SOA 기반 서비스 사이의 오류 데이터 정제 서비스 개발

지 은 미[†] · 최 병 주[††] · 이 정 원[†††]

요 약

현재까지 오류 데이터를 정제하는 기법은 여러 소스로부터 대량의 데이터를 통합하여 데이터베이스에 저장되어 있는 데이터의 품질을 관리함으로써 의미 있는 정보를 추출하기 위함이었다. 급변하는 비즈니스 환경과 무한경쟁 사회에서 지속적으로 생존하려면 환경 변화에 빠르게 대처해야 한다. 최근 시스템의 요구사항이 복잡해짐에 따라 대규모의 분산 시스템을 통합·구축하기 위한 서비스 기반 구조 (Service Oriented Architecture)로 확산되고 있으며, 여기에서도 각 서비스간의 데이터 정제기법을 통한 신뢰성 있는 데이터 교환이 필수적이다. 따라서 본 논문에서는 서비스들이 하나의 시스템으로 통합되는 과정에서 이벤트를 통해 서비스 간에 전송되는 XML데이터의 품질 관리를 수행하여, 이미 통합되어 저장된 데이터베이스 데이터의 오류를 탐지하여 정제하는 것이 아니라 상호 작용하는 서비스간의 데이터 정제에 초점을 두고 SOA를 기반으로 하는 오류 데이터 정제 서비스를 개발한다.

키워드 : 데이터 품질, 데이터 정제, 서비스 지향 구조

Developing dirty data cleansing service between SOA-based services

Eun Mi Ji[†] · ByoungJu Choi[††] · Jung-Won Lee[†††]

ABSTRACT

Dirty Data Cleansing technique so far have aimed to integrate large amount of data from various sources and manage data quality resided in DB so that it enables to extract meaningful information. Prompt response to varying environment is required in order to persistently survive in rapidly changing business environment and the age of limitless competition. As system requirement is recently getting complexed, Service Oriented Architecture is proliferated for the purpose of integration and implementation of massive distributed system. Therefore, SOA necessarily needs Data Exchange among services through Data Cleansing Technique. In this paper, we executed quality management of XML data which is transmitted through events between services while they are integrated as a sole system. As a result, we developed Dirty Data Cleansing Service based on SOA as focusing on data cleansing between interactive services rather than cleansing based on detection of data error in DB already integrated.

Key Words : Data Quality, Data Cleansing, SOA : Service Oriented Architecture

1. 서 론

정보 기술 환경이 발전하면서 시스템의 요구 사항이 복잡해지고, 시스템 통합 요구의 증가로 독립성 및 유연성이 높아 이기종 시스템 간 통합 구축에 적합한 서비스 지향 구조[1](SOA : Service Oriented Architecture)로 변화하고 있다.

SOA는 서비스 비즈니스 프로세스를 중심으로 하여 네트워크상에서 서로 통신을 하는 일련의 서비스들이 느슨한 결합(loosely-coupled)으로 연결되어 있어, 각각의 기술적 세부사항에 대한 전문적 지식이 없이 플랫폼 독립적 인터페이스들을 갖고 있으며 각 어플리케이션들을 통합하는데 있어서 웹 서비스 방식을 사용한다. 웹 서비스는 단순히 무엇을 어떻게, 어디에 기술하고 어떻게 찾을지에 관한 1차원적인 정의를 기술하는 서비스 지향 구조(Service Oriented Architecture)의 구현 기술이다. 따라서 견고한 기업내외의 비즈니스 상호작용을 지원하는 서비스 작성(composition)과 관리를 위한 구체적인 기본 원리를 제공하고 있는 SOA와는 구별된다[2].

한편, SOA를 지원하는 도구들을 이용하여 서비스들의 조합으로 구성된 시스템은 전체 시스템의 동작을 시뮬레이션 함으로써 각 서비스들이 잘 연결되었는지 확인 할 수 있지만, 이는 서비스 인터페이스에 한정되어 연결된 각 서비스의 입. 출력에 대한 명세가 올바르게 짝지어진 것인지를 확인할 뿐, 실제 데이터의 올바른 사용은 확인 할 수 없다. 즉, 서비스품질을 보장하는 동시에 서비스를 작성, 통합하고 관리하는 것[3]도 SOA에서의 중요한 부분이지만, 서비스들이 정확하게 작업을 수행하는 지를 검사하기 위해서는 서비스 간에 전송되는 데이터의 품질을 측정할 필요가 있으며, 보

※ 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 지원사업의 연구 결과로 수행되었음(IITA-2007-(C1090-0701-0032)).
† 준 회 원 : 이화여자대학교 컴퓨터학과 석사과정
†† 정 회 원 : 이화여자대학교 컴퓨터학과 교수
††† 종신회원 : 아주대학교 정보통신대학 전자공학부 조교수
논문접수 : 2007년 6월 15일, 심사완료 : 2007년 10월 12일

장하는 것 역시 매우 중요하다[4].

또한 SOA의 개발 방법론은 개발 환경과 플랫폼에 관계없이 서비스를 유연하게 사용할 수 있다는 이점을 갖지만, 각 서비스들이 약 결합으로 이루어져 서비스에서 다른 서비스로의 데이터 품질에 대한 관리를 수행하지는 못한다. 현재 데이터의 품질을 탐지하고 이를 정제하는 연구는 데이터 베이스, 데이터 마이닝 등의 분야에서 연구되고 있다. 그러나 SOA를 지원하기 위한 데이터 정제를 단일 데이터베이스나 데이터 마이닝 시스템의 데이터 품질 정제 도구에만 의존한다면 서비스간의 상호작용하는 데이터의 품질을 보장할 수 없고, 독립적인 컴포넌트로 개발된 정제 도구가 있다고 하더라도 데이터의 오류 탐지 규칙을 설정하고 규칙의 변화를 반영하기 위해서는 정제 엔진을 반복해서 갱신해야 하므로 융통성 있는 오류 데이터 탐지가 불가능하다. 또한 SOA기반 시스템 개발자는 서비스간의 데이터 입출력의 제약조건을 조절하기 위해서 서비스 내부를 수정해야 하는 문제점이 발생하게 된다. 따라서 서비스간의 상호 작용하는 데이터만을 별도로 관리 할 수 있는 독립적인 서비스 개발이 요구 된다.

본 논문에서는 SOA의 기본 원리에 충실하면서 서비스간의 상호 작용하는 데이터의 품질을 보장할 수 있도록 서비스 간의 실시간 데이터의 오류를 탐지하고 정제할 수 있는 방법을 제안하고, SOA를 지원하는 표준으로 XML을 사용하여 데이터 집합과 제약 조건에 대한 변화를 융통성 있게 처리할 수 있는 서비스로 개발한다. 먼저 서비스간의 정제되어야 할 오류 데이터의 체계를 확립하고 오류를 탐지하고 정제하기 위한 규칙을 제안하며, 제안된 규칙을 기반으로 SOA의 기본 원리를 만족시키기 위한 서비스를 설계하고 개발 절차를 확립한다.

개발된 오류 데이터 정제 서비스는 SOA를 기반으로 하여 작성되는 서비스의 질을 높이고 e-Business시스템과 같이 상호 작용이 많은 시스템의 데이터를 효과적으로 관리할 수 있게 한다. 실제 개발된 서비스는 SOA를 기반으로 CRM(Customer Relationship Management) 시스템과 ERP(Enterprise Resource Planing) 서비스를 통합하는데 적용하여 두 서비스간의 상호 작용하는 데이터의 오류를 정제하는 실험을 하였다. 실제 오류 데이터를 탐지 및 정제하고 이에 대한 통계를 제시함으로써 두 서비스간의 상호 작용하는 데이터의 오류를 30% 넘게 줄이는 효과를 보였다.

2. 관련 연구

2.1 기존의 데이터 품질 관리 기법

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[5]. 이러한 데이터 품질 문제를 해결하려는 노력은 여러 분야에 걸쳐 비용이 많이 소모되고 가장 시간을 많이 소비하게 만들고 있다[6]. 과거에 정확도, 완전성, 유일성, 최신성, 일관성으로 정의 되던 데이터 품질은 실제로 이런 조건들을 총체적으로 평가할 수 있는 방법이 없고, 문맥에 의존하여 무엇이 중요한지 일괄적으로 판단 내리기 힘들며 불완전하고 모호한 특성으로 데이터 품질에 대한 정의가 최근에 와서 바뀌고 있다. 따라서 데이터 품질에 대한 정의를 좀 더 실질적으로 바꾸어 데이터 사용 측면을 반영하고 프로세스에서 개선할 수 있도록 하며 메트릭을 정의하여 측정 가능하도록 한 품질로 정의하고 있다[5].

2.1.1 데이터 정제 기법

데이터의 품질이 좋지 않다고 판단되는 경우, 이를 정제하는 연구는 데이터베이스, 데이터마이닝 등의 분야에서 많이 연구되고 있다. 이러한 연구들은 데이터베이스의 한 테이블 내에 중복된 값들을 찾는 문제[7, 8], 스키마 매핑 시 이름이나 구조상 충돌을 일으키는 데이터를 정제하는 문제[9] 등으로 주로 데이터베이스의 오류 데이터를 찾아내는데 초점을 맞추고 있다. 한편 통계적인 방법을 이용하여 데이터 집합 내에서 의심스럽거나 잃어버린 데이터를 찾아내는 연구도 활발히 진행되고 있다[10]. 그러나 이미 데이터가 모두 수집된 데이터베이스 내에서의 데이터 정제 기법만을 고려한다면 웹 서비스를 이용하는 SOA기반 시스템들에서 상호 작용하는 데이터의 품질을 고려할 수 있는 방법이 없다. 본 논문에서는 서비스 통합시 데이터의 품질을 보장할 수 있는 방법을 제안한다. 이는 최근의 데이터 품질의 정의와 같이 프로세스 자체에 데이터 품질을 개선하려는 노력과 SOA의 서비스 작성과 관리를 위한 기본 원리를 따르는 목적에 부합한다.

2.1.2 기존의 데이터 정제 도구

데이터 마이닝에서는 정제된 데이터가 필수적이기에 데이터 마이닝을 기초로 하는 ERP, CRM과 같은 다양한 어플리케이션에서는 데이터 정제 도구를 필요로 하고 있으며, 이러한 이유로 MonArch[11], SLAAM[12], ZipIt[13], The Ascential™ Enterprise Integration Suite[14], HummingBird[15]와 같은 데이터 정제를 수행하는 다양한 도구들이 개발되었다.

MonArch는 CRM 솔루션으로 .NET 기반의 컴포넌트로 개발되어 다양한 CRM 기능과 고객의 정보에 관련된 데이터를 정제하는 것이 특징이며, 컴포넌트 기반으로 개발되어 재사용 및 타 어플리케이션으로의 확장에 유리하다. 하지만, 플랫폼과 구현언어에 독립적이지 못하여 상호 운용 시 문제가 발생하게 된다. SLAAM(Systematic List Analysis And March)은 고객의 주소 이전 추적이나 중복자 검색 등에 사용하고 있으나 독립적인 컴포넌트나 서비스로 개발되어 있지 않다. ZipIt[11]은 우편번호 데이터의 정제를 수행하는 도구로 자신의 자료 특성에 맞는 동의어를 쉽게 등록하여 활용할 수 있으며, Eye checking 기능과 동의어 추가 기능을 활용하여 정제 효율을 높일 수 있는 특징을 가진다. ZipIt 또한 SLAMM과 유사하여, 컴포넌트 기반의 어플리케이션 개발에 사용하기가 어렵다는 문제점을 가진다. 더불어 우편번호 데이터라는 특정된 데이터만을 정제할 수 있어 활용

범위가 좁다. The Ascential^{TM} Enterprise Integration Suite은 기업에서 사용하는 데이터의 프로파일링, 데이터 평가, 데이터 정제, 메타데이터 관리와 데이터 통합을 위한 ETL (Extraction, Transform, and Load) 도구이다. 이 도구는 다양한 데이터 소스로부터 데이터를 가져 올 수도 있고 다시 소스나 타겟, 프레임 등을 고려하지 않고서도 데이터를 통합 할 수 있는 API를 제공한다. 그러나 순수한 데이터 정제를 위한 측면에서 활용되기 어렵고 독립적인 서비스로 작동할 수 없다. HummingBird는 기업의 포털, 문서, 기록, 지식관리 및 협업, BI(Business Intelligence), 데이터 통합을 위한 기술을 포함하는 어플리케이션의 통합 솔루션으로 BI 솔루션이 기업의 정보를 보고 및 분석하는 동안 정제 도구가 그 정보를 변환 및 정제하게 된다. 그러나 BI에 종속된 하나의 기능으로 독립적으로 사용될 수 있는 컴포넌트도, 서비스도 아니다. 이러한 기존의 데이터 정제 도구들은 데이터 소스로 이동하기 전 과정에서 데이터를 정제함으로써 어플리케이션 및 엔터프라이즈의 데이터 표준에 부합할 수 있도록 해주나, 개발된 어플리케이션에서 지원해주는 특정 데이터만 정제하는 단점을 가진다.

따라서 본 논문에서는 데이터품질 정제 도구를 종속되는 시스템이나 어플리케이션과는 독립적으로 동작할 수 있는 서비스를 개발한다. 아래의 <표 1>은 지금까지 설명한 기존의 데이터 정제 도구들의 특성과 본 논문에서 제안한 CleanS (dirty data Clean Service) 서비스 데이터 정제 도구를 비교한 표이다.

2.2 SOA

2.2.1 기본 원리

SOA는 서비스 개발에 견고함을 제공해 줄 수 있도록 서비스 작성과 관리를 위해 서비스 정의 시 명시적인 구현-독립적인 인터페이스 사용, 위치 투명성과 상호 작용을 강조한 통신 프로토콜의 사용 과 재사용 비즈니스 기능을 캡슐화한 서비스 정의 같은 기본 원리를 정의하고 있다[17].

이러한 기본 원리를 만족시키기 위해 SOA는 크게 서비스 기술(description)과 기본적인 오퍼레이션, 서비스 작성, 그리고 서비스 관리 계층으로 나누어 그 역할과 기능을 정의하고 있다. 서비스 기술과 기본적인 오퍼레이션에는 웹 서비스와 유사하게 서비스 제공자와 고객 사이의 서비스 기술, 공개, 검색 등의 역할을 포함한다. 서비스 작성 계층에서는 여러 서비스들 간의 실행과 데이터 흐름을 관리하는 조화(coordination) 기능, 서비스에서 발생하는 이벤트들을 필요한 곳에 알리는 모니터링(monitoring) 기능, 서비스들 간의 파라미터 타입이나 제약 조건의 적합성(conformance) 기능, 그리고 전체적으로 서비스 작성 시 비용, 수행 능력, 보안, 인증, 무결성, 신뢰성, 확장성, 가용성 등의 서비스의 질을 보장할 수 있는 작성 (Quality of Service composition) 기능을 강조하고 있다. 또한 서비스 관리 계층에는 엔터프라이즈들이 서비스 플랫폼, 즉 서비스나 어플리케이션 전략을 관리하도록 해주는 여러 가지 기능을 제공한다[3].

이렇게 SOA는 컴포넌트 기술이나 웹 서비스 기술로 분산 컴포넌트들을 서비스로 다루고 이들을 통합하기 위해 여러 가지 설계 및 관리에 관한 기본 원리를 정의하고 있는 새로운 시스템 통합 패러다임이라고 볼 수 있다.

따라서 본 논문에서는 SOA의 기본 원리에 충실하면서 서비스간의 상호 작용하는 데이터의 품질을 보장할 수 있도록 서비스 간의 데이터의 품질을 탐지하고 정제할 수 있는 방법을 제안하고 이를 서비스로 개발한다.

2.2.2 SOA 지원 도구

SOA를 지원하기 위한 대표적인 최신 통합 기술로서 ESB (Enterprise Service Bus) 개념이 등장하였다. ESB는 SOA의 기본 원리를 따르면서 통합 인프라 구조를 제공하고 이벤트 중심의 분산 시스템 환경을 가능하도록 SOA를 지원하는 논리적인 버스의 개념이다.

〈표 1〉 기존 데이터 정제 도구와 본 논문에서 제안한 CleanS와의 비교

	정제 데이터 유형	동작 형태	환경	타 어플리케이션으로의 확장	문제점 및 특성
MonArch	CRM	종속	컴포넌트 기반 .net	△	- 플랫폼과 구현언어에 독립적이지 못함. - 상호운용 시 문제 발생. - 한 솔루션의 일부로 구현되어 정제를 목적으로 재사용 어려움.
SLAAM	CRM	독립	X	△	- 컴포넌트나 서비스로 개발되지 않아 재사용, 확장이 어려움.
ZipIt	ZIPCODE	독립	X	△	- 컴포넌트나 서비스로 개발되지 않아 재사용, 확장이 어려움. - 우편번호 데이터라는 특정된 데이터만을 정제할 수 있어 활용 측면에서 부족함.
Ascential Software	CRM	종속	X	△	- 데이터 정제를 위한 측면에서 활용되기 어렵고, 독립적인 서비스로 작동할 수 없음.
HummingBird	CRM	종속	X	△	- 컴포넌트나 서비스로 개발되지 않아 재사용, 확장이 어려움.
CleanS	없음 (유동적)	독립	SOA 기반	○	- SOA 기반의 서비스로 개발되어 플랫폼과 구현 언어에 독립적. - 현재는 6가지의 데이터 정제 규칙을 기반으로 하지만 규칙을 정규 표현 형태로 유동적으로 추가 가능하여 정제되어야 할 데이터의 특성에 영향 받지 않음. - 선행되어야 할 서비스와 후속 서비스의 종류에 상관없이 재사용 가능.

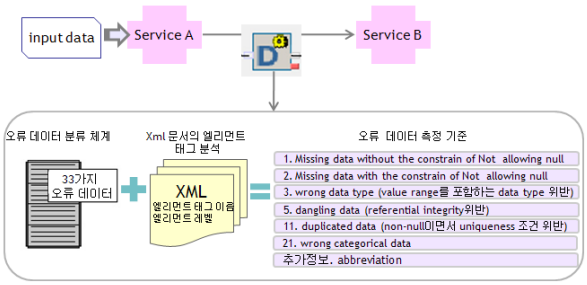
ESB를 위해 기본적으로 커뮤니케이션, 통합, 서비스 상호 작용과 관련된 특성들을 만족시켜야 하며 그 외에도 보안, QoS (Quality of Service), 메시지 프로세싱 등과 같은 특성을 지원한다. 또한 SOA를 지원하는 ESB는 JMS(Java Messaging Server), 웹 서비스, XML 등 표준을 기반으로 시스템을 통합하고[16], 분산된 서비스들에 대하여 이벤트 중심의 비즈니스 프로세스 관리를 수행한다. 비즈니스 프로세스의 추가적인 프로그래밍 없이 동적으로 시스템을 설정하고 미들웨어를 자동으로 구성하며, 동적 확장성과 변화 관리가 수월하다는 장점을 가지고 있다. 이러한 SOA의 ESB 개념을 지원하기 위한 도구들은 IBM, BEA, SAP, HP, MS, Oracle, Temax, Fiorano Software등의 회사에서 개발 하고 있다. 본 논문에서도 Fiorano Software의 ESB를 기반으로 서비스를 개발하여 실제적인 적용 사례를 제시할 것이다.

3. 오류 데이터 측정 기준

이 장에서는 본격적인 오류 데이터 정제 서비스를 설계하기 전에 서비스 간에 오류로 판단되는 데이터의 종류를 분류한다. 선행 연구로서 데이터베이스로의 데이터 수집, 통합, 저장 등에서 발생 할 수 있는 오류들을 33가지[17] 분류하였고, 본 논문에서 요구되는 서비스간의 상호 작용 시 발생할 수 있는 오류를 위해서는 재정리한 6가지[18] 데이터 오류 타입을 이용하여 SOA상에서 오류 데이터 정제를 위한 기본적인 서비스를 설계한다. 또한 입출력에 관련된 모든 명세를 데이터 교환의 표준으로 자리 잡은 XML로 명세화 한다[19].

(그림1)과 같이 오류 데이터 정제 서비스에서 오류 데이터 측정 기준은 데이터가 전송되는 단계에서의 오류데이터 분류 체계를 기반으로 하며, 실제로 사용되고 있는 여러 XML문서의 엘리먼트 태그를 통해서 어떠한 오류 데이터가 발생할 수 있는지를 분석한 후, 그 분석을 바탕으로 시스템을 구성하는 각 서비스들은 오류 데이터 정제 서비스에게 제공하는 오류 데이터 측정 기준을 바탕으로 자신의 XML 데이터와 기능에 맞는 기준을 선택한다.

XML 문서들에서 엘리먼트의 태그이름과 각 엘리먼트의 레벨 및 데이터들을 바탕으로 문서의 DTD(Document Type Description)와 실 데이터를 이용하여 오류 데이터의 측정 기준을 분석한다. 각 엘리먼트의 태그들이 가지고 있는 데이터가 지닌 의미와 적용 가능한 오류 데이터의 기준을 찾아내고, 서비스들이 데이터를 전송할 때 검사해야 할 오류들을 분석하여 해당 기준을 설정한다.



(그림 1) 오류 데이터 정제 서비스를 위한 오류 데이터 측정 기준

ESB 상에서 서비스 라이브러리에 등록된 서비스에서 표준화된 데이터의 입출력을 위하여 사용하는 XML 데이터의 DTD뿐만 아니라, 여러 XML 문서들을 분석하여 오류 데이터 측정 기준을 일반화 한다. 이러한 과정을 통해 얻어진 오류 데이터 측정 기준들이 오류 데이터 정제를 위한 XML 문서에 새로운 DTD 형식에 따라 입력된 데이터에 오류 데이터 측정 기준이 삽입된다.

XML 문서의 엘리먼트에 대한 분석을 통해 얻어진 오류 데이터 측정 기준들 가운데 각 서비스들은 자신들이 사용하는 데이터가 어떠한 측정 기준을 적용하여 오류 데이터를 분류할 것인지를 결정한다. XML 문서로부터 오류 데이터 분류 체계 중에서 관련 있는 항목들을 찾아내고, 필요에 따라 세부 카테고리를 지정하여 오류 데이터 측정을 위한 기준을 찾아낸다. XML 문서로 입력된 데이터의 오류는 사용자가 지정한 오류 데이터 측정 기준에 따라 탐지한다.

데이터 정제 서비스는 실시간으로 서비스들이 서로 주고받는 XML 문서를 대상으로 하고 있기 때문에 필요에 따라 오류 데이터 측정 기준을 확장할 수 있어야 한다.

4. 오류 데이터 정제 서비스

오류 데이터 정제 기법을 이용하여 SOA상에서 오류 데이터 정제를 위한 기본적인 서비스를 제안한다.

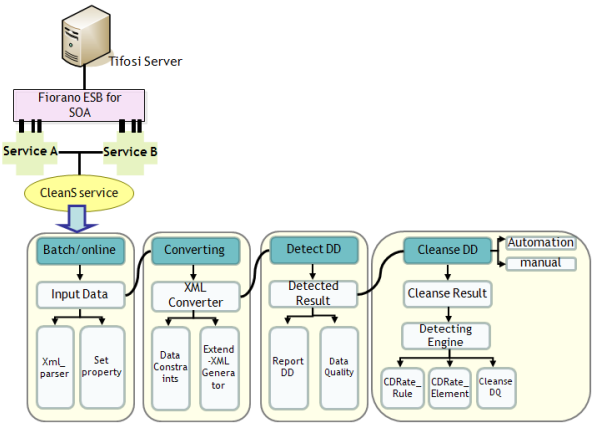
4.1 CleanS 서비스

오류 데이터 정제 서비스(**CleanS** : dirty data **Clean S**ervice)는 서비스와 다른 서비스를 연결하는 중간에 사용하며, 이는 한 서비스에서 다른 서비스로 데이터가 넘어갈 때 오류가 발생하는지 아닌지를 탐지하고 정제하는 서비스이다.

CleanS 서비스는 SOA를 기반으로 서비스를 개발하고 시스템을 통합하는 과정에서 이벤트가 발생하기를 기다리고 이벤트 발생 후 서비스의 입력으로 XML_Parser 모듈을 사용하여 데이터들을 분류한다. 서비스 사용자로부터 각 데이터에 따라 오류 데이터 측정 기준을 입력 받은 후, XML 문서로 변환한다. 사용자 입력 데이터와 측정 기준이 포함되어 있는 XML 문서를 가지고 오류 데이터를 탐지하여 그 결과를 보여 주고, 서비스 사용자가 오류 데이터를 정제할 수 있는 환경을 제공한다. 오류 데이터를 정제한 후, 현재 입력된 데이터의 오류 데이터 측정 기준 및 데이터에 따른 오류 데이터 발생률과 서비스 적용 후 오류 데이터 발생률을 비교해 줌으로써 오류가 얼마나 정제 되었는지를 보여준다.

다음의 (그림 2)는 CleanS 서비스의 모듈의 흐름을 나타낸다.

서비스는 Windows 2000 server 환경에서 Java 2 Platform, Enterprise Edition 1.4.2를 기반으로 구현되었고, SOA를 지원하는 ESB 상에서 서비스로 제공되기 위하여 Fiorano사의 Fiorano Business Integration Suite인 Fiorano ESB™을 사용하였다.



(그림 2) CleanS 서비스 모듈

또한, CleanS 서비스는 아래에 기술된 기능을 수행하는 모듈들로 구성되어 있다.

- **일괄처리/온라인 단계(Batch/Online)** : CleanS 서비스가 무엇을 대상으로 오류 정제를 측정할 것인지를 선택하는 모듈로써 오프라인의 데이터 품질에서 온라인 데이터 품질까지 그 대상을 확장하기 위한 기능을 제공한다.
- **Input Data(서비스 입력 값)** : 시스템을 구성하는 서비스의 입력으로써, XML 데이터가 이벤트를 통해 입력되는 지를 확인한다.
- **변환(Converting)** : 입력데이터와 사용자의 데이터제약조건을 결합시키는 과정을 제공한다.
- **XML 변환(XML Converter)** : 서비스 A로부터 입력된 데이터들을 XML 엘리먼트로 일대일 매핑 하고 필요에 따라 계층 구조를 줄 수 있다.
- **사용자의 데이터 제약 조건 (Data Constraints)** : 먼저 서비스 개발자로부터 입력 데이터들에 대해 적용되어야 할 규칙과 필요에 따라 값의 범위 및 카테고리에 대한 제약 조건을 입력 받는다. 현재 CleanS 서비스는 (그림 1)에서 제시한 대로 6가지의 규칙을 제공하며, 각 규칙별로 사용자가 정의해야 하는 값은 다르다. 규칙 1, 2, 5, 11의 경우 규칙 번호만 정의하면, 사용자의 별도의 입력 없이 오류를 탐지할 수 있고, 규칙 3은 값의 유효 범위를 검사하므로 값의 최소값과 최대값을 값의 범위로서 정의해야 한다. 마지막으로 규칙 21은 미리 정의해 놓은 우편번호, 날짜, 전화번호, 이메일 주소, URL, 파일이름, 연산형식, 숫자의 8가지 범주 중에 하나를 선택한다.
- **확장-XML 문서 생성 (Extended-XML Generator)** : 이렇게 얻은 데이터 제약 조건은 XML로 변환된 입력 문서와 결합하여 각 데이터 별 제약 조건이 명시된 XML 문서로 확장된다. (그림 3)은 CRM 고객 주문 서비스 양식의 한 예로서 인터페이스 양식을 XML로 변환한 것이다. 데이터 명세를 XML로 표현하며 보통 컴포넌트 기반

```
<?xml version="1.0" encoding="UTF-8" ?>
- <Order>
    <OrderID>654435</OrderID>
    <CustomerID>987651</CustomerID>
    <Email>ayrton@florano.com</Email>
  - <ShipDetails>
      <Mode>Air</Mode>
      <Address>Fiorano Software Inc., 718 University Avenue, Suite
        212, Los Gatos, California.</Address>
      <Country>USA</Country>
      <PostalCode>95032</PostalCode>
      <PhoneNumber>408.354.3210</PhoneNumber>
    </ShipDetails>
    <Product>CISCO 1601R ROUTER</Product>
    <Quantity>1</Quantity>
    <Price>1000$</Price>
    <BillAmount>1000$</BillAmount>
    <OrderDate>15/02/04</OrderDate>
  </Order>
```

(그림 3) CRM 고객 주문 서비스 양식의 XML 명세

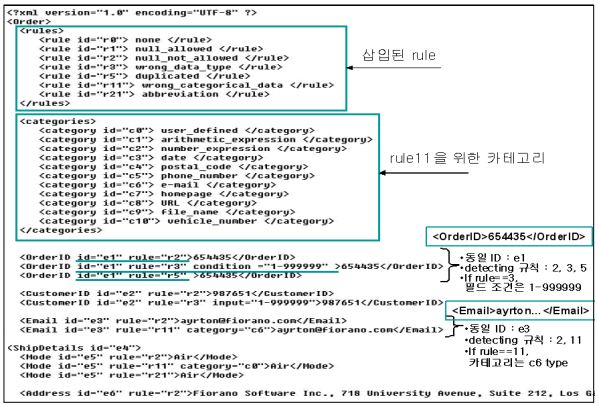
```
<!ELEMENT ruleSpecification
(rules, categories, elements)>
<!--Rule Declaration-->
<!ELEMENT rule (rule*)>
<!ELEMENT rule #PCDATA>
<!--Categor Declaraion-->
<!ELEMENT categories (category*)>
<!ELEMENT categories #PCDATA>
<!--Data Declaration-->
<!ELEMENT elements (E*)>
<!ELEMENT E (child_element* | #PCDATA)>
<!--Atrribute Declaration-->
<!ATTLIST rule id ID#REQUIRED>
<!ATTLIST category id ID#REQUIRED>
<!ATTLIST E
                    id          ID      #IMPLIED
                    rule        IDREF   #IMPLIED
                    category    IDREF   #IMPLIED
                    input 1     CDATA   #IMPLIED
                    input 2     CDATA   #IMPLIED>
```

(그림 4) 확장 기준 DTD

인터페이스와는 달리 각 데이터들의 계층 구조를 표현할 수 있는 것을 볼 수 있다. 따라서 <ShipDetails> 엘리먼트는 <Mode>, <Address>, <Country> 등의 선적에 관련된 정보를 자세히 기술함으로써 이 엘리먼트 집합이 모두 하나의 엘리먼트, <ShipDetails>를 보충 설명하기 위한 묶음이라는 것을 드러낼 수 있다. 또한 입력 데이터와 제약 조건에 대한 융통성 있는 처리를 위해서 두 데이터를 결합하는 만큼, (그림 4)와 같은 XML DTD을 기준으로 확장한다.

이 DTD는 규칙 선언 부분, 특별히 '잘못된 범주 데이터'를 걸러내기 위한 규칙 11을 위한 범주, 입력 명세로 받아들인 엘리먼트들과 데이터 값, 그리고 각 엘리먼트 별 제약 조건의 네 부분으로 구별된다. 이 확장 기준이 되는 DTD는 다음과 같은 조건을 만족시킨다.

- 확장 조건 1. 하나의 엘리먼트는 다수 중복되어 올 수 있으므로 규칙과 그에 따른 범주가 스키마에 고정되지 않고 융통성 있게 확장될 수 있다.
- 확장 조건 2. 하나의 엘리먼트에 여러 규칙에 의해 제



(그림 5) 제약 조건과 결합된 XML 입력 명세

약 받는다면 속성으로 정의되는 엘리먼트 ID에 의해 구별된다.

- 확장 조건 3. 규칙 11을 위한 범주는 다른 엘리먼트의 선언이나 규칙의 선언에 영향을 주지 않고 확장 가능하다.

이 DTD를 기준으로 다음 (그림 3)의 XML 입력 명세를 확장하면 (그림 5)와 같다.

<rules> <rule...> ... </rules> 부분과 <categories> <category ...> ...</categories> 부분은 서비스를 위해 설계된 규칙과 범주가 그대로 첨가되는 모습을 보여준다. 엘리먼트 선언 부분에서 기존의 <OrderID>654435</OrderID> 엘리먼트와 데이터 값이 세 줄에 걸쳐 확장된 것을 볼 수 있다. <OrderID>는 서비스 사용자에 의해 규칙 2, 3, 5의 제약 조건이 적용되어 세 번의 엘리먼트 선언이 중복된다. 그러나 엘리먼트의 ID 속성에 의해 “e1”이라는 값이 동일하게 부여되어 후에 XML 파서는 이를 동일한 엘리먼트임을 확인할 수 있다 (확장 조건 2).

- **오류 데이터 탐지 단계(Detect DD) :** 사용자에 의해 정의된 속성과 오류 데이터 측정 기준에 의하여 데이터 정제를 측정하여 오류 데이터를 탐지를 한다. 확장-XML 문서를 XML 파서를 이용하여 필요한 정보만을 추출한다. 파싱 결과 각 데이터 별로 데이터 이름(element name), 데이터 ID(element ID), 처리되어야 할 규칙 번호 (rule No.), 카테고리(category No.), 규칙에 필요한 제약 조건(condition)이 추출된다. 예를 들어 (그림 5)의 문서는 다음과 <표 2>와 같이 추출된다.
- **탐지 (Detecting Engine) :** 파싱 후 얻은 데이터 정보를 오류 데이터 측정 기준을 적용한다. 대략적인 알고리즘은 다음 (그림 6)과 같다.

오류 탐지 알고리즘에서 규칙과 카테고리 리스트 정보는 후에 통계를 위해 테이블에 각각 저장되고 <표 2>와 같이 데이터의 모든 정보를 담은 n개의 element 리스트를 입력으로 한다. 모두 6개의 규칙별로 입력되는 데이터의 오류를 걸러내며, 걸러 내는데 사용된 규칙에 따라 error_type을 설정하게 된다.

규칙 1(case 1)은 ‘Null’값은 허용하고 데이터 값이 존재하지 않는 데이터를 찾으며 규칙 2(case 2)는 ‘Null’ 값도 허용하지 않고 데이터 값이 존재하지 않는 데이터를 찾는다. 규칙 3(case 3)은 추가 정보로 입력 받은 값의 범위를 벗어난 데이터를 찾아내며 규칙 5(case 5)는 이미 수집된 데이터 집합에서 중복되면 안 되는 제약 조건을 가진 데이터의 중복 여부를 확인한다. 또한 규칙 11(case 11)은 미리 정의해 놓은 8개의 값의 범주 규칙을 벗어난 데이터를 걸러 내며 category lookup 테이블은 규칙 11을 위한 각 데이터의 범주에 따른 정규 표현식을 설정해 놓은 것으로 만약 검사해야 할 표현식이 달라질 경우category lookup 테이블의 표현식 변경하면 된다. 또한 규칙 11 은 특별한 도구를 사용하거나 시소러스를 고용한 상태에서 오류를 걸러 내는 것이 아니라 오류를 완벽하게 찾아 낼 수는 없다. 그러나 정규 표현(regular expression)을 이용하여 Date는 yy/mm/dd, yyyy/mm/dd, 우편번호나 전화번호 자릿수, 메일 주소는 string@string(.string)+, 홈페이지나 URL http://(string.)+string(/string)에 맞추어 오류를 걸려 낼 수 있다. element 리스트의 ruleNo를 참조하여 규칙을 적용하고 위배되면 error_type을 설정하게 된다.

규칙 21(case 21)을 위한 user-defined library는 (Doctor, Dr.), (Drive, Dr.), (road, rd) 와 같이 일반적인 단어와 그 약어를 정리 해 놓은 것으로 계속해서 갱신될 수 있다. 본래 element 정보에 첨부된 error_type은 오류 정제 시 그 데이터가 어떠한 규칙에 위배 되었는지를 서비스 사용자에게

〈표 2〉 확장-XML 문서로부터 추출된 정보

elementName	elementID	ruleNo	category	condition1	condition2	data
orderID	1	2	-	-	-	654435
orderID	1	3	-	1	999999	654435
orderID	1	5	-	-	-	654435
customerID	2	2	-	-	-	987651
customerID	2	3	-	1	999999	987651
Email	3	2	-	-	-	ayrton@fiorano.com
Email	3	11	6	-	-	ayrton@fiorano.com
:	:	:	:	:	:	:

```
procedure Detecting( ruleNo[1..i] : pairs of rule No. and rule name, category[1..j] : pairs of category No. and category name, element [1..n]:
                     list  including element name, ID, ruleNo, category, conditions, data)
returns element [1..n] including detected error types
begin
    move all pairs of ruleNo to rule_table;   move all pairs of category to cat_table;
    for (k=1; k<=n; k++)
    begin
    switch (rule No) {
  case 1: if data == empty then error_type = 1;
  case 2: if (data == 'Null' or empty) then error_type = 2;
  case 3: if (data < codition1) and (data > condition2) then error_type = 3;
  case 5: compare C&P(data); // compare data of current and data with identical element name of previous data stream, find
                             duplicated data
  case 11: // category lookup error type 11.1~11.8
            if (category == 1)  // arithmetic expression
            then check data if the form follows [0-9]+{[+-*/]+[0-9]+}+;
            else if (category == 2) // number expression
                 then check data if the form follows [A-Za-z]*[0-9]+[A-Za-z]*;
            else if (category == 3) // date
                 then check data if the form follows 'yyyy/mm/dd or yy/mm/dd'
            else if (category == 4) // postal code
                 then check data if the form follows 'd=[0-9], ddddd or ddd-ddd'
            else if (category == 5) // phone number
                 then check data if the form follows 'd=[0-9], ddd-ddd-ddddd'
            else if (category == 6) // e-mail address
                 then check data if the form follows 'string@string(.string)'
            else if (category == 7 or 8) // home page or URL
                 then check data if the form follows 'http://(string.)+string(.string)*'
            else if (category == 9) // file name
                 then check data if the form follows 'string(.string)*'
 case 21: if data is not in user-defined library for abbreviation
                 then error_type = 21;
  }
   append error_type to the end of the list (element[k]);

end
end
```

(그림 6) 오류 탐지 알고리즘

제시함으로써 정제 행위를 돕는다.

- **오류 데이터 탐지 결과 단계(Detected Result)** : 분류된 오류데이터를 리포트하고, 전체 데이터의 정제와 칼럼별 데이터 정제를 그래프로 보여준다.
- **오류 데이터 정제 단계(Cleanse DD)** : 정제를 위한 정보는 오류로 탐지된 데이터 이름과 값을 동시에 제공한다. 아직까지 대부분의 연구에서의 정제 과정은 도메인 전문가에게 맡기고 있다. 따라서 본 논문에서도 정제 행위의 주체는 서비스 사용자로서 앞의 탐지 과정에서 위배된 규칙과 카테고리를 제시함으로써 정제를 위한 최대 정보를 제공하는 것을 목표로 한다. 만약 서비스 사용자도 정제할 수 없는 데이터라면 그대로 처리하지 않고 본 서비스의 앞 단에 결합된(여기에서는 서비스 A) 서비스에 피드백 메시지를 줄 필요가 있다.
- **오류 데이터 정제 결과 단계(Cleansed Result)** : 사용자로 하여금 오류데이터가 얼마나 정제되었는지를 쉽게 파악할 수 있도록 소스 데이터와 정제된 데이터의 오류 데이터 발생률을 비교함으로써 오류 데이터의 정제율을 보여준다.

이로써 ClsenS 서비스는 서비스 A와 B사이에 삽입되어 데이터의 오류를 탐지하고 이를 토대로 오류를 정제함으로써 보다 질 좋은 데이터를 서비스 B에게 넘길 수 있게 된다. 다음 장에서는 이 장에서 제안한 오류 데이터 탐지 및 정제 서비스를 실제 적용한 사례를 제시하고 그 결과를 분석할 것이다.

5. 서비스 구현 및 적용

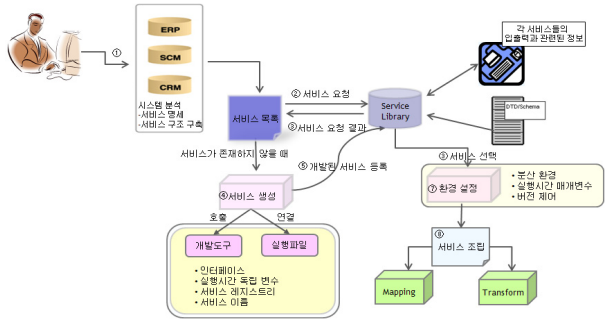
이 장에서는 실제 본 논문에서 제안한 오류 탐지와 정제 기법을 기반으로 ESB 상에서 오류 데이터 정제를 위한 서비스를 구현하고 이를 실제 분산 시스템에 적용한 결과를 보일 것이다.

5.1 서비스 개발 절차

오류 데이터 정제를 위한 서비스는 전체 시스템이 구성되어 실시간의 데이터에 대하여 수행되기 때문에, 오류 데이터 정제 서비스 역시 ESB 상에서 서비스로 구현되어야 한다. ESB는 시스템에서 필요로 하는 서비스를 추출하여 명세화하고, 이에 따라 기존의 서비스를 재사용하거나 새로운 서비스를 생성하여 서비스 라이브러리에 등록한 후 저장된 서비스를 사용하여 전체 시스템을 구성할 수 있는 개발 환경을 제공한다.

(그림 7)은 서비스를 개발, 등록, 사용하는 절차를 보여준다.

① 시스템 분석: 요구된 시스템을 분석하고, 분석 결과로 얻어진 요구되는 서비스를 명세화하고 시스템 구조를



(그림 7) 서비스 개발

구축한다.

② 서비스 요청: 시스템 분석을 통해 필요한 서비스 목록을 얻고, 필요한 서비스들의 기능에 적합한 기능 명세와 입출력 인터페이스 및 DTD/Schema를 정의해 놓은 서비스 라이브러리에 요청한다.

③ 서비스 요청 결과 전송: 서비스 라이브러리에 요청된 서비스의 존재 여부를 결과로 전달 받는다.

④ 새로운 서비스 생성: 서비스가 존재하지 않을 경우에는 개발 도구를 호출하여 서비스 개발 환경을 구성하거나, 실행파일을 연결함으로써 서비스를 생성한다. 이때 서비스와 관련하여 인터페이스, 실행시간 독립변수, 레지스트리, 서비스 이름 및 프로퍼티 등을 정의한다.

⑤ 서비스 등록: 새로 개발된 서비스의 기능을 입출력 인터페이스 및 DTD/Schema를 서비스 라이브러리에 등록한다.

⑥ 서비스 선택: 시스템을 위해 필요한 서비스들을 선택한다.

⑦ 환경 및 속성 설정: 시스템으로 통합되는 서비스들의 환경 및 속성(분산 환경, 실행시간, 매개변수, 버전제어 등)을 설정한다.

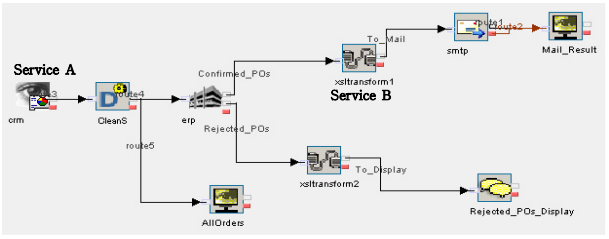
⑧ 시스템 조립: XML DTD로 명세 화된 I/O를 연결함으로써 시스템을 조립한다. 서비스들은 정의된 서비스의 인터페이스를 단순 매핑 시키는 방법과 서비스간의 I/O를 테일러링 하는 방법으로 전체 시스템으로 조립된다.

5.2 서비스 적용 사례

5.2.1 적용 대상

본 논문에서는 CleanS 서비스 활용사례로써 아래 (그림 8)과 같이 SOA시스템에 적용한 환경은 물품 거래 시스템으로 제품을 주문하는 CRM 서비스(Service A), 주문에 대한 승인과 거절을 결정하는 ERP 서비스(Service B)를 중심으로, 거래 정보를 전달하고 보여주는 여러 서비스들을 통합하여 서비스를 구성하였다. Service A와 Service B를 결합할 때 두 시스템간의 상호 작용하는 데이터의 오류 관리를 위해 본 논문에서 개발한 “CleanS (dirty data **Clean** Service)”를 적용하였다.

물품 구매자가 고객의 정보를 담당하는 CRM 시스템을 통하여 개인 정보와 함께 구매 정보를 제품 관리 시스템인



(그림 8) CRM 서비스와 ERP 서비스 사이의 오류 데이터 정제 CleanS 서비스 적용 예

ERP 시스템에 전송한다. ERP 시스템은 전달 받은 정보를 가지고 주문을 승인할 것인지 거절할 것인지를 결정한 후, 구매자에게 이메일을 통해서 승인된 결과를 전송(그림에서 ‘erp’ 뒷단에 이어지는 Confirmed_POs 경로)하기도 하고 대화창을 통해 거절된 결과를 전송(Rejected_POs 경로)하기도 한다. 이러한 적용 예는 여러 시스템들이 통합되고, 시스템 간에 대규모 데이터가 이동하는 대표적인 시스템으로 볼 수 있다. 구현된 오류 데이터 탐지 및 정제 서비스(CleanS)가 CRM 시스템과 ERP 시스템 사이에 독립적으로 삽입되어 사용자가 지정한 목적에 따라서 사용자의 관점에서 데이터의 오류를 탐지하여 구매자의 잘못된 선택, 서비스 간 데이터의 처리 범위가 다른 경우, 네트워크 전송 장애로 인한 데이터 전송 오류 등을 포함한 CRM과 ERP간의 상호 작용 데이터의 오류들을 실시간으로 탐지해 낼 수 있다.

5.2.2 서비스 실행

다음 (그림 9)는 사용자 시나리오를 도식화 한 것이다. 개발된 서비스의 입력 변환, 탐지 및 정제 , 결과에 대한 단계별 사용자 시나리오를 시퀀스 다이어그램으로 표현하였다. 시나리오는 하나의 입력 이벤트에 대한 사용자의 시나리오를 가정하였고 여러 이벤트에 대한 통계는 5.3절의 적용 결과에서 설명한다.

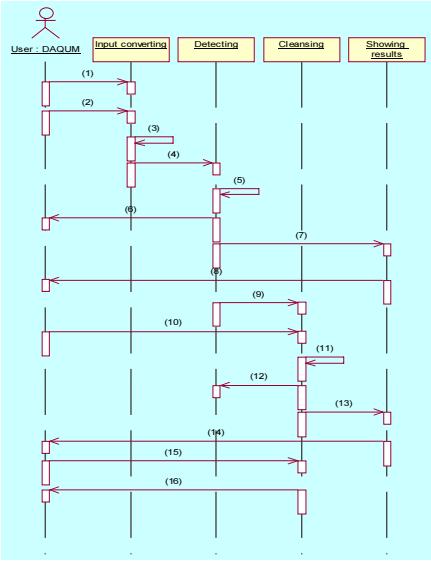
5.2.2.1 입력 변환 과정

먼저, 본 서비스와 연결된 CRM 시스템으로부터 데이터 집합을 입력 받는다(그림 9의 (1)). 입력 데이터 집합은 자동으로 아래 (그림 10(a))과 같은 DTD를 기준으로 입력 데이터 집합을 XML화하고 이 XML 문서를 토대로 사용자에게는 (그림 10(b))과 같은 화면이 제공된다. 사용자는 이 화면을 통하여 (그림 10(c))과 같은 입력 데이터에 대한 제약조건을 입력한다(그림 9의 (2)). 그리고 이 두 입력을 토대로 제약조건으로 확장된 XML문서를 생성한다(그림 9(3)).

5.2.2.2 탐지 및 정제 과정

입력변환 과정에서 생성된 데이터와 제약 조건이 결합된 XML 문서를 바탕으로 데이터의 오류 탐지 알고리즘을 이용하여 오류 데이터를 분류하고, 사용자에게 보여줌으로써 사용자가 오류 데이터를 정제할 수 있는 환경을 제공한다.

사용자에게 규칙 1, 즉 ‘Null도 허용하는 상태에서 empty인 데이터’는 오류 탐지를 위한 제약 조건을 적용한 데이터

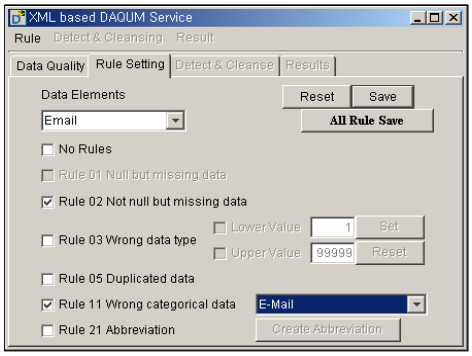


(1) Input Data Set
(2) Input Constraint of Data Set (Rule Setting)
(3) Generate Extended_XML
(4) Parse Extended_XML
(5) Detect Dirty Data
(6) Display Dirty Data
(7) Requesting dirty data rate
(8) Display dirty data rate
(9) Request cleansing dirty data
(10) Input new data for cleansing
(11) Cleanse Dirty Data
(12) Request detecting dirty data after cleansing
(13) Request displaying the results
(14) Display Dirty Data Rate (cleansed data)
(15) Request sending data set to next service
(16) Convert cleansed data to XML and send Data to next service

(그림 9) 사용자 시나리오

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Order [
        <!ELEMENT  Order              (OrderID,CustomerID,Email,ShipDetails,
        <!ELEMENT  OrderID      (#PCDATA)>
        <!ELEMENT  CustomerID   (#PCDATA)>
        <!ELEMENT  Email        (#PCDATA)>
        <!ELEMENT  ShipDetails  (Mode,Address,Country,PostalCode,PhoneNumber)>
        <!ELEMENT  Product      (#PCDATA)>
        <!ELEMENT  Quantity     (#PCDATA)>
        <!ELEMENT  Price        (#PCDATA)>
        <!ELEMENT  BillAmount   (#PCDATA)>
        <!ELEMENT  OrderDate    (#PCDATA)>
        <!ELEMENT  Mode         (#PCDATA)>
        <!ELEMENT  Address      (#PCDATA)>
        <!ELEMENT  Country      (#PCDATA)>
        <!ELEMENT  PostalCode   (#PCDATA)>
        <!ELEMENT  PhoneNumber  (#PCDATA)>
]>
```

(a) 서비스의 입력 데이터 DTD



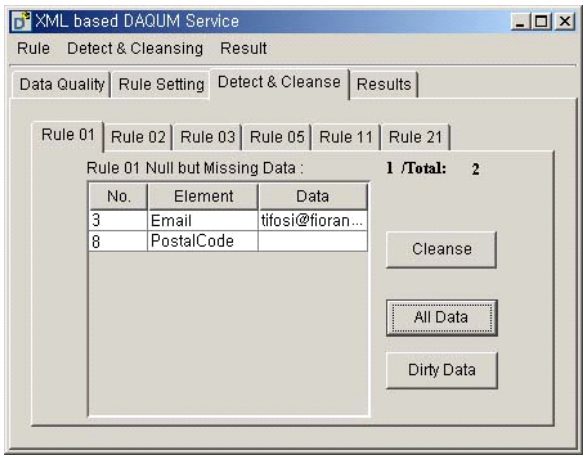
(b) 데이터 제약 조건 사용자 입력 과정

No.	데이터의 엘리먼트	오류 측정을 위한 기준
1	Order	No Rule
2	OrderID	Rule 2, Rule 3, Rule 5
3	CustomerID	Rule 2, Rule 21
4	Email	Rule 1, Rule 11(E-Mail)
5	ShipDetails	No Rule
6	Mode	Rule 2, Rule 21
7	Address	Rule 2
8	Country	Rule 2, Rule 21
9	PostalCode	Rule 1, Rule 11(Postal Code)
10	PhoneNumber	Rule 2, Rule 11(PhoneNumber)
11	Product	Rule 2, Rule 21
12	Quantity	Rule 2, Rule 3,Rule 11(Number Expression)
13	Price	Rule 2, Rule 3, Rule 11(Number Expression)
14	BillAmount	Rule 1
15	OrderData	Rule 2, Rule 11(Data)

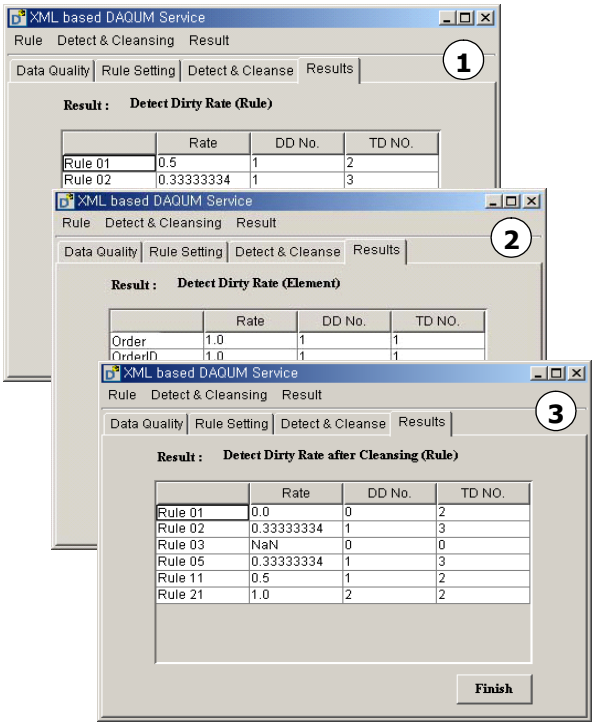
(C) 전체 데이터 오류 탐지 규칙

(그림 10) 서비스 실행 1 : 입력 변환 단계

를 모두 보여 주는 화면이다. 이 규칙을 검사한 데이터는 총 2개의 데이터로 3번째(No.3) 데이터인 'Email'과 8번째(No.8) 데이터인 'PostalCode'를 보여주고 있다. 그 중, 하나만이 위배되었음을 알려 주고 (즉 'Rule 01 Null but Missing Data: 1/Total : 2(그림 11)), 사용자는 이 탐지 결과를 보고 'Email'은 empty가 아니므로 empty 데이터인 'PostalCode'만을 정제 하게 된다(그림 9(9)). 그렇다면 'PostalCode'에 필요한 데이터(예를 들어 '12-340')를 사용자가 입력하여(그림 9(10)) 오



(그림 11) 서비스 실행 2: 탐지 및 정제 화면



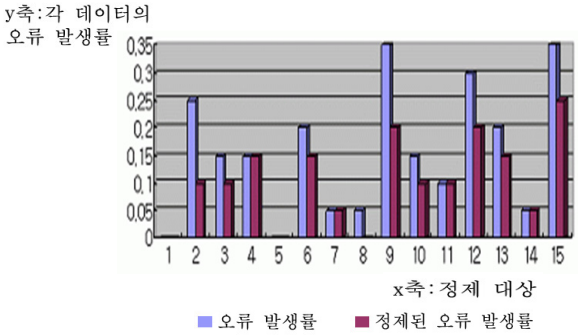
(그림 12) 오류 데이터 정제 결과 화면

류 데이터를 정제하게 되고(그림 9(11)), 필요한 정제 과정을 마치면 다시 한 번 탐지 과정(그림 9의 (12), (5))을 거칠 수 있다. 이 외에도 사용자에게 데이터에 대한 정보를 보여주거나('All Data') 오류 데이터만(Dirty Data)을 보여줄 수도 있다.

5.2.2.3 정제 결과

오류 데이터를 탐지하고 정제하는 과정이 끝나면, 오류 데이터에 대한 결과를 보여 준다.

(그림 12)의 ① 은 탐지 직후, 규칙에 따른 오류 데이터의 발생률로 조사된 오류 데이터 수(DD No.)와 데이터의 수(TD No.) ②는 데이터에 따른 오류 발생률, 마지막 ③은 정제 후, 오류 발생률을 비교한다.



(그림 13) 전체 데이터의 오류율과 서비스 적용 후 오류 데이터 발생률

5.3 적용 결과

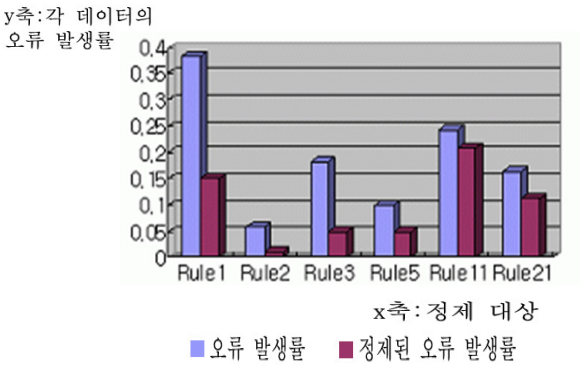
서비스들로 시스템을 구성하고 데이터 품질을 관리하면, 서비스가 실행된 후 서비스가 사용하는 데이터에 적합한 오류 데이터 측정 기준을 한번 설정한 이후 반복적인 이벤트 발생을 통해 입력된 데이터의 오류를 측정한 결과를 사용자가 얻게 된다. 본 논문에서는 개발한 서비스가 과연 얼마나 오류 데이터를 정제하는 데 효과적인지 보이기 위해 CRM에서 ERP로 총 데이터 이벤트를 200회 발생 시켰다.

위의 (그림 13)의 x축은 CRM에서 ERP로 입력되는 데이터의 순서에 따라 일련의 번호를 매긴 것이고 y축은 각 데이터의 오류 발생률을 나타낸 것이다. 그리고 앞의 막대는 탐지만 한 후에 오류 발생률을 측정한 것이고, 그 다음은 정제한 후에 다시 오류 발생률을 비교한 것이다.

위의 그림을 보면 첫 데이터인 엘리먼트 1번과 엘리먼트 5번은 'Order'와 'ShipDetails'라는 XML문서의 계층 구조를 표현하기 위한 형식 데이터일 뿐 데이터 값을 가지지 않으므로 전혀 오류 규칙이 적용되지 않았다. 엘리먼트 2의 경우 CRM 시스템에서 ERP 시스템으로 구매 주문을 보냄에 있어서 가장 중요한 OrderID로서 본 서비스를 사용하여 규칙 2, 3, 5에 의해 오류 데이터가 탐지되고 오류 데이터의 피드백을 통한 데이터 정제가 잘 이루어졌음을 알 수 있다.

다시 말해 'OrderID'는 25%의 오류를 보이던 것이 본 서비스를 사용하여 정제한 후 10%로 감소된 결과를 보인다. 반면 입력된 데이터 중에서 4(Email), 7(Address), 11(Product), 13(Price), 그리고 14(BillAmount)번 데이터들은 상대적으로 낮은 정제 율을 보였다. 그 이유는 4와 11, 13 데이터들은 탐지 규칙 11과 21에 의해 오류로 판정되고 정제되어야 하는데, 오류로 판단은 되었으나 정제 되어야 할 올바른 데이터를 알 수 없는 경우에 해당한다. 예를 들어 'Email'의 경우 'tifosiafiorano.com'의 경우 'string@string(.string)'에 위배되어 오류로는 탐지 된다. 이 데이터를 잘 아는 경우 'tifosi@fiorano.com'이라고 재입력 할 수 있지만 확신할 수 없기 때문에 바로 정제할 수 없는 것이다.

따라서 Null을 검사하거나 값의 범위를 검사하는 규칙 1, 2, 3, 5와 같은 경우에는 정제 율이 높지만 규칙 11과 21과 같이 오류로 탐지 하긴 하였지만 이를 정제할 옳은 데이터가 무엇인지 확신할 수 없는 경우는 낮은 정제 율을 보였다.



(그림 14) 전체 데이터의 오류 탐지 규칙에 따른 오류 데이터 발생률

다음 (그림 14)은 이러한 현상을 규칙 별로 나타낸 것이다.

전체적으로 초기에 입력된 전체 데이터의 오류 발생률과 서비스 적용 후 정제된 데이터 비율은 18.08%에서 12.31%로 감소된 결과를 보였다. 이는 정제 비율로만 보았을 때, 오류 데이터를 약 31.91% 감소시킨 결과이다. 더 높은 정제율을 보일 수 없는 것은 이 서비스가 서비스 간 상호 작용 데이터에 국한된 것이며 자연어 처리 기법 혹은 상업적인 도구를 고용하고 있지 않다는 데 있다.

이렇게 SOA를 기반으로 서비스를 작성하고 통합하여 새로운 서비스를 만드는 경우, 본 논문에서 개발한 서비스간의 상호 작용하는 데이터의 오류 탐지 및 정제 서비스를 이용한다면, 오류 데이터 정제를 이미 다 수집된 데이터베이스로 넘겨 오류 데이터를 쓸모없는 데이터로 만들 것이 아니라 사용자에게 피드백을 줌으로써 최대한 유효한 데이터로 만들 수 있다는 것을 보였다. 또한 개발된 CleanS는 기존의 정제도구가 가지고 있는 구현 플랫폼과 언어에 종속적이고 특정 애플리케이션을 위해 개발되어 확장성이 결핍되어 있는 문제를 다음과 같이 해결할 수 있음을 보였다.

- SOA 기반의 독립적인 서비스로 개발되어 플랫폼과 구현 언어에 독립적이다.
- XML 기반의 규칙 명세 방법은 개발자가 추가하고자 하는 규칙을 서비스의 인터페이스에 영향을 주지 않고 추가 정의하는 것이 가능하게 하였다. 따라서 정제 되어야 할 데이터의 특성에 의존성을 줄일 수 있다.
- 서비스의 입출력 데이터의 종류와 형식에 상관없이 XML 스트림을 파싱하여 사용하므로 선행되어야 할 서비스와 후속 서비스의 종류에 상관없이 재사용 가능하다.

6. 결 론

기존의 데이터 품질 관리에 대한 연구는 시스템이 이미 사용한 데이터가 저장되어 있는 데이터베이스 내에 데이터를 대상으로 하고 있다면, 본 논문에서 제안한 CleanS 서비스는 각 서비스들이 상호 작용하여 데이터가 전송되는 단계에서 데이터의 품질 관리를 수행한다. 이를 통하여 대규모 시스템간의 통합 과정에서 안전한 데이터 사용과 정제된 데이터의 사용으로 시스템 품질 향상을 도모할 수 있다. 또한 데이터의 오류를 실시간으로 탐지하고, 개발된 오류 데이터 정제 서비스는 e-business 시스템과 같이 상호 작용이 빈번한 서비스의 통합 시 데이터의 품질관리를 위해 유용하게 사용될 수 있으며 XML을 이용한 탐지 규칙 설정으로 서비스 사용자의 데이터 제약 조건 설정에 융통성을 제공할 수 있는 SOA를 기반으로 하는 오류 데이터 탐지 및 정제 서비스를 개발하였다. 개발된 오류 데이터 탐지 서비스는 SOA를 기반으로 CRM 시스템과 ERP 서비스를 통합하는데 적용하여 두 서비스간의 상호 작용하는 데이터의 오류를 정제하는 실험을 통해 데이터의 오류를 30% 넘게 줄이는 효과를 보였다.

현재, 오류 데이터의 정제 율을 더 높이기 위해 오류 탐지 시 특별한 도구를 고용하고 오류 제약조건 설정의 인간 개입을 최소화하기 위해 e-Business 트랜잭션을 위한 데이터 온톨로지 구축 및 학습 알고리즘에 관한 연구가 진행 중이다.

참 고 문 헌

[1] P. Krogdahl, G. Luef, and C. Steindl, “Service-Oriented Agility: An initial analysis for the Use of Agile methods for SOA development,” In Proceedings of the 2005 IEEE International Conference on Service Computing(SCC ’05). Vol.2, pp.93-100, July, 2005.

[2] 이경하, 이규철, “웹 서비스의 표준화 동향과 발전 방향”, 한국정보과학회 데이터베이스 연구회지, 제19권 제1호, pp.80-87, March, 2003.

[3] M. P. Papazoglou and D. Georgakopoulos, “Service-Oriented Computing,” Communication of the ACM, Vol.46, No.10, pp.25-28, Oct., 2003.

[4] 지은미, 최병주, 이정원, “SOA에서의 오류 데이터 정제 서비스 개발”, 정보처리학회 2007년도 춘계학술발표대회 논문집(상) 우수논문, 제14권 제1호, pp.649-652, 2007.

[5] Theodore Johnson, and Tamraparni Dasu, “Data Quality and Data Cleaning,” Tutorials of 10th SIGKDD, Aug., 2004.

[6] T. Dasu, T. Johnson, S. Muthukrishnan, V. Shkapenyuk, “Mining Data Structure; Or, How to Build a Data Quality Browser,” In Proceedings of SIGMOD Conf., pp. 240-251, 2002.

[7] M. Hernandez and S. Stolfo, “Real-world data is dirty: data cleansing and the merge/purge problem,” Data Mining and Knowledge Discovery, Vol.2, No.1, pp.9-37, 1998.

[8] M. Lee, H Lu, T Ling, and Y. Ko., “Cleansing Data for Mining and Warehousing,” In Proceedings of 10th DEXA, 1999.

[9] M. Hernandez, R. Miller, and L. Hass, “Schema Mappings as Query Discovery,” In Proceedings of Intl. Conf. VLDB, 2001.

[10] M. M. Breunig, H.-P. Kriegel, R. Ng, J. Sander, “LOF: Identifying Density-Based Local Outliers,” In Proceedings

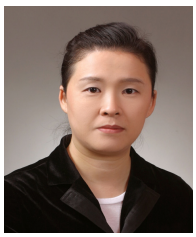
of SIGMOD Conf., 2000.
[11] MonArch, www.00db.co.kr
[12] SLAAM, www.slaam.co.kr
[13] ZipIt, www.sujiewon.co.kr
[14] The AscentialTM Enterprise Integration Suite, www.ascential.com.
[15] HummingBird, www.hummingbird.com
[16] Ortiz Jr., Sixto; "Getting on Board the Enterprise Service Bus," Published by the IEEE computer Society, pp.15-17, 2007.
[17] Won Kim, Byoung-Ju Choi, Eui-Kyeoung Hong, Soo-Kyoung Kim, Doheon Lee, "A Taxonomy of Dirty Data," The Data Mining and Knowledge Discovery Journal, Vol.7 No.1, pp.81-99, 2003.
[18] J. W. Lee, E. Y. Moon, and B. J. Choi, "Data cleansing for Service-Oriented Architecture," Springer-Verlag, Lecture Notes in Computer Science Vol 3590, pp.87-97, 2005.
[19] G. Shankaranarayanan and Y. Cai, "A Web Services Application for the Data Quality Management in the B2B Networked Environment," In Proceedings of 38th Hawaii International Conference on System Sciences, IEEE, 2005.

지 은 미

e-mail : jjiem@ewhain.net
2005년 상명대학교 컴퓨터소프트웨어공학과 (학사)
2006년~현재 이화여자대학교 대학원 컴퓨터학과 석사과정
관심분야 : 소프트웨어공학, SOA, 소트프웨어 테스트, 소프트웨어 프로세스

최 병 주

e-mail : bjchoi@ewha.ac.kr
1983년 이화여자대학교 수학과 졸업(학사)
1988년 Purdue Univ. 전산학과(석사)
1990년 Purdue Univ. 전산학과(박사)
1995년~현재 이화여자대학교 컴퓨터학과 교수
관심분야 : 소프트웨어공학, 소프트웨어 테스트, 소프트에웨어 및 데이터 품질 측정, 소프트웨어 프로세스, 임베디드 시스템 테스팅, SOA



이 정 원

e-mail : jungwony@ajou.ac.kr
1993년 이화여자대학교 전자계산학과 (학사)
1995년 이화여자대학교 대학원 전자계산학과 (석사)
1995년~1997년 LG종합기술원 주임연구원
1997년~2003년 이화여자대학교 대학원 컴퓨터학과(공학박사)
2003년~2006년 이화여자대학교 컴퓨터학과 BK교수, 전임강사 (대우)
2006년~현재 아주대학교 정보통신대학 전자공학부 조교수
관심분야 : SOA, 유비쿼터스컴퓨팅, 임베디드 소프트웨어