATA	2

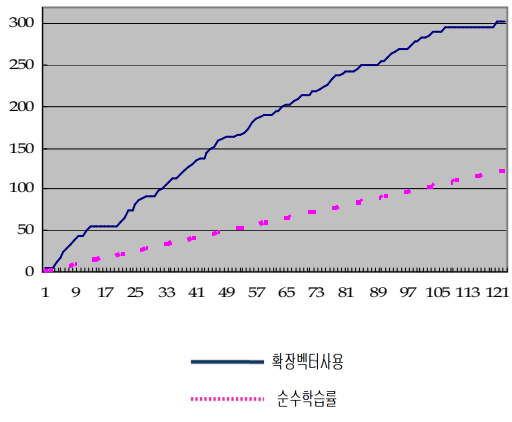
그림 6. 학습률/자동화율 평가를 위한 훈련 데이터 집합의 예
Fig. 6. Part of Training Data Used for Evaluation of Learning/Automation Rates

먼저 학습을 위해 Fiorano 도구에 이미 등록된 실제 서비스로부터 추출한 123개의 데이터를 훈련 데이터로 삼았으며 CRM, ERP와 같은 큰 규모의 서비스에서부터 chatting이나 SMTP Bridge 서비스 등 간단한 서비스에 이르는 총 21개의 서비스로부터 추출하였다. 위의 그림 6은 사용된 데이터의 일부를 보여준다 (데이터 이름, 레벨, 형식, 설정된 규칙순).

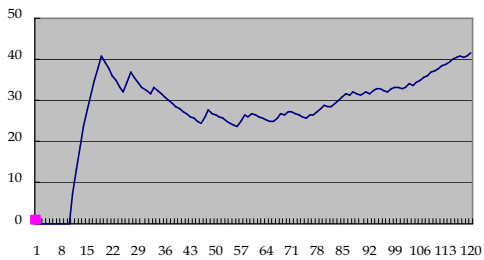
여기에서 'email' 혹은 'mail' 데이터는 1, 14, 36, 51, 66에 걸쳐 분포되어 있고 'Price (40)'와 'Cost(97)'와 같이 유사 단어도 보인다. 이러한 데이터를 학습한 결과 다음 그림 7과 같은 결과를 보였다.

그림 7(a) 그래프의 X축은 데이터의 일련 번호, Y축은 학습된 데이터의 수, 점선은 입력 데이터의 순수 학습율이며 실선은 본 논문에서 제시한 확장 벡터를 사용한 학습 증가율이다. 입력된 순수 데이터의 수가 123개인데 반해 학습으로 인해 증가된 LDB의 데이터 수가 303개로 증가하였다. 이는 한 데이터에 대해 평균 2.46개의 데이터를 의미 확장을 통해 더 학습시킨 것으로 볼 수 있다. 즉 email은 (electronic mail, e-mail, mail)과 같은 데이터를 더 학습시키고 다른 서비스에서 유사 데이터가 입력되었을 때, email은 물론이고 e-mail과 같은 데이터에 대해서도 동일한 데이터 제약 조건으로 규칙 설정을 자동화 할 수 있는 것이다.

그림 7(b)는 자동화율이며 평균 41.46%를 얻었다. 초반에 이 비율이 0인 의미는 입력이 학습된 데이터가 없는 초기 상태이며, 20까지 급격하게 상승한 것은 유사한 입력 데이터들이 들어 왔기 때문이다. 그러다가 123개의 데이터에 대해서 거의 40%에 포화된(saturated) 결과를 보였다. 실험 데이터가 임의로 생성한 것이 아닌 이미 SOA를 지원하는 도구에 등록되어 있는 비즈니스 영역에서 사용되는 서비스를 사용하였기 때문에 많은 수의 데이터를 학습할 수는 없었다. 그러나 40%가 넘는 자동화 율을 통해 데이터 제약 조건 설정을 자동화 할 수 있다는 것을 입증하였다. 만약 더 많은 실제적인 비즈니스 데이터를 획득한다면, 자동화율은 더 높은 결과를 보일 것이다.



(a) 학습률



(b) 자동화율

그림 7. 학습률/자동화율

Fig. 7. Learning Rate and Automation Rate

VI. 결론 및 향후 과제

본 논문에서는 SOA를 기반으로 서비스 간에 상호 작용하는 데이터의 품질 관리를 위한 오류 정제 서비스를 대상으로 데이터 제약조건 설정 시 인간 개입을 최소화하기 위한 학습 기법을 제안하였다. 이 기법을 오류 정제 서비스에 삽입한 결과, 학습을 통해 제약조건 설정을 41% 넘게 자동화할 수 있음을 보였다. 향후, 훈련 데이터의 수를 확대하여 자동화율을 높여야 할것이다. 또한 오류 정제 서비스의 탐지 규칙을 확장하여 오류 탐지율을 높이는 연구가 필요하다.

참고문헌

〔1〕 V. Kapoor, "Services and Automatic Computing: A Practical Approach for Designing Manageability," In Proceedings of the 2005 IEEE International Conference on Service Computing (SCC'05), Vol. 2, pp.41-48, July 2005.

〔2〕 T. Erl, "*Service-Oriented Architecture: Concepts, Technology, and Design*" Prentice Hall, 2005.

〔3〕 S. Choi, J. Her, and S. Kim, "QoS Metrics for Evaluating Services from the Perspective of Service Providers," In Proceedings of the IEEE International Conference on e-Business Engineering(ICEBE2007), pp.622-625, Oct. 2007.

〔4〕 S. Kalasapur, M. Kumar, and B. Shirazi, "Evaluating Service Oriented Architectures (SOA) in Pervasive Computing," In Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications (PERCOM'06), pp.276-285, Mar. 2006.

〔5〕 J.W.Lee, E.Y.Moon, and B.J.Choi, "Data Cleansing for Service-Oriented Architecture," LNCS, Vol.3590, pp.87-97. 2005.8.

〔6〕 지은미, 이정원, 최병주, "SOA 기반 서비스 사이의 오류 데이터 정제 서비스 개발," 정보처리학회 논문지D, 제14권 7호, 829 ~840쪽, 2007년 12월.

〔7〕 Theodore Johnson, and Tamraparni Dasu, "Data Quality and Data Cleaning," Tutorials of 10th SIGKDD, Aug. pp.181~191, 2004.

〔8〕 M. Hernandez and S. Stolfo, "Real-World Data is Dirty: Data Cleansing and The Merge/Purge Problem," Data Mining and Knowledge Discovery, Vol.2(1), pp.9-37, 1998.

〔9〕 M. Hernandez, R. Miller, and L. Hass, "Schema Mappings as Query Discovery," In Proceedings of Intl. Conf. VLDB, pp.77-89, 2000.

〔10〕 Erhard Rahm, Hon Hai Do, "Data Cleaning: Problems and Current Approaches," IEEE Data Engineering Bulletin, Vol. 23(4), pp.3-13, 2000.

〔11〕 S.S.Yau, and F. Karim, "Component Customization for Object-Oriented Distributed Real-Time Software Development," in Proceedings of the 2000 IEEE international Symposium on Object-Oriented Real-Time Distributed Computing(ISORC '00), pp.156-163, Mar. 2000.

〔12〕 Fiorano's Prebuilt Service Guide, *http://www.fiorano.com/downloads/fesb/prebuiltservices.pdf* 2006.

〔13〕 T. Mitchell, "*Decision Tree Learning*," in T. Mitchell, Machine Learning, The McGraw-Hill Companies, Inc., pp. 52-78, 1997.

〔14〕 Rich Caruana, and Alexandru Niculescu-Mizil, "*An Empirical Comparison of Supervised Learning Algorithms*" In Proc. of the 23rd International Conference on Machine Learning, pp. 161-168, 2006.

〔15〕 김천식, 홍유식, "텍스트마이닝을 이용한 XML 문서 분류 기술," 한국컴퓨터정보학회 논문지, 제 11권 제 2호, 15-23쪽, 2006년 5월.

〔16〕 노창현, 조규철, 미용범, 이종식, "의사결정트리 기법을 이용한 그리드 자원선택 시스템," 한국컴퓨터정보학회 논문지, 제 13권 제 1호, 1-10쪽, 2008년 1월.

〔17〕 JESS, *http://herzberg.ca.sandia.gov/jess* pp. 1266-1276, Nov. 2000.

저 자 소 개

이 정 원

2003년 이화여자대학교 컴퓨터공학 (박사)

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

2006년~현재 : 아주대학교 전자공학부 조교수

관심분야 : SOA, 유비쿼터스 컴퓨팅, 임베디드 소프트웨어