실행시간 프로세스 모니터를 위한 XML 기반의 데이터 저장소의 설계

정 윤 석[†] · 김 태 완[††] · 장 천 현[†††]

요 약

실시간 시스템은 시스템이 적시성을 보장하는지 파악하기 위해 실시간 감시 기법을 이용한다. 실시간 감시의 대상은 내부 시스템만이 아니라 네트워크 상에 존재하는 원격 시스템도 포함된다. 각 시스템에서 발생하는 데이터를 감시하기 위해서는 데이터를 일시적 혹은 장기적으로 저장할 데이터 저장소가 필요하며, 이러한 데이터 저장소는 실시간 감시를 지원할 수 있도록 시간 제약과 데이터 저장소에 대한 접근성을 고려해 설계해야 한다. 이에 따라 본 논문에서는 시간 제약과 접근성을 고려한 XML 기반의 데이터 저장소 및 전송 구조를 제시한다. XML기반의 데이터 저장소는 표준화된 데이터 포맷인 XML을 기반으로 설계하여 TCP/IP 및 HTTP를 지원하는 모든 플랫폼에서 원격으로 데이터 저장소 접근이 가능하며, 별도의 변환과정 없이 데이터를 사용할 수 있다. XML기반의 전송 구조는 DOM, XML-RPC 및 저장 후 전송 기법을 이용하여 데이터 접근 및 전송 시간을 최소화하도록 설계하였다. 더 나아가 본 논문에서는 XML 기반의 데이터 저장소 및 전송구조를 이용하여 실시간 감시를 수행할 때, 기준이 되는 시간적 한계치를 제시하기 위해 측정 실험을 수행하였다. 본 논문에서 설계한 XML 기반의 데이터 저장소 및 전송 구조 그리고 실험 결과는 기본적으로 실시간 감시 및 제어를 필요로 하는 분야 및 응용 분야에서 이용할 수 있다.

Design of a XML-based Data Store Architecture for Run-time Process Monitor

Yoon Seok Jeong[†] · Tae Wan Kim[††] · Chun Hyon Chang[†††]

ABSTRACT

Monitoring is used to see if a real-time system provides a service on time. The target of monitoring is not only an interior system but also a remote system which is located in the remote network. Monitoring needs data store to monitor data from each system. But a data store should be designed on the considerations of time constraints and data accessibility. In this paper, we present an architecture of XML-based data store and network delivery. XML-based data store is based on XML which is a standardized data format. So any platform which supports TCP/IP and HTTP can access data in the data store without any conversion. The XML-based delivery architecture is designed to reduce the time of data access and delivery. In addition, some experiments were tested to provide the timing guideline to be kept by a real-time system which uses the architecture presented in this paper. The architecture of XML-based data store and delivery designed in this paper can be used in the domains of remote real-time monitoring and control.

키워드 : 실시간 감시(Real-time Monitoring), 실행시간 프로세스 모니터(Run-time Process Monitor), XML 기반 데이터 저장소(XML-based Data Store)

1. 서 론

여러 분야에서 실시간 시스템의 요구가 증가하면서 시스템의 동작 상태를 파악할 수 있는 실시간 감시 기능이 요구되고 있으며, 이에 따라 실시간 감시에 관한 많은 연구가 수행되었다. 그러나 기존 실시간 감시의 경우 실시간 프로세스의 수행 능력과 감시 기법에 초점을 맞추었고, 이로 인해 감시 결과를 저장하는 데이터 저장소는 데이터 호환성의 결여, 실시간성의 미고려, 누적 데이터를 위한 저장 구조의 부재 등의 문제를 안게 되었다.

이러한 데이터 저장소의 문제를 해결하기 위해 본 논문에서는 XML(eXtensible Markup Language) 기반의 데이터 저장소 구조를 제시한다. 이 구조는 XML을 기반으로 하여 분산환경 및 웹 환경에서도 다른 변환 과정 없이도 데이터를 사용하는 호환성을 제공하도록 설계하였다. 더불어 프로세스의 현재 상태 데이터 외에도 로그 데이터를 저장할 수 있는 데이터 저장소 구조를 설계하여 누적 데이터 저장 구조의 부재 문제를 해결하였고, XML 기반의 전송 구조 및 전송 기법을 제안하여 시간 조건을 준수할 수 있도록 하였다. 더 나아가 본 논문에서 제시한 저장소 및 전송 구조를 기반으로 시간 측정 실험을 수행하여 이 구조들이 지원할 수 있는 시간적 한계치를 제공하였다.

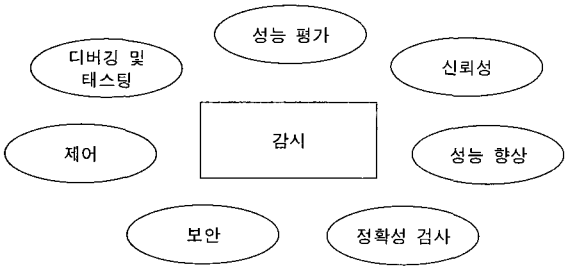
※ 본 연구는 2002년도 건국대학교 학술진흥연구비 지원 및 대학 IT연구센터 육성·지원사업의 연구결과로 수행되었음.
† 준 회 원 : 건국대학교 대학원 컴퓨터공학과
†† 정 회 원 : 건국대학교 대학원 컴퓨터공학과
††† 종신회원 : 건국대학교 컴퓨터공학과 교수
논문접수 : 2003년 7월 19일, 심사완료 : 2003년 11월 20일

본 논문의 2장에서는 기존 실시간 감시 기능의 분석과 문제점을 도출하고, 3장에서는 실시간 감시의 데이터 저장소 개선을 위한 방안으로써 XML 기반의 데이터 저장소, XML-RPC 기반의 전송 구조 및 기법을 제안한다. 4장에서는 제시한 구조를 기반으로 시간 측정 실험을 수행하고, 실험 구조 및 결과를 제시한다. 마지막 5장 결론에서는 본 논문의 성과 및 향후 연구 방향에 대해서 기술한다.

2. 관련 연구

2.1 기존 감시 기능의 분석

실시간 시스템이 산업 전반에 보급되면서 실시간 시스템의 수행을 감시하는 실시간 감시 기능에 대한 다수의 연구가 수행되었다. 예를 들어 프로세스 차원의 감시, 시스템 차원의 감시, 하드웨어 차원의 감시, 네트워크 차원의 감시에 관한 연구가 이루어졌다[3-5, 9]. 이들 연구를 종합해 볼 때 감시란 '데이터를 수집하고, 데이터를 통하여 각각의 시간 별로 어플리케이션 및 운영체제가 어떤 작업을 수행했는지를 파악하는 것'이며, 감시의 사용 목적 별로 (그림 1)과 같이 일곱 가지로 나눌 수 있다[2, 4, 11, 13].



(그림 1) 감시의 기본 사용 목적

감시 기능은 모니터가 담당한다. 모니터는 시스템의 상태를 감시하고, 이를 분석함으로 비정상적 행위를 찾아내고 이를 해결하는 소프트웨어이다[10]. 실시간 시스템에서 사용하는 모니터는 기본적으로 실시간 프로세스의 수행을 감시하도록 설계된다[4-6, 9, 12]. 프로세스의 감시 결과는 상태 데이터의 형태로 데이터 기억장소에 저장된다. [6, 7]의 연구에서는 MCB(Method Control Block)에 프로세스의 상태정보를 유지하도록 설계하였으며, [10]의 연구에서는 MIB (Monitor Information Base)에 유지하도록 설계하였다. [4, 5]의 연구에서는 데이터 저장소에 상태정보를 유지하도록 설계하였다.

2.2 기존 감시 기능의 문제점

기존 연구에서는 주로 실시간 프로세스의 수행 및 감시 기법에 초점을 맞추었으며, 그 결과, 감시 결과인 프로세스 상태 데이터를 저장하는 저장소와 관련된 연구는 부족하다. 분산 환경을 고려한 감시와 관련한 연구에서도 데이터 호환성이나 시간 특성을 고려하지 않고 데이터 저장소를 설계하였으며[10], 결과적으로 데이터 저장소는 다음과 같은 한계를 안고 있다.

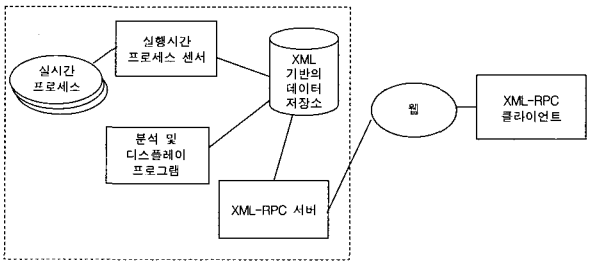
- 데이터 호환성의 결여 : 프로세스의 상태 데이터는 특정 플랫폼이나 특정 어플리케이션에 종속적이다. 이로 인해 외부 시스템이나 이종의 플랫폼에서는 관련 데이터를 이용할 수 없다[4, 5, 13].
- 시간 제약조건을 고려치 않음 : 실시간 시스템에서 프로세스는 시간 제약을 준수해야 한다. 하지만 실시간 프로세스 수행 이후 감시 데이터가 데이터 저장되고 이용되는 과정에서는 시간 제약을 고려하지 않고 있다.
- 누적 데이터 저장 구조의 부재 : 기존의 실시간 감시는 현재 상태 데이터 만을 고려한다. 이로 인해 결과 분석 시 필요한 누적 데이터를 유지할 수 있는 저장 구조가 없다.

3. 실시간 감시의 데이터 저장소 개선을 위한 방안

2장에서는 기존의 감시 기능과 데이터 저장소가 안고 있는 문제점을 분석하였다. 본 장에서는 이러한 문제점을 극복하기 위해, 데이터 호환성 및 시간 제약 조건을 고려한 데이터 저장소 구조 및 관련 기법을 제시한다.

3.1 전체 시스템 구조

(그림 2)는 실시간 감시를 지원하는 시스템의 전체 구조를 보여준다. 실시간 감시는 실행시간 프로세스 모니터에 의해 내부·외부 시스템에서 실시간 프로세스를 대상으로 수행된다.



(그림 2) 실시간 감시를 지원하는 시스템의 구조

실행시간 프로세스 모니터는 실행시간 프로세스 센서, 데이터 저장소, 분석 및 디스플레이 프로그램으로 구성된다. 각 구성 요소의 기능은 다음과 같다.

- 실행시간 프로세스 센서 : 프로세스 내에 코드 형태로 삽입되어 실시간 프로세스로부터 상태 데이터를 수집한다.
- 데이터 저장소 : 실시간 프로세스의 상태 데이터를 저장하며 관리한다. 데이터 저장소는 특정 어플리케이션이나 시스템에 종속되지 않고 독립적으로 존재하며, 저장된 데이터는 원격으로 접근할 수 있다.
- 분석 및 디스플레이 프로그램 : 데이터 저장소로부터 데

이터를 읽거나 분석한 후 이를 출력한다.

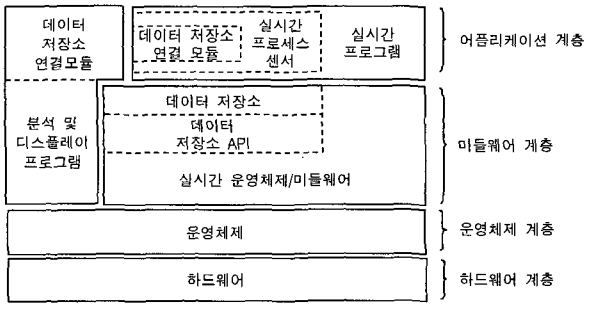
데이터 저장소에 저장된 데이터는 전송 구조를 통해서 외부 시스템으로 전송된다. 전송 구조는 XML-RPC 서버와 XML-RPC 클라이언트로 구성되며 각 요소의 기본 기능은 다음과 같다.

- XML-RPC 서버 : 외부 시스템(XML-RPC 클라이언트)의 요청을 받아 데이터를 전송한다.
- XML-RPC 클라이언트 : 원격의 데이터 저장소에 있는 데이터를 획득하기 위해 XML-RPC 서버에 데이터 전송 요청을 한다. XML-RPC 클라이언트 모듈은 데이터 접근을 위한 기본 모듈이므로 다양한 어플리케이션 기능과 연동될 수 있다.

3.2 시스템 체계

(그림 3)은 실시간 감시 시스템의 체계를 보여준다. 시스템 체계는 크게 4개의 계층으로 구성된다.

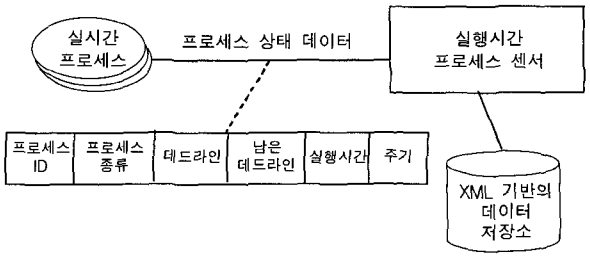
- 어플리케이션 계층 : 운영체제 상에서 동작하는 일반 어플리케이션과 실시간 프로그래밍을 지원하는 실시간 어플리케이션을 지원한다.
- 미들웨어 계층 : 실시간 운영체제 혹은 실시간 미들웨어를 통해 실시간 프로세스의 동작에 필요한 기본 기능을 제공한다[8]. 또한 관련 API들을 제공한다.
- 운영체제 계층 : 디바이스 제어, 스케줄링, 멀티 쓰레딩 등의 운영체제의 기본 기능을 제공한다.
- 하드웨어 계층 : 시스템이 동작하기 위한 디바이스를 제공한다.



(그림 3) 실시간 감시 시스템의 체계

3.3 실시간 프로세스 데이터 추출

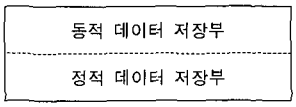
실시간 프로세스 모니터는 실시간 시스템에서 동작하는 실시간 프로세스를 감시하며, 프로세스로부터 상태 정보를 획득한다. (그림 4)는 프로세스의 상태 정보를 획득하는 과정을 보여준다. 실시간 프로세스는 반복적으로 수행되며, 매 주기마다 자신의 상태 정보를 발생시킨다. 상태 데이터는 프로세스 ID, 프로세스 종류, 데드라인, 남은 데드라인 시간, 실행시간, 주기이다.



(그림 4) 실시간 프로세스 데이터 획득 과정

3.4 XML 기반의 데이터 저장소 구조

데이터 저장소는 일시적으로 저장되는 데이터와 지속적으로 저장되는 데이터를 저장할 수 있도록 설계하였다. (그림 5)는 두 가지 속성의 데이터를 저장할 수 있는 데이터 저장소의 구조를 보여준다. 동적 데이터 저장부는 프로세스 상태 데이터를 일시적으로 저장하며, 새롭게 데이터를 수집할 경우 기존의 데이터는 새로운 데이터로 대체한다. 정적 데이터 저장부는 히스토리 데이터를 저장하기 위한 공간으로써 누적된 데이터와 이를 통계적으로 분석한 데이터를 저장한다.

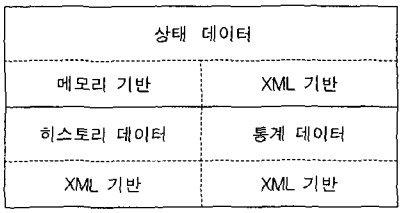


(그림 5) 두 가지 속성 데이터를 저장하기 위한 데이터 저장소 구조

데이터 저장소는 다음을 고려하여 설계한다.

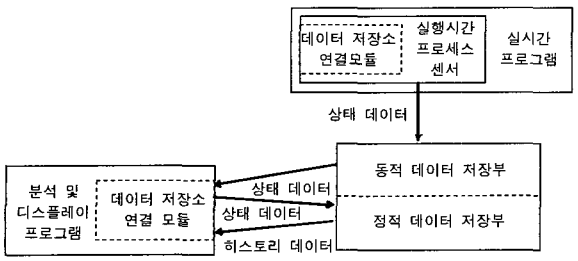
- 표준을 따르는 저장 형태를 유지할 것
- 저장된 데이터는 이종의 플랫폼이나 웹에서 이용할 수 있을 것

위의 사항을 고려하여 데이터 저장소를 XML기반으로 설계하였다. XML을 이용할 때 TCP/IP와 HTTP를 지원하는 외부 시스템에서는 별도의 변환 과정 없이 데이터 저장소에 저장된 상태 데이터를 이용할 수 있다는 장점이 있다[1]. 또한 XML 기반의 전송구조를 이용하여 웹 및 분산환경에서도 사용할 수 있다.



(그림 6) XML 기반의 데이터 저장소 구조

(그림 7)은 데이터 저장소를 중심으로 상태 데이터가 동적 데이터 저장부 및 정적 데이터 저장부에 저장되는 과정을 보여준다.

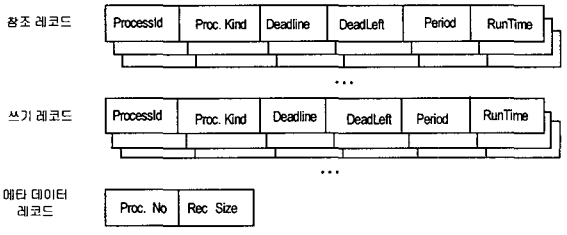


(그림 7) 데이터 저장소를 중심으로 한 데이터 흐름

실행시간 프로세스 모니터는 실시간 프로그램 내에 삽입되어 실시간 프로세스의 동작을 감시하며 프로세스 상태 데이터를 추출한다. 실행시간 프로세스 센서는 데이터 추출 기능만을 수행하여 실시간 프로세스 동작에 미치는 영향을 최소화한다. 프로세스 상태 데이터는 데이터 저장소의 동적 데이터 저장부에 저장되고, 디스플레이 프로그램은 이 데이터를 읽어 출력한다. 정적 데이터 저장부에 저장되는 데이터들은 동적 데이터 저장부에서 추출된 데이터들로서, XML 문서 형태로 변환 저장되기 때문에 실시간 시스템에서는 시간적 낭비 요소가 될 수 있다. 따라서 XML 문서 변환 및 저장은 분석 및 디스플레이 프로그램이라는 별도의 어플리케이션이 담당한다. 이를 통해 XML 문서변환 및 저장이 실시간 프로세스의 동작이나 센서가 데이터를 추출하는데 미치는 영향을 최소화하였다.

데이터 저장소가 독립적으로 존재하는 구조의 장점을 요약하면 첫째, 데이터를 발생시키는 실시간 프로세스에 미치는 영향을 최소화할 수 있으며, 둘째, 데이터 저장소에 저장된 데이터 역시 어플리케이션 독립적이라는 것이다.

3.5 동적 데이터 저장부의 설계



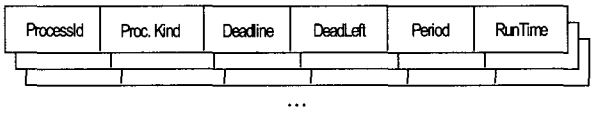
(그림 8) 동적 데이터 저장부의 구조

동적 데이터 저장부는 프로세스의 상태 데이터를 일시적으로 저장한다. 저장되는 데이터는 프로세스 ID, 실행시간, 잔여 실행시간, 실행주기, 데드라인 등의 프로세스 정보로 구성된다. 이들 데이터는 실행시간 프로세스 모니터나 기타 어플리케이션이 접근하여 읽고 쓰게 된다. (그림 8)은 참조 레코드, 쓰기 레코드, 메타데이터 레코드로 구성된 동적 데이터 저장부의 구조를 보여준다. 참조 레코드와 쓰기 레코드는 데이터 중복구조를 갖는다. 이는 실시간 프로세스가 데이터를 쓰는 동안 다른 어플리케이션이 데이터를 읽을 때 발생하는 데이터 무결성 문제를 해결하기 위해서이다.

3.6 정적 데이터 저장부의 설계

(그림 9)는 정적 데이터 저장부에 저장되는 히스토리 데이터의 구조를 보여준다. 히스토리 데이터는 프로세스 상태 데이터의 누적이므로 동일한 구조를 갖지만 물리적으로는 XML 파일 형태로 저장되게 된다. (그림 9)의 상태 데이터를 나타내는 각 요소는 state 요소에 속하며 state 요소는 id, kind, deadline, deadlineleft, period 및 runtime으로 구성된다. 여기서 각 프로세스를 구분하기 위해서 id 요소를 이용한다.

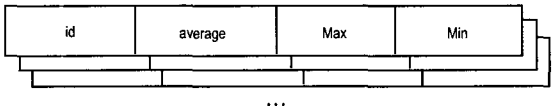
XML Schema 문서는 정적 데이터 저장부 구조의 구현을 보여준다. 대상이 되는 프로세스의 수에 따라 복수의 <process> 요소가 생성되게 된다.




```
<xs : schema  xmlns : xs = http://www.w3.org/2001/XMLSchema
 elementFormDefault = "qualified">
<xs : element name = "process">
<xs : complexType>
<xs : element ref = "state" minOccurs = "0" maxOccurs
                = "unbounded"/>
<xs : sequence>
          <xs : element ref = "id" />
          <xs : element ref = "kind" />
          <xs : element ref = "deadline" />
          <xs : element ref = "deadlineleft" />
          <xs : element ref = "period" />
<xs : element ref = "runtime" />
</xs : sequence>
</xs : element>
</xs : complexType>
</xs : element>
<xs : element name = "id" type = "xs : integer"/>
<xs : element name = "kind" type = "xs : integer"/>
<xs : element name = "deadline" type = "xs : integer"/>
<xs : element name = "deadlineleft" type = "xs : integer"/>
<xs : element name = "period" type = "xs : integer"/>
<xs : element name = "runtime" type = "xs : integer"/>
</xs : schema>
```

(그림 9) 히스토리 데이터 구조

(그림 10)은 통계 정보를 저장하기 위한 통계 데이터 구조와 XML Schema를 보여준다. id는 프로세스를 구별하는 일련번호이며 average, max, min, count는 각각 대상 상태 값의 통계 평균값, 최대값, 최소값, 상태 값의 모집단 개수를 의미한다.




```
<xs : schema  xmlns : xs = http://www.w3.org/2001/XMLSchema
  elementFormDefault = "qualified">
<xs : element name = "history">
<xs : complexType>
```

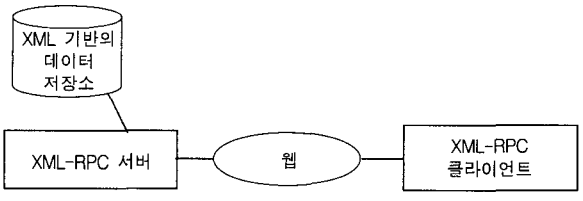
```
<xs : element ref = "statistics" minOccurs = "0" maxOccurs
                = "unbounded"/>
<xs : sequence>
<xs : element ref = "id" />
<xs : element ref = "average" />
<xs : element ref = "max" />
<xs : element ref = "min" />
<xs : element ref = "count" />
</xs : sequence>
</xs : element>
</xs : complexType>
</xs : element>
<xs : element name = "id" type = "xs : integer"/>
<xs : element name = "average" type = "xs : float"/>
<xs : element name = "max" type = "xs : integer"/>
<xs : element name = "min" type = "xs : integer"/>
<xs : element name = "count" type = "xs : integer"/>
</xs : schema>
```

(그림 10) 통계 데이터 구조

3.7 전송 구조

원격 프로시저를 호출하는 방법에는 CORBA, DCOM, SOAP, XML-RPC 및 기타 프로토콜 등이 존재하며, 각 프로토콜은 각각 장단점을 가지고 있다[7].

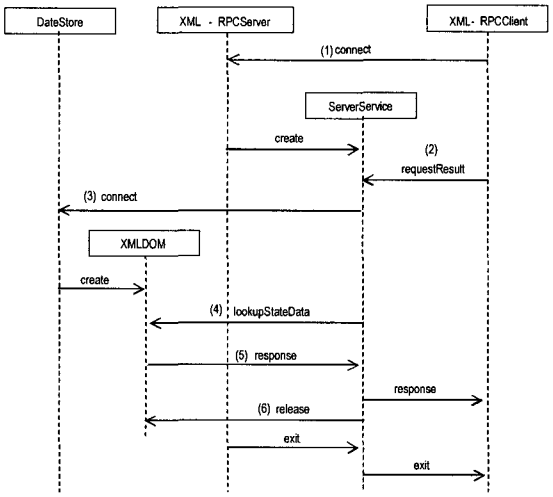
본 논문에서는 플랫폼 독립성, 사용 용이성을 고려하여 XML-RPC 기반의 원격 프로시저 호출 방법을 사용한다. XML-RPC는 HTTP를 통한 간단하고 이식성 높은 원격 프로시저 호출방법이다. XML-RPC는 Perl, Java, Python, C, C++등의 여러 언어를 지원하며, 유닉스, 리눅스, 윈도우 플랫폼에서 동작한다. (그림 11)은 XML-RPC 기반의 전송 구조를 보여준다. XML-RPC 서버는 XML-RPC 클라이언트의 요청에 따라 요구하는 데이터를 XML 기반의 데이터 저장소에서 찾아 제공하는 역할을 한다.



(그림 11) XML 기반의 전송 구조

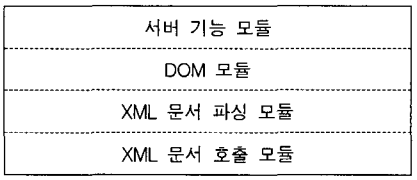
(그림 12)는 XML 기반의 전송 절차에 대해서 보여준다.

① XML-RPCClient는 서버의 URL을 통해 XML-RPCServer에 접속한다. XML-RPCServer는 ServerService를 생성하여 XML-RPCClient에게 서비스를 제공하도록 한다.
② XML-RPCClient는 ServerService에게 자신이 필요로 하는 서비스를 요청한다.
③ ServerService는 DataStore에 접속하고, DataStore는 프로세스 상태 데이터를 가지고 있는 XML 문서를 호출하고 파싱한 후 DOM을 생성한다.
④ ServerService는 XMLDOM에게 자신이 필요로 하는 프로세스 상태 데이터를 요청한다.
⑤ XMLDOM은 ServerService가 요청한 상태 데이터를 반환하고, ServerService는 XMLDOM 객체를 해제한다.
⑥ 서비스가 끝나면 XML-RPCServer는 ServerServer를 종료하고 XML-RPCClient를 종료한다.



(그림 12) XML 기반의 전송 절차

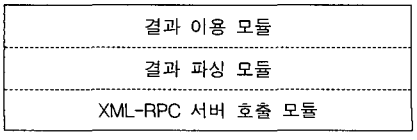
상기의 전송 절차를 고려할 때 XML-RPC 서버는 (그림 13)과 같은 구조를 갖도록 설계해야 한다.



(그림 13) XML-RPC 서버의 구조

- XML 문서 호출 모듈 : 정적 데이터 저장부에 저장된 XML 문서를 읽어들이는 모듈
- XML 문서 파싱 모듈 : XML 문서 호출 모듈을 통해 읽어 들인 XML 문서를 파싱하여 객체형태로 변환하는 파싱 모듈
- DOM 모듈 : 읽어 들인 XML 문서의 내용을 객체 모델로 변환하여 API를 이용해 접근·관리할 수 있도록 하는 모듈
- 서버 기능 모듈 : XML-RPC 클라이언트의 서비스 요청을 받아들여 결과를 반환하는 기능을 하는 모듈

XML-RPC 클라이언트는 서비스를 요청하고 요청 결과를 반환 받는 (그림 14)와 같은 구조를 갖는다.



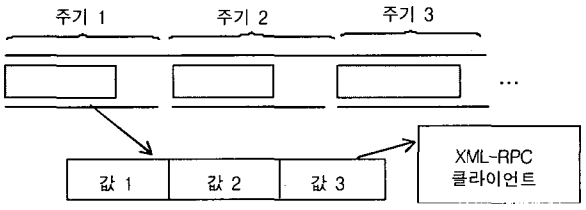
(그림 14) XML-RPC 클라이언트 구조

- XML-RPC 서버 호출 모듈 : 정적 데이터 저장부에 저장된 XML 문서를 읽어 들이는 모듈
- 결과 파싱 모듈 : XML 문서 호출 모듈을 통해 읽어들인 XML 문서를 파싱하여 객체형태로 변환하는 파싱 모듈
- 결과 이용 모듈 : 상태 데이터를 읽어 들인 후 이를 이용하여 다른 기능을 수행하는 모듈

3.8 전송 기법

XML 기반의 데이터 전송은 전송 효율성 및 데이터 호환성을 보장한다. 그러나 웹 기반의 실시간 감시는 필연적으로 네트워크 상의 데이터 전송 지연이 발생한다. 따라서 이를 해결하기 위한 전송 메커니즘이 필요하다. 본 논문에서는 저장 후 전송 기법을 제안한다.

저장 후 전송 기법은 사용자가 요구하는 일정 수준의 시간 제약을 준수하기 위해서 사용한다. 예를 들어 사용자가 원격에서 실시간 감시를 하고자 할 때 실시간 프로세가 100ms 주기마다 동작을 하고 네트워크 상의 지연이 1초라면, 10개의 상태 데이터 중 9개는 상실하게 된다. 이러한 데이터 상실을 막기 위해서 저장 후 전송 기법을 사용한다.



(그림 15) 저장 후 전송 기법의 수행 과정

n개의 실시간 프로세스의 실행주기의 집합을 $P = \{p_1, p_2, \cdots, p_n\}$이라 하고, d_{ps}를 전송 데이터 크기에 따른 네트워크 상에 발생하는 전송 시간이라면, 시간 윈도우 크기 ws는 다음과 같다.

$$ws = \frac{d_{ps}}{Min(P)}$$

d_{ps}는 실험을 통해서 도출한다. 결과적으로 한 번 데이터 전송 시 몇 번의 주기 데이터를 누적 시킬 것인지는 ws에 의해 결정된다. 이를 알고리즘으로 정리하면 다음과 같다.

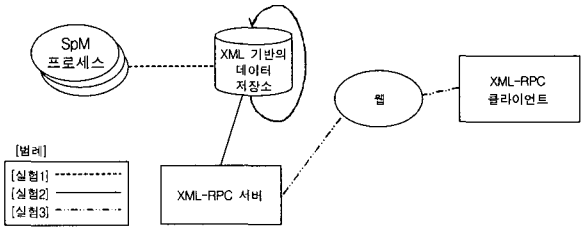
```
{
  최소주기 = 0 ;
  각 프로세스에 대해,
  {
    주기 값(pi)를 파악한다.
    if ( 최소주기 < pi )
    최소주기 = pi ;
  }

  전송 데이터 크기에 따른 네트워크 상에 전송 시간(dps)을 계산한다.
  윈도우 크기(ws)를 계산한다 :
     ws = dps / 최소주기 ;
  ws 만큼의 주기를 누적한 데이터를 전송한다.
}
```

(그림 16) 저장 후 전송 알고리즘

4. 실 험

4.1 실험 구조

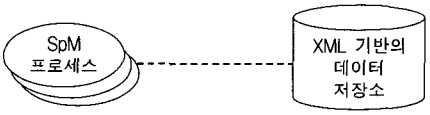
전체 실험 구조는 (그림 17)과 같다. 본 논문에서는 실시간 프로세스의 실행에서부터 원격지에서 프로세스 상태 데이터를 감시하는 것까지 전체 감시 과정을 세 부분으로 분리하여 각 부분에 대한 실험을 수행하였다. 실험의 목적은 데이터 저장소와 관련된 각 부분에서 소요되는 시간을 파악하여 실시간 감시를 위한 세밀한 시간적 한계치를 제시하기 위해서이다.



(그림 17) 전체 실험 구조

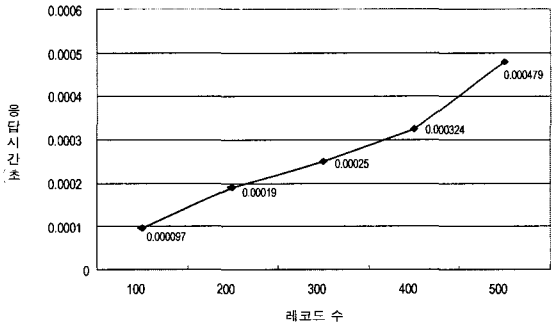
4.2 실험 1 : 실시간 프로세스 수 증가에 따른 상태 데이터 저장 시간 변화 측정 실험

본 실험에서는 실시간 프로세스 수의 증가에 따라 데이터 저장소에 상태 데이터가 저장되는 시간의 변화를 측정한다. 실시간 프로세스 수의 증가는 곧 입력 레코드의 증가임을 가정하고 입력하는 상태 데이터 레코드의 수를 증가시켰다. (그림 18)은 본 실험의 구조를 보여준다.



(그림 18) 실시간 프로세스 수 변화에 따른 상태 데이터 저장 시간 측정 실험 구조

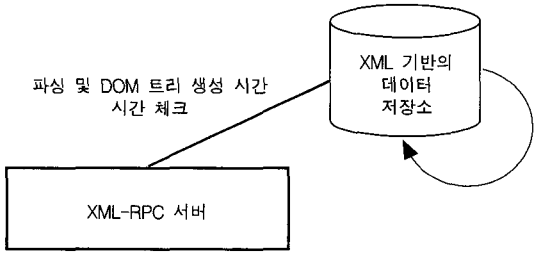
(그림 19)는 본 실험을 통해 도출된 결과이다. 입력 레코드의 수가 일정하게 증가할 때 응답시간 역시 일정한 증가를 보이는 것을 알 수 있다.



(그림 19) 실험 1 결과 그래프

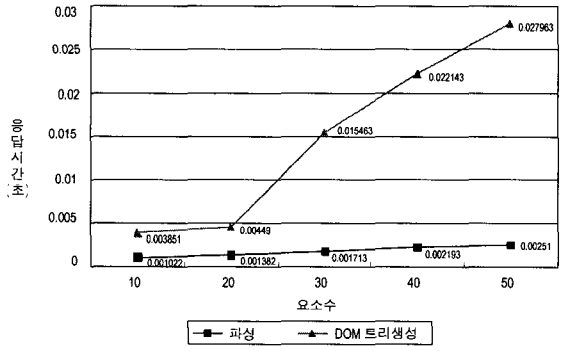
4.3 실험 2 : 데이터 저장소와 XML-RPC 서버간 데이터 입출력 시간 측정 실험

본 실험은 XML 기반의 데이터 저장소에 저장된 XML 파일 내의 상태 데이터를 XML-RPC 서버가 읽는 시간을 측정한다. 이때 측정하는 시간은 두 가지로 나눌 수 있다. XML 파일을 파서가 파싱하는 파싱시간과 DOM 트리를 생성하여 읽는 DOM 트리 생성 및 읽기시간이다. 본 실험은 데이터 저장소와 XML-RPC 서버 구간 내에서 이루어지며 (그림 20)은 본 실험의 구조를 보여준다.



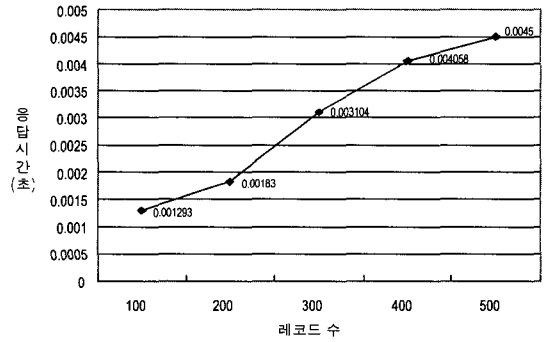
(그림 20) 데이터 저장소와 XML-RPC 서버간 데이터 입출력 시간 측정 실험 구조

(그림 21)은 본 실험의 결과이다. XML 파일 내의 요소수가 일정하게 증가하면 파싱시간과 DOM 트리 생성 및 읽기 시간도 증가하는 것을 확인할 수 있다.



(그림 21) 실험 2-1 결과 그래프

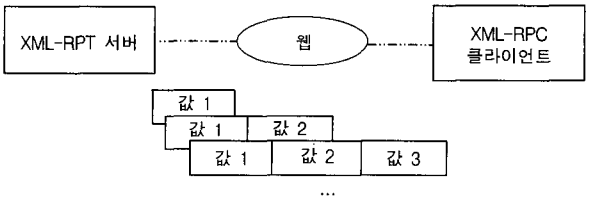
(그림 22)는 XML 변환 과정을 거치지 않고, 단순히 데이터 저장소에서 데이터를 읽는 시간을 측정한 실험결과이다. 추출하는 데이터 레코드의 수를 증가시킬 때 응답시간이 일정하게 증가하는 것을 볼 수 있다. 실험 2-1은 XML 문서를 구성하는 요소단위로 실험을 하였고, 실험 2-2는 레코드 단위로 실험을 하였기 때문에 결과를 단순 비교하기에는 다소 어려움이 있지만 XML 변환 과정을 거치는 경우는 그렇지 않은 경우보다 응답시간이 더 늘어나는 것을 알 수 있다. 이러한 결과는 실시간 시스템에서 처리 시간 증가라는 문제를 야기할 수 있다. 이 문제를 해결하기 위해서는, 앞에서도 설명한 것처럼, 독립적인 어플리케이션이 XML 문서 변환을 처리함으로써 증가된 응답 시간을 보정할 수 있다. 결과적으로 시간적인 측면에서 XML 변환 과정이 실시간 프로세스의 동작에 미치는 영향을 최소화하면서도 XML 문서 변환이 주는 장점을 취할 수 있다.



(그림 22) 실험 2-2 결과 그래프

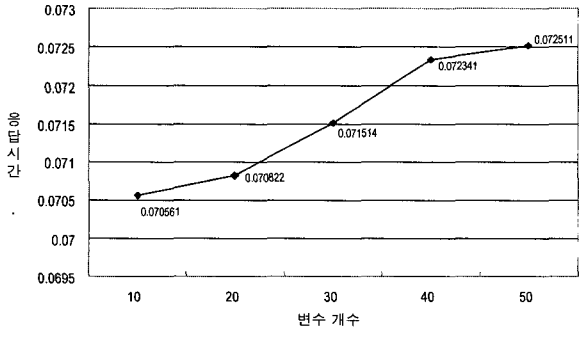
4.4 실험 3 : 전송 데이터 크기 변화에 따른 응답시간 측정 실험

(그림 23)는 본 실험의 실험 구조를 보여준다. 본 실험은 XML-RPC 서버와 XML-RPC 클라이언트 사이에서 데이터 전송시 걸리는 응답시간을 측정한다. 실시간 감시를 위해서는 이러한 어느 정도의 네트워크 상의 전송 지연이 발생하는지 파악해야 한다. 본 실험은 윈도우 사이즈(실시간 프로세스의 주기의 누적) 변화에 따라 전송 지연 시간의 변화를 측정한다. 윈도우 사이즈 증가는 전송되는 데이터의 증가이기 때문에 정수형 변수의 수를 증가시키는 것으로 대체하였다.



(그림 23) 전송 데이터 크기 변화에 따른 응답시간 측정 실험 구조

(그림 24)은 본 실험의 결과를 보여준다. XML-RPC 서버와 XML-RPC 클라이언트 간에 전송되는 데이터 양이 일정하게 증가할 때 네트워크 지연 시간도 일정하게 증가하는 것을 알 수 있다.



(그림 24) 실험 3 결과 그래프

4.5 실험의 결론

전체 실험을 통해 각 구간에서 이동하는 데이터의 증가는 응답시간을 증가시키는 것을 알 수 있다. 또한 실험 3의 결과를 이용해 본 논문에서 제시한 저장 후 전송 기법에서 필요한 전송 데이터 크기에 따라 네트워크 상에 발생하는 지연 시간을 파악하여 시간 윈도우 사이즈를 구할 수 있다.

〈표 1〉 실험 3의 결과치

변수 개수 xi	응답시간 yi
10	0.070561
20	0.070822
30	0.071514
40	0.072341
50	0.072511

<표 1>의 결과치를 이용해 다음의 회귀 분석 모델을 구축할 수 있다.

$$\hat{y} = 0.069924 + 0.000054x$$

상기의 회귀 분석 모델을 통해 시간 윈도우 크기를 구할 수 있다. x를 변수 개수(데이터 크기)라 할 때 시간 윈도우 크기는 회귀 분석 모델을 통해 d_{ps}를 구할 수 있으므로 시간 윈도우 크기는 다음과 같이 구할 수 있다.

$$ws = \frac{0.069924 + 0.000054x}{Min(P)}$$

5. 결론 및 향후 방향

본 논문에서는 기존의 실시간 감시 체계가 가지고 있던 데이터 저장소의 문제점을 해결하기 위해 XML 기반의 데이터 저장소 구조를 제안하였다. 이 데이터 저장소는 프로세스의 상태 데이터 뿐 아니라 히스토리 데이터를 저장할 수 있는 XML 기반의 데이터 저장소 구조로 설계되었다. XML 기반이기 때문에 데이터의 어플리케이션 종속성을 줄이고 분산환경 및 웹 환경에서도 별도의 변환작업이 필요없는 데이터 호환성을 제공한다. 또한 본 논문에서는 XML 기반의 전송 기법을 제안하여, 실시간 감시에서도 시간 조건을 보장하는 방안을 제시하였다. 또한 데이터 저장소를 중심으로 데이터의 입력, 전송과 관련한 실험을 수행하여 시간적 한계치를 제시하였다.

향후 작업으로는 첫째, 특정 실시간 프로그래밍 모델(TMO 모델 등)을 이용해서 본 논문에서 제시한 XML 기반의 데이터 저장소를 구현할 예정이며, 둘째, 데이터 저장소에 히스토리 데이터를 이용하여 실행시간 동안에 분석할 수 있는 동적 분석 기능을 추가할 예정이다.

참 고 문 헌

[1] World Wide Web Consortium, http://www.w3c.org.

[2] "How To Enable Process Accounting on Linux," http://kldp.org.

[3] Eric Kidd, "XML-RPC HOWTO," http://kldp.org/HOWTO/html/XML-RPC-HOWTO/.

[4] 정윤석, 김태완, 장천현, "실행시간 프로세스 모니터를 위한 구조 설계", 정보처리학회 춘계 학술발표논문집, 제10권 제1호, 2003.

[5] Yoon Seok Jeong, Tae Wan Kim, Chun Hyon Chang, "Design and Implementation of a Run-time TMO Monitor on LTMOS," Proc. Embedded Systems and Applications, Jun., 2003.

[6] B. J. Min, et al., Implementation of a Run-time Monitor for TMO Programs on Windows NT, IEEE Computer, Jun., 2000.

[7] Kim, J. G. and Cho, S. Y., "LTMOS : An Execution engine for TMO-Based Real-Time Distributed Objects," Proc. PDPTA'00 LasVegas, Vol.V, pp.2713-2718, Jun., 2000.

[8] S. H. Park, "LTMOS(LinuxTMO System)'s Manual," HU FS, Mar., 2000.

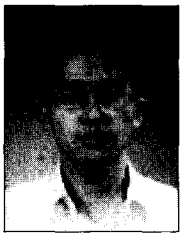
[9] A. K. Mok and G. Liu, "Efficient Run-Time Monitoring of Timing Constraints," Proc. Real-Time Technology and Application, Jun., 1997.

[10] Hyung-Taek Lim, et al., "Monitor based Fault Management in a Distributed Environment," 1995.

[11] B. A. Schroeder, "On-line Monitoring : A Tutorial," IEEE Computer, June, 1995.

[12] S. Sankar and M. Mandal, "Concurrent Runtime Monitoring of Formally Specified Programs," IEEE Computer, Mar., 1993.

[13] Mike Loukides, "System Performance Tuning," O'Reilly, 1990.



정 윤 석

e-mail : ysjeong@cse.konkuk.ac.kr

2000년 건국대학교 컴퓨터공학과(공학석사)

2000년~현재 건국대학교 컴퓨터공학과 박사과정

관심분야 : 실시간 시스템, 시스템 모니터링, 객체지향 프로그래밍, XML

김 태 완

e-mail : twkim@konkuk.ac.kr

1994년 건국대학교 전자계산학과(공학사)

1996년 건국대학교 전자계산학과(공학석사)

1996년~2001년 현대중공업 기전연구소 연구원

1996년~현재 건국대학교 컴퓨터공학과 박사과정

2003년~현재 경민대학 인터넷비즈니스과 겸임교수

관심분야 : 프로그래밍 언어, 실시간 프로그래밍, 자동화 소프트웨어, 산업기기 감시 진단 제어 시스템

장 천 현

e-mail : chchang@konkuk.ac.kr

1977년 서울대학교 계산통계(학사)

1979년 KAIST 전산학(석사)

1985년 KAIST 전산학(박사)

현재 건국대학교 컴퓨터공학과 정교수

관심분야 : 프로그래밍 언어, 컴파일러, 실시간 시스템