서비스 지향 아키텍처에서의 효율적인 데이터 프레임 워크 구축

노홍찬O 여윤구 이동현 박상현
연세대학교 컴퓨터과학과
{fallsmalO, yyk, ldh, sanghyun}@cs.yonsei.ac.kr

An Efficient Data Framework for Service Oriented Architecture

Hongchan Roh0 Yun-gu Yeo Donghyun Lee Sanghyun Park
Department of Computer Science, Yonsei University

요 약

최근 서비스 지향 아키텍처를 적용한 많은 시스템들이 개발되어 왔고 그에 따라 서비스 지향 아키텍처에 대한 관심도 증가하고 있다. 하지만 여태껏 연구되어왔던 주제들은 전체 아키텍처의 효율적인 구성 및 개발 프로세스 구축에 초점을 맞춘 반면, 데이터 프레임워크를 보다 효율적으로 비즈니스 프레임워크와 분리하는 데 초점을 맞춘 연구는 부족하다. 이에 우리는 실 서비스되고 있는 대용량 서버 관리 서비스를 서비스 지향 아키텍처로 전환하고 그 중 데이터 프레임워크를 보다 추상화시키고 비즈니스 프레임워크와 느슨한 결합을 이룰 수 있도록 하는 연구를 진행하였다. 본 논문에서 제안하는 데이터 프레임워크는 실시간 데이터 전송에 유리한 트리 구조의 데이터를 사용자가 정의한 규칙을 기반으로 비즈니스프레임워크의 다양한 비즈니스 로직에서 효율적으로 활용할 수 있는 다양한 형태의데이터로 바꿀 수 있는 계층을 제공한다. 또한 이러한 계층에 대해 실 서비스에서 활용할 수 있는 정의된 제약조건 하에서 실험을 진행하였고 그 결과 만족스러운 성능을 보이는 것으로 나타났다.1)

1. 서 론

최근의 급변하는 비즈니스 환경은 적응적인 서비스 구조를 요구한다. 기존의 하나의 서비스를 구성하는 구성요소들이 수직적으로 통합되는 구조는 이러한 급변하는 비즈니스 환경에 적합하지 못한다. 수직적인 통합구조에 있어서는 각각의 구성요소들이 서로 간에 매우 의존적으로 설계되어 있으므로 새로운 요구사항이 발생할 시에 전체 서비스 구조에 대한 대대적인 수정이 필요하기 때문이다. 그러므로 보다 적응적인 서비스 구성이 필요하며 이를 위해 각각의 구성요소들이 수평적으로 통합될 수 있는 구조가 필요하다.

서비스 지향 아키텍처 (SOA: Service-Oriented Architecture)는 이러한 급변하는 비즈니스 환경에서 여러 모듈들을 수평적으로 통합할 수 있는 구조이다. 기존의 하나의 통합된 단위의 프로그램을 재사용 가능한 단위의 모듈로 분리하고 각각의 모듈이 연결되는 타 계층의 모듈과의 의존성을 제거하여 해당 모듈 외에 그 모듈을 대체할 수 있는 해당 계층의 다양한 다른 모듈들과의 호환성을 갖도록 각 계층 간의 인터페이스(interface)를 보다 추상화하여 설계하는 방법이다[1].

이러한 서비스 지향 아키텍처는 1996 가트너(Gartner) 그룹에 의해 처음 소개되었고 OASIS 그룹의 정의에 의하면 서로 다른 도메인(domain)에 있는 분산된 비즈니스 로직들을 조직화하고 그를 이용하여 새로운 서비스를 만들어 내고 또한 그러한 서비스들을 모아 다른 새로운 서비스를 창조해낼 수 있는 패러다임(paradigm)이라고 요약될 수 있다. 이러한 서비스 지향 아키텍처는 느슨한 결합성(loose coupling), 상태 부재성 (statelessness), 조립성(composability), 자치성(autonomy), 재활용성(Reusability), 준수성(Contract), 발견용이성(Discoverability)의 일곱 가지 성질을 만족시켜야만 한다. 첫째, 느슨한 결합성은 서로 다른 계층의 모듈 간의 의존성이 낮아야 한다는 것을 뜻하며, 둘째, 상태부재성은 특정 모듈의 수행에 따른 상태 정보가 그 모듈과 연결되는 타 모듈에 영향을 미치지 않아야 한다는 것이다. 셋째, 조립성은 목적하는 서비스 구성을 위해 개별 모듈들이 잘 조립되어지고 조정되어질 수 있어야 한다는 것이다. 넷째, 자치성은 서비스가 내재된 모든 모듈들에 대한 완전한 제어권을 가져야 한다는 것을 의미한다. 다섯째, 재활용성은 각각의 모듈들이 해당 모듈에 현재 연결된 타 계층의 모듈들과의 호환성만 지니는 것이 아니라 필요에 따라 앞으로 달리 연결될 수 있는 다른 모듈들과의 호환성도 가져야 한다는 의미를 내포하고 있다. 여섯째, 준수성은 각 계층 간의 통신을 위해 정의된 프로토콜을 준수해야 한다는 것을 뜻한다. 마지막으로 발견용이성은 특정 모듈이 새로운 서비스 구축에 재활용될 수 있도록 외부에서 검색이 가능하고 인지가 가능하도록 설계되어야 한다는 의미이다[2].

이러한 서비스 지향 아키텍처에 대한 연구는 1996년 가트너가 처음 소개한 이후로 SODA[3], SOAD[4], SOUP[5], SOMA[6] 등 SOA에 대한 다양한 정의와 개발 프로세스가 제안되어 왔지만 현재까지 기술적으로 적용

1) 본 연구는 중소기업청의 2007년도 산학협력실 지원사업(S5107A13101)의 지원을 받아 수행되었습니다.

할 수 있을 만큼의 수준으로는 구체화되지 못하고 있는 실정이다. 이러한 SOA 설계 방법론의 구체화와 관련된 연구로서 SOA 프레임웍 아키텍처[1]가 있으며 이 논문은 SOA 실질적인 어플리케이션 개발에 적용될 수 있도록 SOA의 계층을 보다 세분하고 각 계층 별을 여러 모듈로 구체화시켜 SOA관한 새로운 프레임워크(framework)를 제시하였다. 또한 [7]은 논문 및 특허에 대한 정보 검색을 수행하는 구체적인 SOA 서비스 개발 예를 제시하였다.

본 논문에서는 [1]이 제시한 데이터 프레임워크를 보다 세분화 하여 데이터 프레임워크가 보다 추상화되고 비즈니스 프레임워크와 느슨하게 연결될 수 있도록 하는 새로운 계층을 제안한다. 이 계층은 실시간 데이터 전송을 위해 최적으로 구성된 트리형태의 데이터를 미리 사용자가 정의한 규칙에 의해 2차원 매트릭스 형태의 데이터로 전환 시키며 이 과정에서 [5]가 제안한 SOA에서의 XML의 유용성을 고려하여 사용자 정의 규칙 및 데이터의 변환 결과를 모두 XML형태로 구성하였다. 또한 제안한 계층을 기존 데이터 프레임워크에 적용하여 실제 서비스에서 제공되는 실 데이터를 처리하는 성능을 측정하여 서비스의 제약조건을 만족하는지 확인하였다.

본 논문의 구성은 다음과 같다. 2장에서는 [1]이 제시한 세분화된 SOA의 프레임워크에 대해서 좀 더 자세히 알아보고, 3장에서 본 논문에서 제시한 하는 데이터 프레임워크의 새로운 계층에 대해 자세히 기술한다. 4장에서는 제시된 계층이 목표로 하는 서비스의 제약조건을 만족하는지 확인한 실험 결과를 제시하고 마지막으로 5장에서 결론을 맺는다.

2. SOA 프레임워크 아키텍처

[1]이 제안한 SOA 통합 프레임워크은 그림 1에서 알 수 있듯이 크게 4가지 계층으로 구분된다. 첫째, UI 프레임워크로서 전체 서비스 어플리케이션의 사용자 인터페이스를 담당한다. 그 구체적인 예로서 사용자의 권한관리, 외부 인터페이스 레이아웃 관리, 인증 서비스 등이 있다. 둘째, Interface 프레임워크로서 기존의 여러 서비스들을 조합하여 목표하는 새로운 서비스를 만들어내는 역할을 한다. 또한 각 서비스들 간의 실행 순서 제어, 정의된 프로토콜에 적합한 데이터 전달 등의 역할을 맡는다. 셋째, 비즈니스 프레임워크로서 단위 기능을 처리하는 여러 비즈니스 로직 모듈들의 집합들로 구성되어 있다. 각각의 집합은 하나의 서비스에 대응되고 이러한 다수의 서비스들이 하나의 비즈니스 프레임워크를 이룬다. 각각의 서비스들은 해당 서비스에서 정의한 규칙을 통해 소속된 비즈니스 로직들에 대한 실행순서를 제어한다. 넷째, 데이터 프레임워크는 비즈니스 프레임워크 계층에 추상화된 데이터 스키마(data schema)와 표준 질의 언어를 제공하여 쉽게 데이터에 접근하고 조작할 수 있도록 하는 역할을 맡는다.

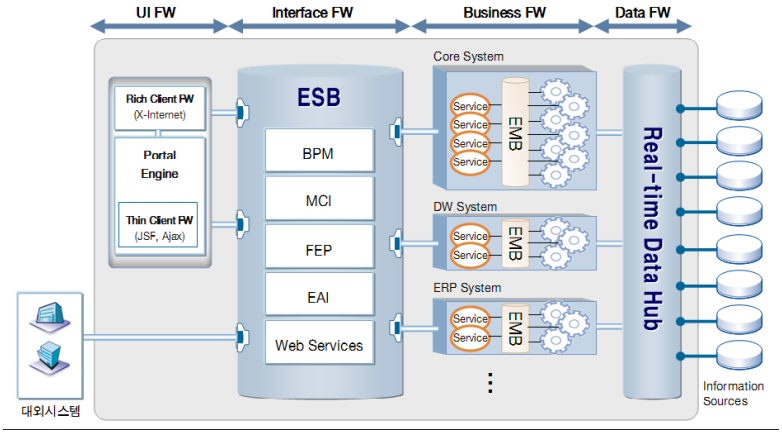

그림 1. SOA 통합 프레임워크

3. 제안하는 새로운 데이터 관리 계층

[1]이 제안한 데이터 프레임워크는 비록 데이터 구조를 추상화하여 상위 비즈니스 로직과의 의존성을 낮추긴 하였지만 정의된 하나의 데이터 스키마를 사용함으로서 다양한 비즈니스 로직에서의 효율적인 처리를 수행하는 데에 어려움을 겪을 수 있다. 이러한 문제점을 해결하기 위해 본 논문에서는 각 비즈니스 모델에서 보다 다루기 쉬운 형태로 데이터를 사용자 정의 규칙에 의해 변환하는 계층을 데이터 프레임워크에 추가할 것을 제안한다.

이를 위해 실제 서비스되고 있는 (주) 아이엔 소프트의 Open manager에서 사용되는 트리 형태로 추상화된 데이터를 기반으로 새로운 데이터 관리 계층의 구조 및 관련 알고리즘들에 대해 제안하고 보다 구체적으로 실제 구현 시의 이슈들에 대해 살펴보려고 한다. Open manager는 실시간으로 서버들을 원격 관리해주는 서비스이다. Open manager가 관리하는 서버들의 상태가 트리형태의 메시지로 Open manager에 도달하게 되고 Open manager는 이러한 서버의 상태를 체크해서 필요한 조치를 원격으로 해당 서버들에 취하게 된다. 이러한 Open manager의 서비스를 SOA기반으로 구현하려는 것이 최종 목표이며 본 논문은 그 중간 단계로서 데이터 프레임워크에 대한 효율적인 구조를 제시하려 하는 것이다. Open manager의 메시지 형태는 다음과 같다.

```
message CODE
{
  string code;
  integer success;
}
message WEBSTATUS
{
    string datetime;
    integer attempt;
    integer fail;
    message CODE codes[]
  }
```

그림 2. Open manager의 메시지 형태

그림 2에서 알 수 있듯이 Open manager의 메시지는 여러 개의 필드들로 이루어져 있으며 하나의 필드로서 다른 메시지의 배열이 내재될 수 있는 구조로 트리 형태로 구성될 수 있다. 이러한 구조는 서비스 특성 상 실시간으로 많은 데이터가 생성되고 전송되어야 하기 때문에 보다 압축적으로 데이터를 구성하고 전송해야하기 때문이다[2]. 이러한 환경에 있어 데이터를 압축적으로 구성하고 전송해야 하므로, 데이터 외의 태그 문자열이 부가적으로 들어갈 수 있는 XML이나, 데이터가 중복되어 여러 릴레이션에 포함될 수 있는 관계형 모델은 적합하지 않다.

하지만 서버로부터 데이터 전송이 끝난 후 Open manager가 데이터를 수집하게 되면 데이터를 트리 형태로 유지하는 것은 상위 비즈니스 로직의 처리 과정에 트리 형태의 데이터를 비즈니스 로직이 다루기 편한 데이터 형태로 변환시키는 부하를 줄 수 있으므로 비효율적이게 된다. 그러므로 이러한 트리 형태의 데이터를 상위 비즈니스 로직들이 처리하기 쉽도록 보다 유연한 형태로 바꿀 필요성이 있다.

이를 위해 본 논문은 트리 형태의 데이터구조를 미리 SOA 시스템에 의해 정의된 규칙에 의해 보다 조작이 편리한 2차원 행렬 형태로 변환할 수 있는 계층을 제안한다. 또한 간단한 데이터 조작은 트리 형태의 데이터를2차원 행렬 형태의 데이터로 변경하는 과정에서 수행할 수 있도록 미리 정의된 함수를 통해 데이터 조작을 할 수 있도록 하였고 그 외에 메시지에서 하나의 필드를 키 값으로 하여 전체 메시지에 대한 간단한 통계 데이터를 산출해낼 수 있는 기능을 추가하였다

표1은 미리 정의된 함수들을 기능 별로 분류하여 보여준다.

표 1. 미리 정의된 함수들

함수분류	함수명
문자열 함수	substr, strcat, int_to_str, float_to_str
계산 함수	add, sub, mul, div
통계 함수	sum, average

시스템 정의 규칙은 키 값을 기준으로 하여 통계처리를 하는 규칙과 그렇지 않은 일반 규칙으로 나뉘는데 그 문법은 문맥 자유 문법에 의해 다음과 같이 정의된다

$$[\text{TransformRule}] \rightarrow [\text{rule}] \mid [\text{agg-rule}] \quad (1)$$

식 (1)은 모든 규칙(TransformRule)이 일반 규칙(rule)과 통계처리 규칙(agg-rule)로 구분되어 둘 중에 하나만 적용될 수 있음을 나타낸다.

$$\begin{aligned} &\text{rule} \rightarrow [\text{columnGroup}] \mid [\text{coumnGoup}]\ \text{rule} \\ &\text{columnGroup} \rightarrow [\text{column}] \mid [\text{column-array}] \\ &\text{column} \rightarrow [\text{columnData}] \mid \\ &\text{column-array} \rightarrow [\text{columnData}] \mid [\text{columnData}]\ \text{column-array} \end{aligned} \quad (2)$$

식 (2)는 일반 규칙의 문법을 정의한다 크게 일반 규칙은 일반 컬럼(column)과 컬럼-배열(column-array)로 변환(유도)될 수 있으며 컬럼과 컬럼-배열의 차이는 다음과 같다. 일반 컬럼 하나는 기존 메시지의 하나의 필드에 대응될 수 있다. 즉 2차원 행렬 상의 하나의 컬럼의 이름과 그 컬럼의 값은 대응하는 메시지의 필드의 이름과 값과 동일하다. 하지만 해당 필드가 그림 2에서의 예처럼 타 메시지의 배열일 경우 2차원 행렬로 해당 메시지를 변환하는데 문제가 생긴다 그림 2의 예에서 WEBSTATUS 메시지의 datetime이라는 필드는 WEBSTATUS 메시지 하나 당 하나의 값을 가지게 되지만 codes 필드는 배열의 원소마다 다른 code값과 success 값을 가지게 된다. 이에 따라 WEBSTATUS의 메시지가 한 개가 존재하고 codes 배열의 크기를 n이라 가정하면 WEBSTATUS 메시지와 CODE 메시지 간에 1:n 관계가 형성되고, 변환 결과인 2차원 행렬을 구하기 위해서는 관계형 모델의 조인 연산에 해당하는 작업이 수행되어야 한다. 즉, 결과로 생성되는 2차원 행렬은 총 n개의 행을 가지게 되며 WEBSTATUS의 datetime, attempt, fail 필드 값들이 n번 중복되어 2차원 행렬의 n개의 행에 쓰여야 하는 것이다. 이러한 중복된 데이터의 생성을 막기 위해 컬럼배열을 도입하였으며 컬럼배열을 이용하면 내재된 메시지의 특정 필드 값을 레이블로 하여 자동으로 새로운 컬럼들을 생성하고2차원 행렬에 단 하나의 행만을 추가하게 된다 예를 들어 codes의 배열의 크기가 3일 때의 CODE 메시지들의 예제에 따르는 컬럼배열은 표2와 같다.

표 2. CODE 메시지와 컬럼배열 예제

CODE 메시지	{code:ABC, success:0}, {code:BCD, success:1}, {code:CDE, success:0}		
컬럼-배열	code_ABC	code_BCD	code_CDE
컬럼 값	0	1	0

$$\begin{aligned} &\text{columnData} \rightarrow [\text{value-of}] \mid [\text{returnStringFunction}] \\ &\quad \mid [\text{returnNumberFunction}] \end{aligned} \quad (3)$$

2) 트리 형태의 데이터 구조는 실시간으로 많은 데이터가 생성되고 전송되는 SOA기반 서비스들의 데이터 형태로 적용될 수 있으므로 새로운 데이터 프레임워크 구축에 있어 좋은 예제이다.

returnStringFunction → [TEXT] | [substr] | [strcat] | [int_to_str] | [float_to_str]

returnNumberFunction → [NUMBER] | [add] | [sub] | [mul] | [div] | [sum] | [average] | [str_to_int] | [str_to_float] (4)

식 (3)은 각 컬럼에 해당하는 값이 어떻게 계산될 수 있는지를 정의한 규칙이다. 첫 번째 항목인 value-of는 단순히 value-of의 속성으로 입력되는 해당 메시지의 필드의 값을 가져오는 것이고 returnStringFunction과 returnNumberFunction의 경우 표 1에 정의되어 있고 식 (4)로부터 연결되는 문자열 함수 계산 함수 각각의 결과 값으로부터 해당 컬럼의 값을 구성하라는 의미이다.

add, sub, mul, div → [calculate]
calculate → [operand][operand]
operand → [value-of] | [returnNumberFunction]
sum, average → [sum_aver]
sum_aver → [value-of] | [arrayField]
substr → [value-of] | [returnStringFunction]
strcat → [returnStringFunction][returnStringFunction]$^{+}$ (5)
str_to_int, str_to_float → [str_num]
str_num → [value-of] | [returnStringFunction]
int_to_str, float_to_str → [num_str]
num_str → [value-of] | [returnNumberFunction]

식 (5)는 표 1에서 정의한 함수들에 대한 문법을 정의한다. 함수들은 재귀적으로 오른쪽 operand를 우선 순위로 하여 수행된다.

agg-rule → [key-column][agg-columnGroup]+
agg-columnGroup → [agg-column] | [agg-column-array] (6)
agg-column → [columnData]+
agg-column-array → [columnData]+

식 (6)은 식 (2)와 매우 유사하지만 통계처리 규칙을 정의하고 있다. 통계처리 규칙은 메시지 중 하나의 필드를 키(key)로 정하고 해당 키 값이 같은 메시지들에 대해서는 각각의 컬럼이나 컬럼-배열의 값들을 정의된 통계 함수에 의해 집계해 나가는 방식이다. 예를 들어 WEBSTATUS 메시지의 datetime필드를 키 컬럼으로 정했을 경우 동일한 datetime을 가지는 메시지들은 식 (6)에 의해서 지정된 agg-column들의 값들이 지정된 통계 함수(sum, average)등에 의해서 집계를 하여 결국 생성되는 2차원 행렬에 하나의 datetime값 당 하나의 행만 생성되게 된다.

보다 유연한 규칙 생성을 위해 이러한 규칙에 대한 문

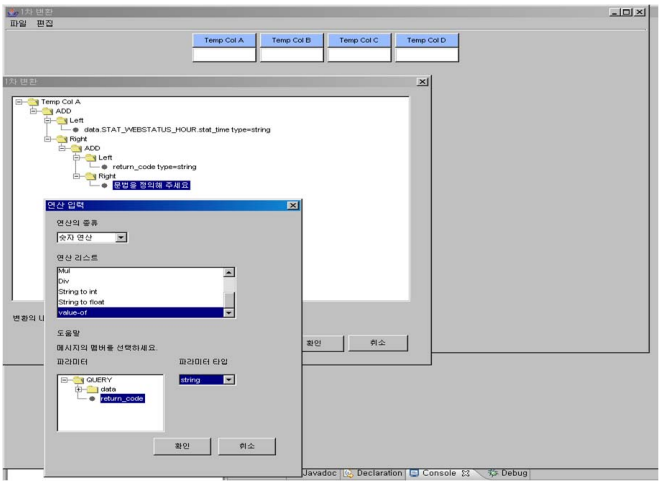

그림 3. 시스템 정의 규칙 관리 GUI 모듈

법들을 XML Schema로 정의하고 시스템 정의 모든 규칙들을 XML 문서로 생성하게 하였다. 또한 사용자가 편리하게 xml 문서를 편집할 수 있도록 그림 3과 같은 GUI 환경에서 시스템 정의 규칙을 관리하는 모듈을 제작하였다. 그림 상단에 나열된 박스들이 규칙에 참여하는 각 컬럼들을 나타내고 각 컬럼에 대한 값을 그 밑의 팝업창에서 트리 형태로 정의되는 세부 규칙에 의해 생성하게 된다.

이렇게 정의된 규칙들을 통해 원래의 메시지를 2차원 행렬로 변환하는 방법은 알고리즘1과 같다.

첫 째, 주어진 데이터를 다른 메시지를 내재하고 있는 메시지와 다른 메시지에 내재된 메시지들을 부모 노드와 내재된 메시지 배열 크기만큼의 자식 노드들로 표현하여 트리구조로 도식화한다.
둘 째, 각 메시지 별로 메시지에 대응되는 노드가 포함하는 리프 노드의 숫자를 계산하여 그 수만큼의 연속된 행들에 그 메시지의 일반 필드(메시지 타입이 아닌 경우)들에 대응하는 열에 동일한 값을 반복하여 기록한다.

알고리즘 1. 메시지 변환 알고리즘

그림 4와 그림 5는 알고리즘 1의 예제를 보여주고 있다. 그림 4는 그림 3의 메시지 구조에 새로운 메시지를 추가하여 변경한 것이고, 그림 5는 그림 4의 메시지 구조를 가지는 데이터의 구성 예를 알고리즘1의 첫 번째 단계에서 설명한 트리 형태로 도식화한 것을 보여준다. 이는 하나의 WEBSTATUS 메시지가 3의 크기를 가지는 CODE 메시지 배열을 내재하고, 해당 배열에 속하는 각각의 CODE 메시지들 CODE[0], CODE[1], CODE[2]는 각각 3, 2, 1 크기의 STATUS 메시지 배열을 내재하고 있는 실제 데이터의 예제를 나타내고 있다. 결국 WEBSTATUS 메시지 하나가 총 3개의 CODE 메시지를 내재하고 각각의 CODE 메시지에 의해 결국 6개의 STATUS 메시지를 내재하게 되는 것이다. 그러므로 이러

한 트리 구조를 2차원 배열로 변환하기 위해서는 총6개의 행이 필요하며 각각의 메시지에서 총 7개의 일반 필드에 대응하는 7개의 열(datetime, attempt, fail, code, success, prev_state, current_state)들이 필요하게 된다. 가장 상위의 메시지인 WEBSTATUS의 필드 정보들(datetime, attempt, fail)은 단 하나의 값만 존재하므로 6개의 행에 같은 값이 반복되어 기록되게 된다. 또한 CODE[0] 메시지의 경우 3개의 STATUS 메시지를 포함하고 있으므로 CODE[0]의 일반 필드들(code, success)을 3개의 행의 대응하는 열들에 자신의 값들을 동일하게 복사하게 된다. 또한 CODE[1]의 일반 필드들도 마찬가지로 CODE[0]가 쓰인 이후의 2개의 행의 대응하는 열들에 자신의 값들을 복사하게 된다. 마지막으로 CODE[2]의 나머지 필드들의 경우에도 1개의 행에 자신의 값을 쓰게 된다. STATUS의 일반 필드들의 경우에는 중복되어 쓰일 필요가 없으므로 메시지 각각의 일반 필드 값을 각각의 행에 자신의 열 위치에 기록하게 된다

```
message STATUS
{
  string prev_state;
  string current_state;
}
message CODE
{
  string code;
  integer success;
  message STATUS status[]
}
message WEBSTATUS
{
    string datetime;
    integer attempt;
    integer success;
    message CODE codes[]
}
```

그림 4. Open manager의 메시지 구조 예제

4. 새로운 데이터 관리 계층의 성능 측정

이러한 새로운 데이터 프레임워크 계층을 현재 개발중인 SOA기반의 서버관리 시스템에 적용하기 위해 해당 시스템에서 요구하는 성능 제약 조건을 만족하는지 실험을 진행하였다. 실시간 데이터이므로 단위 시간 안에 얼마만큼의 데이터를 처리할 수 있는지가 중요한 요건이었고 이를 위해 1초당 1000개 이상의 메시지를 최대 3메가 바이트 용량의 메모리 공간 하에서 처리 하는 것을 제약 조건으로 정의하였다.

표 3은 제안하는 새로운 데이터 프레임 워크 계층의 성능을 동일한 데이터에 시스템 정의 규칙을 달리 적용하여 측정한 결과이다. 시스템 정의 규칙을 정의할 때 컬럼-배열을 사용하는지의 여부와 통계처리 규칙을 사용하는지 여부에 따라서 성능이 크게 좌우되는 것을 볼 수

컬럼-배열 사용	3,964	STATUS[0]	3,964
통계처리규칙 적용	3,908		28

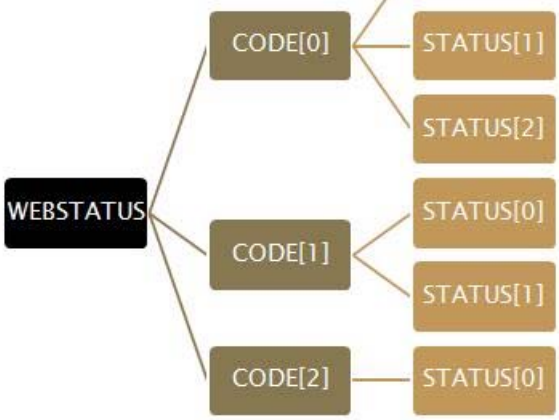

그림 5. 그림 4 메시지의 실제 데이터 예제

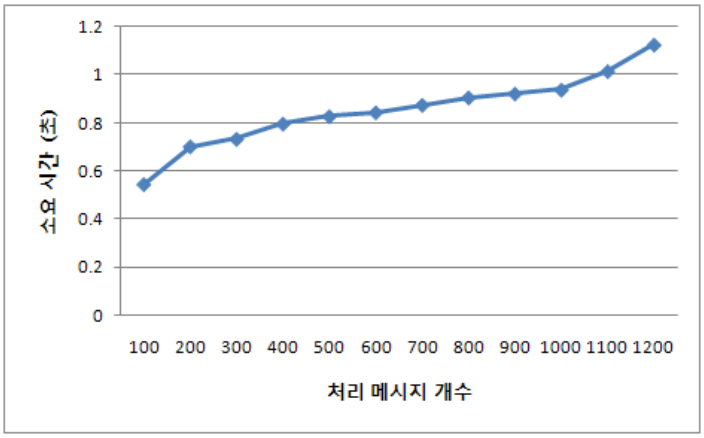

그림 6. 처리 메시지 개수에 따른 소요 시간

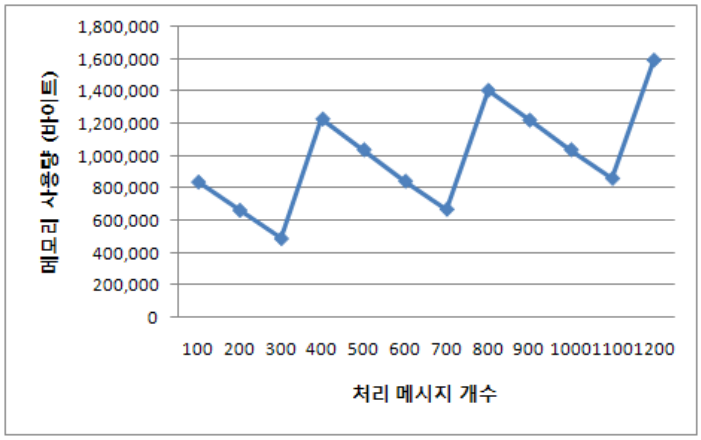

그림 7. 처리 메시지 개수에 따른 메모리 사용량

표 3. 새로운 데이터 프레임워크 계층의 성능

구분	메시지 개수	생성된 행 개수	소요시간(초)	메모리 사용량(바이트)
컬럼-배열 미사용	3,964	31,712	3.328	8,615,112

있다. 그 이유는 컬럼-배열을 사용하는 경우 그냥 컬럼을 사용하는 경우에 비해 트리 구조의 메시지 데이터를 중복된 컬럼 값의 생성 없이 처리할 수 있기 때문이고 그렇게 중복되지 않음으로써 결과적으로 생성되는 행의 개수가 컬럼-배열을 사용할 때 상대적으로 매우 작은 것으로 나타난다. 소요 시간은 행의 개수에 대략적으로 비례하는 결과를 보이며 메모리 사용량의 경우는 행의 개수에 더 밀접하게 비례하는 결과를 보인다.

그림 6은 처리 메시지의 개수 증가에 따라 알고리즘의 처리 속도가 선형적으로 증가하는 것을 보여준다. 또한 그림 7은 같은 조건 하에서 메모리 사용량의 증감을 보여준다. 특이하게 메시지 개수가 증가함에도 메모리 사용량이 감소하는 경향을 구간 별로 보이지만 이는 실제 구현된 프로그램이 자바 언어를 기반으로 개발되었기 때문에 자바 버츄얼 머신(JVM)에 의한 주기적인 가비지 컬렉션의 영향으로 해석될 수 있다. 전반적으로 선형적으로 메모리 사용량이 증가함을 알 수 있다. 그림 6와 그림 7을 통해 많은 메시지 데이터에 있어서도 제안하는 새로운 계층과 관련 알고리즘이 견고하게 동작할 수 있음을 알 수 있다.

이러한 결과를 종합해 볼 때 실 서비스를 위한 제약조건을 제안하는 새로운 데이터 프레임워크 계층이 만족시킨다고 볼 수 있으며 사용자 정의 규칙을 잘 적용하는 것이 성능에 커다란 이점이 있음을 알 수 있다.

5. 결론

본 논문에서는 기존의 서비스 지향 아키텍처를 보다 구체화하여 데이터 프레임워크가 보다 상위 비즈니스 로직과 추상화되어 결합될 수 있도록 새로운 데이터 관리 계층을 추가하였다. 제안하는 데이터 관리 계층은 사용자 정의 규칙을 사용하여 시스템이 원하는 보다 다루기 쉬운 형태로 데이터를 변환할 수 있도록 하였으며 문맥 자유 문법으로 정의한 다양한 규칙을 통해 유연하게 데이터를 조작할 수 있도록 하였다. 이러한 규칙은 XML 형태로 저장되고 GUI 어플리케이션을 통해 생성 및 관리되어 보다 규칙 생성 및 관리에 있어 보다 유연한 구조를 이루었다 할 수 있다.

또한 실 서비스에 요구되는 제약 조건 하에서의 실험을 통해 제안하는 새로운 데이터 관리 계층이 목표로 하는 서비스의 질을 만족시키며 선형적으로 동작하는 알고리즘을 통해 데이터의 크기에 관계없이 견고하게 동작함을 확인하였다.

본 논문에서 제안하는 데이터 관리 계층은 실시간 데이터 전송에 효율적인 데이터 구조인 트리 형태를 2차원 행렬 형태로 변환할 수 있을 뿐만 아니라 XML과 GUI 툴을 기반으로 유연하게 설계된 구조를 바탕으로 다른 형태의 데이터에도 적용할 수 있는 확장성을 지니고 있다. 또한 데이터 프레임워크를 보다 구체화하고 상위 비즈니스 로직으로부터 더욱 추상화한 아이디어는 다른 SOA기반 시스템 설계에 좋은 모델이 될 수 있을 것이다.

참고 문헌

[1] 김성익, 박정일, “SOA 프레임웍 아키텍처,” 정보과학회지, Vol.25, No.1, pp. 27-33, 2007

[2] OASIS, “Reference Model for Service Oriented Architecture 1.0,” 2006

[3] Samir Nigam, "Service Oriented Development of Applications(SODA) in sybase Workspace," Sybase whitepaper, 2005

[4] Olaf Zimmermann, Pal Krogdahl, Clive Gee, "Elements of Service-Oriented Analysis and Design," IBM Journal, 2004

[5] Kunal Mittal, "Service Oriented Unified Process(SOUP)", 2005, http://www.soaconsultant.com/html/soup.shtml

[6] Ali Arsanjani, "Service-Oriented Modeling and Architecture(SOMA)," IBM developerWorks, 2004

[7] 신수민, "SOA기반의 정보시스템 설계 및 구현" 한국콘텐츠학회, Vol.5, No.1, pp. 13-16, 2007