nML 프로그래밍 환경에서 SOAP 클라이언트 구현

(Implementation of SOAP Client in nML Programming Environment)

권 오 경 † 한 태 숙 ††
(Ohkyoung Kwon) (Taisook Han)

요 약 웹 서비스에 대한 구현이 많이 늘고 있다. 웹 서비스는 각 프로그래밍 언어에서 XML를 이용해서 원격 호출을 한다. 이때 사용하는 대표적인 프로토콜이 SOAP으로써 본 논문에서는 nML 프로그래밍 환경에서 SOAP[1] 클라이언트 구현을 제시한다. nML은 한국과학기술원 ROPAS에서 만든 SML과 OCaml의 한국형 함수형 언어이다. nML에서의 *soaptype* 타입으로 SOAP 값을 정의한다. SOAP은 XML Schema에 의해서 객체에 대한 인코딩이 정의된다. 즉 XML Schema가 SOAP 값의 유효성을 판단한다. 본 논문은 XML Schema의 엘리먼트와 타입에 대한 정의로써 *element*와 *typeinfo* 타입을 정의한다. 그리고 상호호환성 테스트를 통하여 안전하게 다른 언어와 대응됨을 보인다.

키워드 : 함수형 언어

Abstract Web service implementations are now rapidly growing. Web services are easily achieved by XML messaging for most programming languages. Applications usually utilize web services through APIs tied to a specific implementation of SOAP. nML is a dialect of SML and OCaml made in ROPAS. The *soaptype* type in nML is defined for the value of SOAP encoding. SOAP encoding specification defines rules for serialization of a graph of typed objects using XML Schema. XML Schema validates XML SOAP value. The *soaptype* type is encoded to XML and decoded from XML. It is necessary to guarantee safe encoding and decoding. So, the definitions for *element* and type definition in XML Schema are specified by *element* type and *typeinfo* type, which include the part of the definitions of XML Schema specification.

Key words : functional languages, nML, XML, XML Schema, SOAP, WSDL

1. 서 론

웹 서비스(web services)가 점점 발전함에 따라 구현이 많이 늘고 있다. 웹 서비스는 각 프로그래밍 언어에서 XML[1]을 이용해서 서비스를 쉽게 이용할 수 있게 하는 것이다. SOAP은 웹 서비스에서 서비스 제공자와 서비스 이용자간의 자료를 주고받는 대표적인 규약이다[2]. nML은 한국과학기술원 ROPAS에서 만든 함수형 프로그래밍 언어이다. nML 시스템은 안전하고 쉽게 구현될 수 있지만, 라이브러리의 부족으로 nML을 사용하기에 어려움이 많다. 그래서 nML을 위한 SOAP 클라이언트를 구현한다면, nML에서 부족한 라이브러리를 웹 서비스를 통해 충족시킬 수 있을 것이다.

SOAP은 객체 지향 언어, 스크립트 언어 등의 다른 언어에서 많이 구현되어 있다. 하지만 엄격한 타입을 가진 함수형 언어에서는 구현되어 있지 않다. SOAP은 구현된 프로그래밍 언어에서 함수 호출에 대한 표현을 XML로 바꾼 뒤(인코딩)에 선택된 프로토콜(주로 HTTP)을 이용해서 서비스를 받게 된다. 받은 서비스에 대한 결과 또한 XML로 되어 있으므로 거기에 대한 디코딩이 필요하다. nML은 엄격한 타입 언어이므로

• 본 연구는 첨단정보기술 연구센터를 통하여 과학재단의 지원을 받았음.
† 비 회 원 : 한국과학기술정보연구원 슈퍼컴퓨팅연구실
okkwon@hpcnet.ne.kr
†† 종신회원 : 한국과학기술원 전산학과 교수
han@cs.kaist.ac.kr
논문접수 : 2002년 6월 12일
심사완료 : 2002년 10월 31일

1) Simple Object Access Protocol이라는 용어에서 Service Oriented Architecture Protocol로 의미가 바뀌어 가고 있다.

SOAP에서의 값인 XML을 nML의 타입으로 명확한 대응이 필요하다. 그리고 XML에 대한 인코딩을 하기 위해서 스키마가 필요한데 SOAP에서는 XML Schema [3]를 사용한다. nML에서 인코딩을 표현하기 위하여 XML Schema을 정의할 수 있어야 한다. 다음과 같은 사항들이 설계하는데 고려되었다.

SOAP 인코딩 정의와 XML Schema 정의와의 관계

다음과 같은 두 가지의 문제가 있다. 첫 번째, SOAP 명세서에는 타입간의 인코딩에 대한 명시를 하고 있다. SOAP 명세서가 아직 표준으로 정의되어 있지 않고, 인코딩에 대해 명시된 사항은 추천사항이지 꼭 명세서를 따라야 하는 것은 아니다. 그러므로 각 구현간에 호환성이 잘 이루어지지 않을 수 있다. 두 번째, XML Schema의 타입 정의로 표현할 수 있는 범위는 일반적인 언어에서 표현할 수 있는 범위보다 넓다. 따라서 본 연구에서는 SOAP 명세서의 인코딩에 맞춰서 구현을 하되, XML Schema의 타입 정의에 유연하게 대응할 수 있게 정의를 할 것이다.

SOAP의 값을 nML의 값으로 표현

nML은 함수형 언어로써 엄격한 타입중심의 언어이다. Java같은 객체지향언어는 `Object` 클래스 등의 상위 클래스를 통해서 값을 할당하고 받을 수 있다. Perl 같은 스크립트 언어는 보통 타입을 가지지 않으므로 값이 안전하지는 않지만 자동으로 할당하고 받게된다. 하지만 nML은 숫자, 문자열 등의 정해진 하나의 타입으로 값을 표현할 수가 없으므로, SOAP 인코딩을 표현하는 `soaptype` 타입을 선언한다. 그러므로 SOAP에서 사용하는 값인 XML과 nML에서 사용하는 `soaptype` 타입간의 대응관계가 중요하다.

값에 대한 유효성

`soaptype` 타입은 값과 타입에 대한 의미를 가지고 있지만, XML로 변환하기 위해서는 엘리먼트에 대한 표현이 필요하다. 제대로 된 값이 되는지 표현하기 위해서 `soaptype` 타입의 의미만으로 부족하기 때문에 새로운 타입이 필요하다. 그래서 XML Schema 정의와의 관계에 맞게 `element` 타입과 `typeinfo` 타입을 정의하도록 한다. nML의 값과 SOAP의 값이 `soaptype`, `element`, `typeinfo` 타입에 잘 대응이 되어서 XML의 자료로 인코딩과 디코딩이 안전하게 됨을 보여준다.

상호 호환성

SOAP의 가장 큰 목적 중에 하나가 다른 환경에 있는 다른 언어를 손쉽게 사용하는데 있다. 그러기 위해서 정의한 것이 웹 서비스 묘사 언어(WSDL : Web Services Description Languages)명세서[4]이다. 이것에 맞게 `soaptype` 타입과 `typeinfo` 타입을 정의할 수 있으면 상호 호환이 가능한 것이다.

본 논문의 구성은 다음과 같다. 2장에서는 nML에서 사용하는 값을 SOAP에서 이용하는 XML의 내용으로 연결하는 것에 대해서 살펴본다. 3장에서는 2장에서 정의한 값의 제대로 된 인코딩과 디코딩을 보장하기 위해 새로운 `element`, `typeinfo` 타입을 도입한다. 4장에서는 SOAP을 nML로 구현한 시스템의 구조와 상호 호환성에 대해서 알아보겠다. 5장 관련 연구에서는 SOAP을 사용하고 있는 여러 인터페이스에 대해서 살펴보겠다. XML과 함수형 언어에서 타입 대응 관계에 대한 다른 연구에 대해서도 보겠다. 마지막으로 6장에서는 결론과 앞으로의 연구 방향에 대해서 논의한다.

2. nML에서의 soaptype 타입 설계

본 장에서는 본 연구에서 중점이 되는 nML에서 사용하는 값을 SOAP에서 이용하는 XML의 내용으로 연결하는 것에 대해서 살펴본다.

2.1 XML와 nML에 대한 값 할당

XML자료에 대해서 각 언어의 값으로 할당하는 경우는 다음과 같다.

- C++인 경우에는 `void` 타입으로 할당한 다음, 사용자가 사용하기 위해 `int`, `string` 등으로 타입 변환을 하면 된다. Java인 경우에는 `Object` 타입으로 할당한 다음, 사용자가 사용하는 타입인 `int`, `string` 등으로 타입 변환을 하면 된다.
- Perl같은 타입이 없는 스크립트 언어와 같은 경우에서는 타입 변환이 자동으로 이루어진다.
- nML같은 엄격한 타입 언어는 값을 단 하나의 타입으로 받으므로, 모든 타입을 하나로 표현하는 `soaptype` 타입을 선언한다. 이런 방법으로 XML 데이타를 `soaptype` 값으로 대응시킬 수가 있다. 이를 위하여 `soaptype` 타입을 정의한다. C++와 Java의 경우에는 `void`와 `Object`등의 타입으로 묻혀져 특정 값에 대한 타입의 속성을 잃게 된다. 하지만 nML의 경우에는 `soaptype` 타입으로 정의가 되고 타입에 대한 속성을 잃지 않고 패턴 매칭으로 쉽게 값을 얻어 올 수 있다.

2.2 *soaptype* 타입

`soaptype` 타입은 SOAP 명세서에 나와있는 인코딩에 의해 정의되어야 한다. 그리고 SOAP에서 사용하는 타입은 DTD가 아닌 XML Schema에 정의된 타입이 사용되므로[3], XML Schema에 정의된 타입과 SOAP 명세서에 정의된 타입과의 관계가 고려되어야 한다.

XML Schema는 자식 엘리먼트를 가지고 있느냐에 따라서 *simpleType*과 *complexType*으로 나뉜다. 여기에 따라서 SOAP 인코딩은 단순(simple), 복합(compound) 두 가지 타입으로 정의된다. 본 연구에서는 *soaptype* 타입을 단순, 복합, 기타 세 종류로 나누어 설계한다.

- 단순타입

SOAP 인코딩을 하는 경우에 XML Schema의 *simpleType* 타입과의 다른 점은 속성이 필요 없다는 것이다. *soaptype* 타입이 필요로 하는 것은 문자열이다. 그림 1에서처럼 S_SIMPLE 선언자만으로도 모든 단순 타입을 정의할 수 있지만, 이것만으로는 nML이 제공하는 타입 시스템의 이점을 살릴 수 없다. 즉 S_SIMPLE 선언자는 문자열을 가지고 있지만, 이것이 실제로 숫자가 될 수도 있고 이진 문자열이 될 수도 있다. 하지만 nML 타입 시스템은 숫자인지 이진 문자열인지 S_SIMPLE의 정의만으로 모른다. 만약 세분화한다면 사용자의 입장에서 현재의 타입이 숫자인지 문자열인지 표현하는 편리성을 얻을 수 있다. 그래서 XML Schema의 *simpleType*에 대해서 SOAP에서 자주 사용하는 단순 타입의 경우 위와 같이 세분화하기로 한다. XML Schema의 단순타입은 매직타입(S_ANYTYPE), 내장타입(S_DECIMAL, S_STRING, S_FLOAT, S_BOOL, S_HEXBINARY, S_BASE64BINARY등), 확장타입(S_INTEGER, S_BYTE, S_INT, S_LONG등)으로 나뉘어져 있다. 위에서 정의되지 않은 경우나 추가되는 경우에는 S_SIMPLE 선언자를 사용하면 된다.

- 복합타입

SOAP 인코딩 명세서는 그림 2에서처럼 구조체(XML Schema의 *Struct*)와 배열(XML Schema의 *Array*) 두 가지 복합 타입을 정의한다. 두 가지 타입 모두

```
type soaptype =  S_ANYTYPE of string (* magic
                 type *)
               | S_DECIMAL of real
               | S_STRING of string
               | S_FLOAT of real
               | S_BOOL of bool
               | S_HEXBINARY of string
               | S_BASE64BINARY of string
               | S_BYTE of int (* 8-bit signed
                 integer *)
               | S_SHORT of int (* 16-bit
                 signed integer *)
               | S_INT of int (* 32-bit signed
                 integer *)
               | S_LONG of int64 (* 64-bit
                 singed integer *)
               | S_SIMPLE of string (*
user-defined simpleType *)
```

그림 1 *soaptype* 타입에서의 단순타입 정의

```
type soaptype = S_RECORD of type_id *
                (contents_type_id * soaptype)
                list
              | S_ARRAY of soaptype array
and type_id = string
and contents_type_id = string
```

그림 2 *soaptype*타입에서의 복합타입 정의

XML Schema의 *complexType* 타입에 해당한다. 이들은 XML Schema의 *complexType* 타입과 그대로 대응해서 사용할 수 있다. 구조체 타입은 S_RECORD 선언자에 대응이 된다. *type_id*는 구조체 타입의 이름이고, *contents_type_id*는 구조체의 각 원소의 해당 이름이다. *type_id*는 XML Schema에서 타입 정의를 할 때 사용되는 이름이 되고, *contents_type_id*는 XML Schema에서 각 엘리먼트들의 이름이 된다. 배열은 S_ARRAY 선언자에 의해서 값을 설정할 수 있다.

- 기타타입

SOAP을 처리하는 서버에서 에러가 발생하거나, 네트워크에 문제가 생겼을 때 클라이언트로 돌아오는 값은 fault이다. 이때는 nML에서 제공하는 예외상황(exception)을 사용하지 않고, 에러 또한 XML값이므로 다른 값들과 마찬가지로 *soaptype* 타입에서 정의하였다. S_UNIT선언자는 기본 타입으로 아무런 값 할당이 없는 타입이다. 받은 XML 값이 빈 엘리먼트일 때 S_UNIT 선언자에 할당된다.

```
type soaptype = S_FAULT of faulttype
              | S_UNIT
and faulttype = { faultcode : t_qname,
                  faultstring : string,
                  faultactor : string,
                  detail : string }
```

그림 3 *soaptype* 타입에서의 기타타입 정의

3. soaptype타입에 대한 유효성(validity)

본 장에서는 XML 값이 nML의 *soaptype* 타입 값으로 제대로 인코딩하고 디코딩되는 것을 보장하기 위해 새로운 타입 *element*와 *typeinfo*를 도입한다.

3.1 XML Schema의 표현

2장에서 nML의 *soaptype* 타입으로 XML의 데이타를 대응시킨 것을 보았다. XML의 엘리먼트 데이타는 어떤 타입인지 명시적으로 "xsi:type"속성으로 나타내지 않으면 엘리먼트 이름으로 유추하게 된다[2]. 이때 XML Schema의 정의를 통해 엘리먼트 이름과 XML 데이타 타입간의 관계를 정의하고 이것을 이용해서 인코딩과 디코딩이 이루어지게 된다.

```
type element = element_name * typeinfo
and element_name = t_qname
and t_qname = QName of (t_namespace * t_name)
            | UQName of (t_name)
and t_namespace = string
and t_name = string
```

그림 4 엘리먼트에 대한 정의

XML Schema는 타입, 엘리먼트, 속성에 대한 정의들의 모음이다. 이때 속성은 SOAP에서 사용되지 않는다. 본 연구에서는 타입과 엘리먼트에 대해서 typeinfo 타입과 element 타입을 정의한다. 즉 XML 엘리먼트는 element 타입을 통해 확인을 받고(유효성) soaptype 타입으로 값을 제대로 인코딩한다. 먼저 element 타입에 대해서 살펴보자.

그림 4는 *element* 타입을 정의한 것이다. XML 데이타의 엘리먼트 이름(*element_name*)과 타입(*typeinfo*)으로 이루어져 있다. *element_name*은 *t_qname* 타입이다. XML Schema는 네임스페이스[5]를 사용하여 각 구성요소의 엘리먼트들을 구분 및 처리하도록 규정하고 있다. 즉 *t_qname*은 인코딩 형태에 따라서 퀄리파이드(qualified) 이름일 수도 있고(QName 선언자) 아닐 수도 있다(UQName선언자).

그림 5에서 123이라는 값을 S_INT 123이라는 nML *soaptype* 값으로 제대로 가져오기 위해서 *element* 타입을 이용하는 것을 볼 수 있다. 네임스페이스인 ns에 arg엘리먼트가 정의가 되어 있고, 그 타입이 T_INT라는 것을 통해 S_INT 123으로 값을 얻어 낼 수 있다.

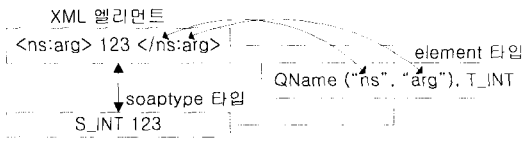

그림 5 XML값에서 nML 값으로 유효성 검사

3.2 *typeinfo* 타입

위에서 보았듯이 각 네임스페이스에 정의되어 있는 XML 타입을 정의해야 한다. 그 타입은 XML Schema에 정의되어 있다. 앞에서 정의한 *soaptype* 타입에 대응하여 SOAP 인코딩 명세서의 맞게 *typeinfo* 타입을 정의하자.

- 단순타입

그림 6에서 *soaptype* 타입의 S_SIMPLE 선언자에 대응하는 타입을 정의한다. XML Schema에서 정의하듯이 *simpleType* 타입은 문자열만으로 이루어진 엘리먼트의 경우에 사용이 된다. 각각의 문자열에 제한조건을 사용하게 된다. 사용자가 아래와 같이 T_SIMPLE타입을 이용해서 XML Schema의 *simpleType* 타입의 제한조건을 만들어 낼 수 있다.

```
type typeinfo = T_ANYTYPE (* magic type *)
              | T_DECIMAL
                  | T_STRING
                  | T_FLOAT
                  | T_BOOL
                  | T_HEXBINARY
                  | T_BASE64BINARY
              | T_BYTE
              | T_SHORT
              | T_INT
              | T_LONG
              | T_SIMPLE of type_qid *
                derivation_type * typeinfo *
                (string -> unit)
                 (* user defined type *)
              | T_ENUM of type_qid * (soaptype)
                list (* enumeration type *)
and derivation_type = NONE
                    | RESTRICTION
                    | EXTENSION
and type_qid = t_qname
```

그림 6 *typeinfo* 타입에서의 단순타입 정의

XML Schema가 상속을 지원하므로 단순 타입을 만들 때에도 상속을 지원한다. *derivation_type*, *typeinfo*를 통해서 제한(RESTRICTION)할 것인지 확장(EXTENSION)시킬 것인지 아니면 상속하지 않는 타입(NONE)인지를 설정한다. 그리고 마지막의 string-> unit함수 타입은 사용자의 제한 조건에 대한 값이다. 즉 숫자만 와야 하는 경우에 문자가 들어오면 예외처리로 하면 된다.

앞장에서 단순타입에 대해서 soaptype 타입을 정의하였듯이 XML Schema의 매직 타입(T_ANYTYPE), 내장 타입(T_DECIMAL, T_STRING, T_FLOAT, T_BOOL, T_HEXBINARY, T_BASE64BINARY등), 확장타입(T_INTEGER, T_BYTE, T_INT, T_LONG 등)을 정의한다. 일부 내장 타입과 string 타입의 확장 타입들은 정의하지 않았다. 일부 내장 타입은 T_ANYTYPE 선언자를 확장해서 정의할 수 있고, string 타입의 확장 타입은 T_STRING 선언자를 통해서 T_SIMPLE타입으로 정의할 수 있다.

그림 6에서 정의한 타입들은 XML Schema에서의 상속과 같은 관계를 가지게 된다. 예를 들어 T_INT를 가진 타입에 대해선 S_INT와 S_SHORT의 값은 가질 수 있지만, S_LONG은 예외상황이 된다. 사용자에 의해 *derivation_type* 을 통해 새로 정의된 단순 타입은 상속 관계에 추가된다.

마지막으로 SOAP 인코딩 명세서는 나열(*enumeration*) 타입을 지원한다. XML 스키마에서 나열 타입은

boolean 타입을 제외한 나머지 *simpleType* 타입에 대해서 정의한다. *typeinfo* 타입에 대해서 T_ENUM 선언자를 이용해서 정의한다. 이때 해당하는 값이 *soaptype* 타입 리스트에 들어 있는 값이 아니면 유효한 값이 아니게 된다. 즉 사용자가 나열 타입을 통해서 값을 보내거나 받기 전에 유효성 검사의 범위를 강화할 수 있다.

- 복합 타입

앞의 *soaptype* 타입은 XML Schema의 *complex Type* 타입에 대응하여 그림 7에서처럼 복합 타입을 정의하였다. 여기서도 SOAP 명세서에 따라서 다시 구조체와 배열 두 가지의 타입으로 나뉜다.

구조체 타입

XML Schema의 complexType 타입은 다음과 같은 조합기(Compositor)로써 정의한다. 조합기는 complex Type안의 내용에 대한 순서에 대한 중요도로 T_RECORD, T_RECORD_CHOICE, T_RECORD_ALL 선언자로 정의된다. 내용 간의 앞뒤 순서가 중요하면 T_RECORD를 사용하고, 정의된 것 중에서 어느 하나만 있으면 된다면 T_RECORD_CHOICE, 엘리먼트 간의 순서는 중요하지 않을 때는 T_RECORD_ALL을 사용하면 된다.

XML Schema에서는 조합기에서 *minOccurs*, *max Occurs*속성을 이용해서 반복되는 내용의 작업이 필요하다. 그래서 T_REPEAT 선언자를 정의한다. 그리고 T_RECORD_ALL 선언자는 *element*항목에 올 수 없다.

```
type typeinfo = T_RECORD of type_qid * element
                  list
              | T_RECORD_CHOICE of type_qid *
                element list
              | T_RECORD_ALL of type_qid *
                element list
              | T_REPEAT of element list *
                minOccurs * maxOccurs
              | T_ARRAY of element * dimen_index
and type_qid = t_qname
and minoccurs = string
and maxoccurs = string
and dimen_index = string
```

그림 7 *typeinfo* 타입에서의 복합타입

배열 타입

soaptype 타입의 S_ARRAY 선언자의 값에 대한 제한 조건이 된다. XML Schema에 대한 관계는 S_ARRAY 타입간의 관계와 같다.

SOAP 인코딩 명세서는 배열의 배열과 다중 배열 모두를 정의한다.

배열의 배열은 *element* 타입을 통해서 다시 T_ARRAY를 정의할 수 있다.

```
<array1 soapenc:arrayType="xsd:string[][1]">
<array2 soapenc:arrayType="xsd:string[2]">
<Item>string</Item>
<Item>string</Item>
</array2>
</array1>
```

위의 XML자료에 대해서 다음과 같은 *element* 타입을 정의할 수 있다.

```
"array1", T_ARRAY (("array2", T_ARRAY
(("Item", T_STRING), "")), "")
```

다중 배열은 *dimen_index*를 통해서 정의할 수 있다. 이때 배열에 대한 인덱스는 *dimen_index*를 통해 나타내고, 값은 S_ARRAY 선언자를 통해 1차원으로 저장한다. *dimen_index*가 빈 문자열일 때는 일차원 배열로 가정한다. *dimen_index*의 형식은 SOAP 인코딩 명세서처럼 숫자와 콤마(,)로 나누어진다.

```
<array1 soapenc:arrayType="xsd:string[1, 2]">
<Item>string</Item>
<Item>string</Item}>
</array1>
```

위의 XML자료에 대해서 다음과 같은 *element* 타입을 정의할 수 있다.

```
"array1", T_ARRAY {("Item", T_STRING), "1, 2"}
```

- UNIT 타입

soaptype 타입의 S_UNIT 타입에 해당하는 타입으로 다음과 같이 정의한다.

```
type typeinfo = T_UNIT
```

T_SIMPLE 선언자를 정의할 때, 상속관계가 아닌 경우에 *typeinfo* 타입의 T_UNIT 선언자를 사용하게 된다. 그리고 인자에 값이 없는 경우에 T_UNIT을 사용한다.

4. 구현 및 테스트

본 장에서는 nML SOAP 클라이언트의 구현과 호환성 테스트에 대해서 살펴본다. SOAP은 언어와 환경에 구분 없이 동작하는 것이므로 서로간의 호환성 테스트가 중요하다. 호환성 테스트는 다른 응용 프로그램의 인터페이스와 웹 서비스 묘사 언어(WSDL)[4]를 통해 어떻게 잘 동작하는지에 대해서 논한다.

4.1 구현

2장과 3장에서 설명한 값과 그것에 대한 유효성을 기반으로 nML 프로그래밍 언어로써 구현을 하였다.

4.1.1 전체 구조

그림 8은 nML SOAP 전체 프로그램의 구성도이다.

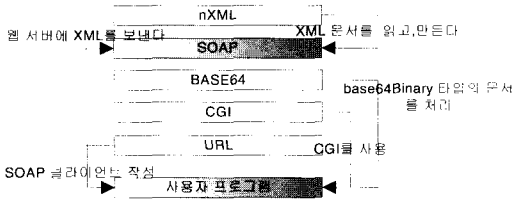

그림 8 nML SOAP 모듈 구조

사용자는 SOAP, CGI, BASE64 모듈을 통해 SOAP 클라이언트를 작성하게 된다.

nXML[6]은 외부에서 받은 XML문서를 읽어서 파싱(parsing)하는 일과 다시 XML문서를 만들어 주는 일을 한다. 하지만 XML이 제대로 되었는지에 대한 유효성 검사는 하지 않는다. BASE64 모듈은 BASE64 이진 형식의 바이트(byte) 열로 바꾸어 주고, 그 반대의 일을 한다. URL에서는 HTTP로 데이타를 보내고 받는 메소드를 제공한다. 그리고 URL의 이름을 처리하는 인코더(encoder)와 디코더(decoder)를 제공한다. CGI는 SOAP을 위한 응용프로그램을 위해서 제공된다. 이제 상세한 SOAP 구조에 대해서 살펴보겠다.

4.1.2 SOAP 모듈 구조

SOAP 모듈은 크게 두 가지 부분으로 나뉜다. 함수를 XML로 만들어서 보내는 부분, XML을 받아서 nML값으로 받는 부분으로 나뉜다. 그리고 이 두 부분을 가지고 직접적으로 사용할 수 있는 저급(low level) 부분과 안쪽의 구현이 어떤 구조로 되어 있는지 신경을 쓰지 않아도 되는 고급(high level) 부분으로 나뉘어져 있다.

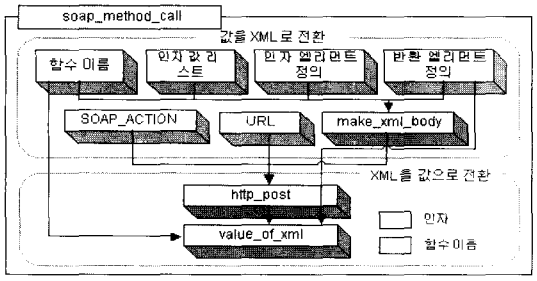

그림 9 nML SOAP 모듈 함수간의 관계

그림 9에서 사용자가 사용하는 함수는 soap_method_call로써 고급 수준의 층(high level layer)을 제공한다. soap_method_call에 있는 인자(parameter)들의 정보로서 SOAP의 함수가 호출이 되는 것이다.

4.2 상호 호환성

SOAP 구현의 가장 큰 목적은 각 언어에서 다른 언어의 함수를 호출할 수 있게 하는데 있으므로, 호환성이 가장 중요하다. SOAP 프로토콜은 웹 서비스 묘사 언어를 통해서 이종간의 상호 호환이 가능하다. 따라서 웹 서비스 묘사 언어에서 정의하는 조건을 여기서 다 표현할 수 있으면 상호 호환성의 조건을 만족하는 것이다. 먼저 웹 서비스 묘사 언어와 *soap_method_call* 함수의 인자들간의 관계와 XML로 인코딩 할 때 어떻게 연결되는지에 대해서 살펴보겠다. 그리고 마지막으로 각종 호환성 테스트에 의해 검증을 하겠다.

4.2.1 웹 서비스 묘사 언어와의 호환성

- 인자 인코딩

웹 서비스 묘사 언어는 SOAP을 이용하기 위해서 인자 인코딩에서 대해서 다음 두 가지 형태를 정의한다.

Literal 미리 정의된 XML Schema 정의에 따라서 인자들을 인코딩하는 방식이다. 정확한 XML Schema에 따라서 XML를 만든다. Encoded 경우는 작성자의 의도와는 다르게 엘리먼트가 인코딩 될 수 있지만 Literal에서는 확실하게 엘리먼트가 인코딩된다..

```
<soap:Envelope xmlns:xsd
 ="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/
soap/envelope/">
   <soap:Body>
      <DocumentLiteral xmlns
      ="http://www.contoso.com">
        <address>
<Street>Street</Street><City>Seoul</City>
        </address>
      </DocumentLiteral>
    </soap:Body>
</soap:Envelope>
```

여기서 address라는 엘리먼트 이름을 미리 정의된 네임 스페이스에 따라서 알 수가 있다. 즉 다음과 같이 nML에서 네임 스페이스 속성과 엘리먼트 이름에 대해서 타입을 정의함으로써 연결시킬 수 있는 것이다.

```
val contoso = "http://www.contoso.com"
val address_typeinfo = T_RECORD (QName
(contoso, "address"), [...])
val address_element = (QName (contoso,
"address"), address_typeinfo)
```

Encoded SOAP 명세서에 있는 인코딩 부분을 이용해서 인자들을 인코딩 하는 방식이다. Literal방식과 달

리 Encoded방식은 XML을 만들 때 정확한 XML Schema 정의에 따르는 것이 아니라, 추상적인 타입 정의에 따라서 만들어진다. 정의된 인코딩 형태에 의해 정해진 규칙에 따라서 XML이 만들어진다. 자료를 보내려고 XML을 만들 때에 다음과 같이 대응이 된다.

```
<soap:Envelope xmlns:tnsTypes
  ="http://www.contoso.com/encodedTypes"
 xmlns:soap="http://schemas.xmlsoap.org/
soap/envelope/">
   <soap:Body soap:encodingStyle
              = "http://schemas. xmlsoap.
                org/soap/encoding/">
     <tnsTypes:DocumentEncoded>
       <address href="#1" />
     </tnsTypes:DocumentEncoded>
     <tnsTypes:Address id="1">
       <Street id="2">string</Street>
       <City id="3">string</City>
     </tnsTypes:Address>
   </soap:Body>
 </soap:Envelope>
```

여기서 Address라는 엘리먼트 이름은 tnsTypes 네임 스페이스에 정의된 타입 이름이다. 즉 타입에 의해 인코딩 되는 것을 엘리먼트에 나타낸 것이다.

```
val tnsTypes
= "http://www.contoso.com/encodedTypes"
val address_typeinfo
= T_RECORD (QName (tnsTypes, "address"), [···])
val address_element = (UQName "address",
address_typeinfo)
```

- 함수 전체 인코딩

웹 서비스 묘사 언어는 SOAP의 함수 전체, 즉 XML의 Body 엘리먼트 전체를 인코딩하는 방법으로서 두 가지 형태를 정의한다. nML SOAP은 인자 인코딩과 동일하게 Document와 RPC 두 가지 모두에 대해서 잘 대응을 하고 있다.

4.2.2 호환성테스트

xmethods.com에서 명시하고 있는 SOAP 상호 호환 테스트 명세서[7]에 따라 시행하고, C#으로 구현되어 있는 마이크로소프트 닷넷[8]과 Java로 구현되어 있는 Apache SOAP[9]을 사용해서 테스트하였다. 마이크로소프트 닷넷 경우의 전체 인코딩은 Document형태이고, 인자 인코딩은 Literal형태이다. Apache-SOAP 경우의 전체 인코딩은 RPC 형태이고 인자 인코딩은 Encoded 형태이다.

echoString 함수에 대해서 살펴보자. nML 클라이언트는 서버 측에 스트링을 보내면 다시 스트링 값을 보내어 받게 된다. 마이크로소프트 닷넷의 경우에 살펴보자.

```
val tns = "http://schemas.xmlsoap.org/soap/encoding/"
val method_name = QName (tns, "echoString")
val param_list = [S_STRING "123"]
val param_element_list [(QName (tns, "str")),
T_STRING)])
```

nML 코드에 대해서 다음과 같이 XML로 변환할 수 있다. 여기서 Document형식이므로 str이라는 인자에 대해서 QName임을 알 수 있다.

```
<tns:echoString
 SOAP-ENV:encodingStyle
 ="http://schemas.xmlsoap.org/
 soap/encoding/"
 xmlns:tns="http://schemas.xmlsoap.org/
 soap/encoding/">
   <tns:str xsi:type="xsd:string">
    123</tns:str>
</tns:echoString>
```

Apache SOAP의 경우에는 param_element_list가 바뀌어져야 한다.

```
val param_element_list [(UQName "str"),
T_STRING)])
```

nML 코드에 대해서 다음과 같이 XML 값으로 바뀌어진다.

```
<tns:echoString
SOAP-ENV:encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/"
xmlns:tns="http://schemas.xmlsoap.org/
soap/encoding/">
  <str xsi:type="xsd:string">123</str>
</tns:echoString>
```

5. 관련 연구

본 장에서는 SOAP을 사용하고 있는 여러 인터페이스(interface)에 대해서 살펴본다. 그리고 XML과 함수형 언어(functional language)에서 타입 대응 관계에 대한 다른 연구에 대해서도 살펴본다.

5.1 웹 서비스를 위한 SOAP 인터페이스

웹 서비스는 HTTP나 SMTP같은 표준 프로토콜(protocol)을 통해서 다른 곳에 있는 소프트웨어 콤포넌

트(component)를 접근하게 하는 것이다. 인터넷과 XML를 이용해서 언어와 플랫폼에 관계없이 다른 콤포넌트들과 통신하는 소프트웨어를 만들고, 웹 서비스 묘사 언어(WSDL)를 통해서 연결을 시키며, UDDI를 통해 서비스를 배포하는 일을 한다. 기존의 다른 벤더들에서는 DCOM, CORBA등의 프로토콜을 통해 소프트웨어 콤포넌트를 접근해야 했었다. 하지만 웹 서비스는 표준화된 XML를 통해 어디서든지 재 사용할 수 있다. SOAP은 웹 서비스에서 서비스 제공자와 서비스 이용자간의 자료를 주고받는 대표적인 규약이다. 이 때문에 웹 서비스를 위한 여러 제품들이 연구 개발되고 있다. 닷넷 프레임워크[8]는 C#, C++, Java등의 여러 언어에 대해서 SOAP 관련 개발을 하고 있다. Apache SOAP[9]과 WASP[10]같은 제품은 Java로 SOAP을 구현하고 있다. 본 연구에서는 nML로 SOAP를 구현함으로써 다른 함수형 언어에서도 쉽게 구현할 수 있음을 보이고 있다.

5.2 XML와 함수형 언어

앞에서 살펴본 각 제품들은 객체 지향 언어를 위주로 해서 만들어진 언어이다. XML을 다루는데 있어서도 객체지향 개념이 많이 사용되고 있다. 즉 XML을 객체 모델화(DOM)하거나, 문서간의 상속관계 등이 많이 사용된다. 하지만 객체 지향 모델만이 XML을 표현하는 방법이 될 수는 없다.

XML은 값을 트리 구조로 표현하기에 자연스럽다. 함수형 언어는 트리 구조를 다루고 표현하는데 쉽고 좋은 특징을 가지고 있다. XML은 구조가 있고 그것을 정의하는 스키마들이 있다. 함수형 언어는 값 중심의 언어이고 값에 대해서 선언을 하는 타입이 있다[11]. XML을 정의하는 스키마 언어에는 DTD, XML Schema등이 있다. 기존에는 DTD를 가지고 함수형 언어를 표현하는 연구가 되어 왔다. 하지만 SOAP은 XML Schema를 사용하기 때문에 XML의 구조 뿐 아니라 타입자체에 대한 연구가 필요하다.

XML을 함수형 언어로 표현하기 위해서 HaXML[12]같이 기존의 함수형 언어로 표현하는 방법도 있고, XDuce[13]와 XMLambda[14]같이 새로 언어를 정의해서 사용하는 방법이 있다. 일반 목적의 언어는 XML을 표현하기에 쉽지 않은 면이 있으므로 잘 맞게 다시 정의하는 것이다. 그러나, 새로 언어를 배워야 하고 그것만을 위해서 사용되기 때문에 응용하기가 쉽지 않다. 본 연구에서는 새로운 언어를 정의하는 것보다는 nML 언어에 SOAP의 표현 수단인 XML을 대응시켰다.

6. 결론 및 향후 연구

본 논문에서는 nML에서의 SOAP 클라이언트를 구현하였다. nML은 엄격한 타입을 가지고 있는 함수형 언어이다. Java에서는 모든 타입을 받을 수 있는 타입을 사용할 수 있다. 하지만 nML에서는 숫자나 문자열 등의 하나의 타입만으로 값을 받아낼 수가 없으므로 다음과 같이 타입을 정의하였다.

nML에서 사용되는 값과 SOAP에서 사용되는 XML의 값이 잘 대응됨을 보였다. 그러기 위해서 SOAP 인코딩에서 명시하고 있는 값들을 `soaptype` 타입으로 정의하였다. 값을 사용하면 그것이 제대로 된 값으로 주고받는지 확인하여야 한다. 그러기 위해서 SOAP은 XML Schema를 사용한다. XML Schema에서는 엘리먼트, 타입, 속성에 대해서 정의할 수 있는데, 여기서는 엘리먼트와 타입에 대한 정의만 사용한다. 엘리먼트는 이름과 타입정의로 이루어진 `element` 타입으로, 타입은 `typeinfo` 타입으로 정의하였다. XML Schema에 대한 정의를 그대로 사용하지만 `soaptype` 타입처럼 SOAP인코딩 명세서에 맞추어 구현하였다.

마지막으로 SOAP의 가장 큰 목적 중의 하나가 다른 환경에 있는 다른 언어를 손쉽게 사용하는데 있다. 그러기 위해 정의한 것이 웹 서비스 묘사 언어 명세서이다. 이것에 맞게 nML 프로그램을 정의할 수 있음을 살펴보고 상호 호환성 명세서를 통해서 다른 언어의 응용 프로그램 인터페이스와 호환됨을 보았다.

본 연구에서는 정의된 값과 거기에 대한 유효성에 대한 타입을 기반으로 nML SOAP 클라이언트를 구현하였다. nML의 값을 XML값으로 인코딩해서 서버에게 값을 전달하고, 서버는 처리한 다음에 서비스를 제공해 주기 위해서 다시 XML로 클라이언트에게 값을 주게 된다. 마지막으로 서버로부터 받은 XML값을 nML의 값으로 안전하게 디코딩하게 된다. 하지만 본 연구에서는 인코딩과 디코딩에 대한 부분만을 언급하였고, SOAPAction등의 헤더(header) 부분에 대한 처리는 논하지 않았다. 나아가서 nML SOAP 서버까지 구현한다면 안전하게 만든 nML 코드를 웹 서비스로서 등록할 수 있을 것이다.

참 고 문 헌

[1] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler. Extensible Markup Language (XML) 1.0 (Second Edition) (W3C Recommendation). http://www.w3.org/TR/REC-xml, WWW Consortium, October 2000

[2] Martin Gudgin, Marc Hadley, Jean-Jacques Moreau, Henrik Frystyk Nielsen. SOAP Version 1.2 Part 2: Adjuncts(W3C Working Draft 2). http://www.w3.org/TR/2001/WD-soap12-part2-20011002/, WWW Consortium, October 2001.
[3] Paul V. Biron, Ashok Malhotra. XML Schema Part 2: Datatypes. (W3C Recommendation) http://www.w3.org/TR/xmlschema-2/, WWW Consortium, May 2001.
[4] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, WWW Consortium, March 2001.
[5] Tim Bray, Dave Hollander, Andrew Layman. Namespaces in XML. http://www.w3.org/TR/REC-xml-names/, WWW Consortium, January-1999.
[6] Hyun-Goo Kang, Eun-Sun Cho, nXML, http://ropas.kaist.ac.kr/n/nxml/
[7] ROUND 1 SOAP Interoperability Tests Specification. http://www.xmethodsnet/soapbuilders/proposal.html.
[8] Jeffrey Richter, Part 2: M Microsoft .NET Framework Delivers the Platform for an Integrated, Service-Oriented Web, http://msdn.microsoft.com/msdnmag/issues/1000/Framework2/Framework2.asp. October 2000.
[9] Apache SOAP Version 2.2. http://xml.apache.org/soap/index.html. 30, 2001.
[10] WASP http://www.systinet.com/index.html.
[11] Bijan Parsia. Functional Programming and XML. http://www.xml.com/pub/a/2001/02/14/functional.html, xml.com. February 2001.
[12] Malcolm Wallace and Collin Runciman. Haskell and XML : Generic Combinators or Type-Based Translation?. In Proceedings of the Fourth ACM SIGPLAN International Conference on Functional Programming(ICFP'99). ACM Press, Sept 1999.
[13] Haruo Hosoya and Benjamin C. Pierce. XDuce: A typed XML processing language. In Proceedings of Third International Workshop on the Web and Databases (WebDB2000), volume 1997 of Lecture Notes in Computer Science, pages 226-244, May 2000.
[14] E. Meijer and M. Shileds. Xmlambda : A functional programming language for constructing and manipulating xml documents. In Submiited to USENIX 2000 Technical Conference, page 13 pages.

권 오 경
1996년 ~ 2000년 고려대학교 컴퓨터학과 학사 졸업. 2000년 ~ 2002년 한국과학기술원 전산학과 석사 졸업. 2002년 ~ 현재 한국과학기술정보연구원 슈퍼컴퓨팅센타 연구원

한 태 숙
1976년 서울대학교 전자공학과 졸업. 1978년 한국과학기술원 전산학과 졸업. 1990년 Univ. of North Carolina at Chapel Hill 졸업. 현재 한국과학기술원 전자전산학과 부교수. 관심분야는 프로그래밍 언어론, 함수형 언어