Proceedings of the 5th WSEAS International Conference on Applied Computer Science, Hangzhou, China, April 16-18, 2006 (pp135-140)

Constructing Web Service in Equivalent Transformation Programming Language

Zheng Cheng* Katsunori Katou Kiyoshi Akama
Information Initiative Center
Hokkaido University
Kita 11 Nishi 5, Sapporo, 060-0811
Japan

Abstract: - research on the Web Services has rapidly developed in recent years. A Web Service is a technology that makes the distributed applications on a network cooperate by using the standard technology of the Internet compared with the Web site. It transforms the methods and the data tendered to it from WSDL into a utilizable class, an mutual transformation of data and communication processing with the server are processed by programming done in a traditional language (C#, Java, and Perl, etc.). Here however, we examine the construction of the Web Service based on *ET* (Equivalent Transformation).

Key-Words: - ET, ET programming, Web Service, WSDL, SOAP, ET Built-In

1 Introduction

In recent years the Internet is rapidly spread to such an extent that it seems that we cannot live without it. In this situation we focus on “Web Services”, one of the technologies that utilize the Internet. The difference between this and traditional Web sites is that user’s requests can be automatically processed by the programs in the Web Service. In the case of a web site, various actions are carried out by a person. For example, "Judgment as to whether or not to trust a party" and "Verification as to whether or not the same bill has come twice", etc. will be processed automatically by computer in a Web Service. In other words, various processing done by humans on the computer up to this point will be automatically done by the Web Service. "Web Service" is a new technology in which not only the person but also the computer (program) retrieves, discovers, uses, and integrates various services (system and application) that exist in WWW [1]. Even though web Service was described in one sentence as “Technology to cooperate application”, Web Service is in fact not a single technology. A Web Service is a combined technology composed of technologies such as message technology, security technology, and transaction management technology, etc., a wide range indeed. However, in the composition of Web Services there are many technologies that are still in the process of being decided on. In addition, where there are two or more competing technologies it is not yet known which of them will survive in the end. Web Services technology is still in the developmental phase. Therefore, the purpose of this research is to make a frame available as for this technology in the *ET* language.

In this paper, a method for the implementation of Web Services in the programming language *ET* is proposed. We examine how to make each *ET* built-in rules (built-in predicate) for *Service Request* and Service provider that exists in the Web Service, and also how to achieve Service based on these *ET* built-in rules. Also, system SRSET (*Service Request* System in *ET*) of *Service Request* is constructed.

2 Equivalent Transformation Programming Language ET

All programs made in this research are described in equivalent transformation programming language *ET*.

2.1 Equivalent Transformation (*ET*) Theory

Equivalent Transformation (*ET*) theory is one of the calculation theories, and a calculation theory by equivalent transformation [2]. The transformation process in which some problem *prb* is changed into problem *prb'* with the meaning of *prb* preserved called equivalent transformation. That is, when we assume problem *prb* meaning to be *M(prb)* and problem *prb'* meaning to be *M(prb')*, it is shown that the relation is composed that is *M(prb) = M(prb')* between *prb* and *prb'*. The calculation in the *ET* theory is performed by

equivalent transformation, and the given problem is solved by repeated equivalent transformations that preserve the meaning. This is illustrated in Figure 1.

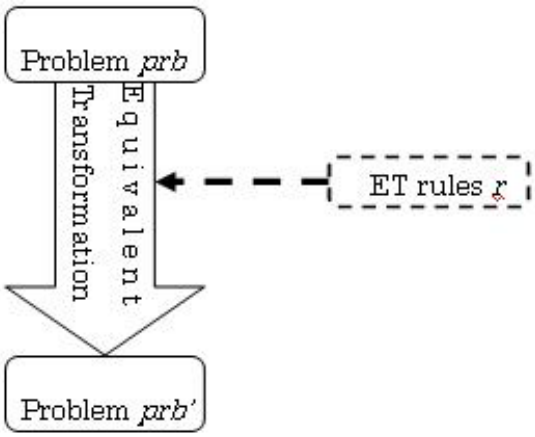

Figure1. Equivalent Transformation by *ET* rules

2.2 ET Programming based on Rules

Equivalent transformation programming language *ET* is designed based on the framework of the equivalent transformation calculation by equivalent transformation (*ET*) theory. The problem solving and programming of the *ET* language by equivalent transformation are associated as follows [3].

First of all, the problem to be solved is given by the definite clause (set) that defines the problem specification that describes the nature of the problem (condition and processing, etc. necessary for the problem solving). To solve the given problem by the definite clause, we simplify the problem by equivalent transformation. The information about the method of the equivalent transformation operation was described by *ET* rules. Information about the method of the operation for the atom of the clause to be transformed is "What transformation (simplification) is given when which atom satisfies what requirements".

Generally, to solve a problem to one or more simplifications by equivalent transformation is necessary, and information about all necessary equivalent transformation operations for the solution (*ET* rules) is a necessity. Necessary *ET* rules for this problem solving is assumed to be *r*, and all sets of *r* is assumed to be *R*. *R* is sets of *ET* rules, and it can be regarded as the programming that will solve the problem. That is to say, what is called programming in the *ET* theory is the description of the necessary *ET* rules for solving the problem. An *ET* program consists only of the *ET* rules, and the calculation by the *ET* rules, the cause equivalent transformation is assumed to be valid and efficient.

When *ET* rules are repeatedly applied to a problem until no further simplification is possible, whatever remains is the solution to that problem. This is illustrated in Figure 2.

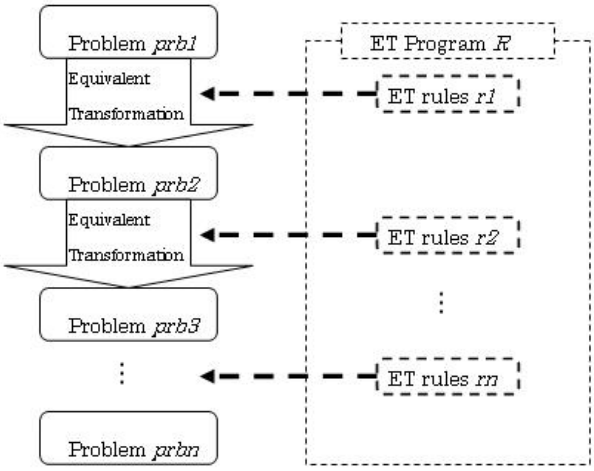

Figure2.Problem solving by equivalent transformation

The *ET* Programming language has the following features.

- It is a programming language due to the rules based on the theory of equivalent transformation.
- Two methods(S type and R type) for describing the rules exist.
- D rule (D is the initial of Deterministic) and N rule (N is the initial of Nondeterministic) exist.
- In the application of N rules, the selection of the transformation object clause, the transformation object atom, and the application rules is arbitrary.

The programming in this paper is constructed with D rule. D rule is described as follows.

(as *<Head atom> < Cond atom-list> : < Body atom-list>*)

3 Analysis of Web Service

A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web Service in a manner prescribed by its description using SOAP messages; typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards [4].The whole image of the Web Service is shown in Figure 3.

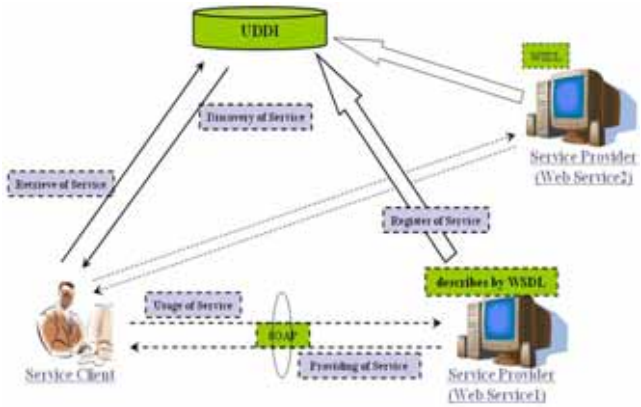

Figure3. The whole image of Web Service

3.1 SOAP message

In a Web Service, one side is specified to be used and the other side is specified to be offered. The used side is called the called *Service Request* and the offered side is called the *Service Provider*. The relation of the communication between *Service Request* to *Service Provider* is shown in Figure 4.

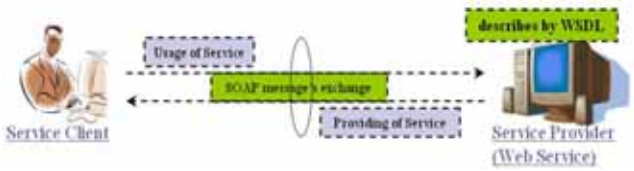

Figure4. Relation of the communication between *Service Request* and *Service Provider*

The key to achieving the service shown in Figure 4 is the exchanging of Soap messages. Therefore, it is essential that the necessary information that composes the SOAP message be obtained.

3.2 Relation of SOAP message to WSDL

A SOAP message is composed of an XML form. Its structure is illustrated in Figure 5.

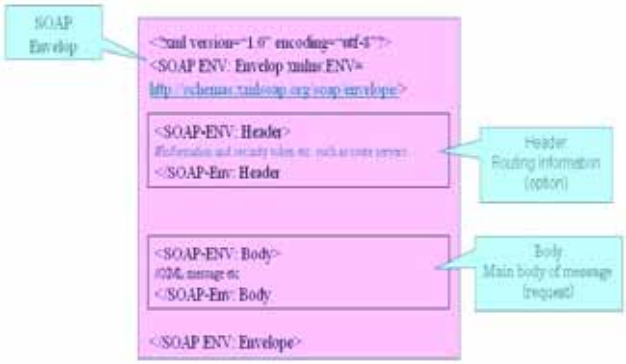

Figure5. The structure of a SOAP message

We can establish the relationship between a SOAP message and WSDL by looking at the structure of WSDL document. This structure is shown in Figure 6.

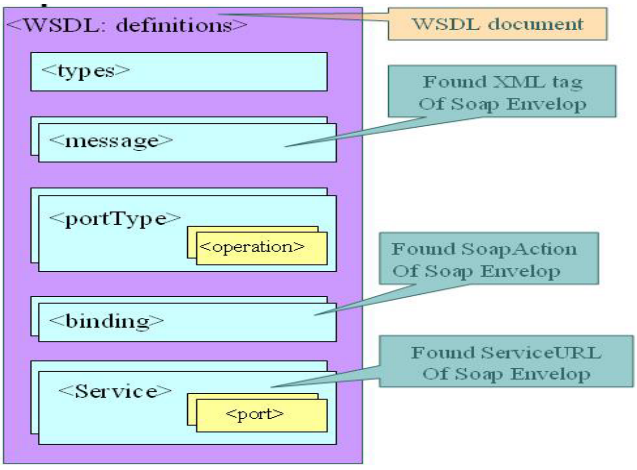

Figure6. The relation of a soap message to WSDL

As a result, by making a SOAP message from WSDL it becomes possible to implement a simple Web Service. The *Service Request* and *Service Provider* of the Web Service can now be constructed using *ET*.

4 Realization of Service Request side

The *service Request* side consists of the following five stages.

(1) Information on the available Web Service is obtained (from web page and WSDL etc.)
(2) An appropriate SOAP message is made for the available Web Service.
(3) Necessary information of the made SOAP message etc. is added behind the URL of the available Web Service.
(4) Transmission of made data
(5) Display of information received in the reply.

Built-in rules were then made from this "Outline of the *Service Request* side".

Built-in rules for Service Request

1. Built-in rules that acquired Information from WSDL
Rules name: wsdl:AcquireInfo
Shape of rules: (wsdl:AcquireInfo *wurl *result_usr *info *soapAction *surl)
Explanation of argument:
*wurl: URL of WSDL
*result_usr: Information that displayed to user
*info: Information on necessary condition in soap body
*soapAction: URL of soapAction
*surl: URL of Web Service

2. Built-in rules that connects with Web server, and receives result of service
Rules name: soap:InvokeWebService
Shape of rules: (soap:InvokeWebService *tag_value *info *soapAction *surl *service)
Explanation of argument:
* tag_value: Service tag name and value that user inputted
*info: Information on necessary condition in soap body
*soapAction: URL of soapAction
*surl: URL of Web Service
*service: Reply result from service provider

Built-in rule wsdl:AcquireInfo can acquire useful information about the construction of the SOAP message (tag name, SoapAction, and Service URL, etc.) from WSDL. For an example, information was acquired from URL of WSDL that is http://teachatechie.com/GJTTWebServices/ZipCode.asmx?WSDL The result is shown in Figure 7.

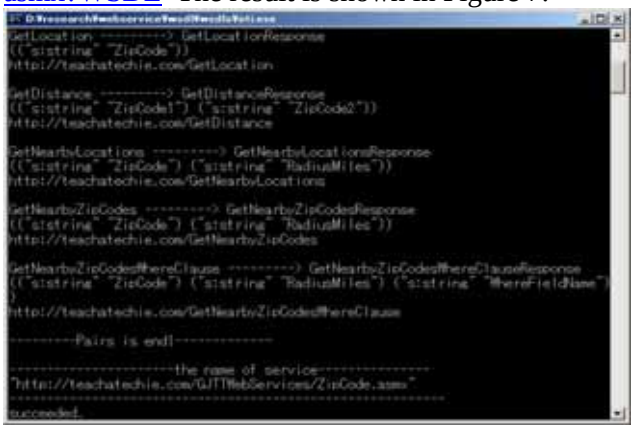

Figure7. Achievement of information from WSDL

Also, the information can be amended. For example: <Sertvice: GetLocation --> GetLocationReponse, Input necessity (condition): Stirng ZipCode, Service execution example: GetLocation cond*1*…cond *n*>. In other words, it makes the user understand and what service it is and how to use said service. The user can input the service tag name and the condition while consulting the example.
Based on the service name tag and condition inputted by the user, along with the relevant information from WSDL, the built-in rule soap:InvokeWebService can select the service generate the SOAP message automatically, send the SOAP message to the server, and then report reply the service results to the user. *Service Request* System- SRSET (*Service Request* System in *ET*) was constructed based on these built-in rules. The interface is shown in Figure 8.

Figure8. The interface of SRSET

5 Realization of Service Provider side

The *Service Provider* side consists of the following five stages.
(1) Receives the data from *Service Request* side. (SOAP message)
(2) Necessary data (information) is pulled out from the received data (SOAP message).
(3) Computation based on the data (processing).
(4) Use the acquired data to make a SOAP message.
(5) Reply to the *Service Request* side.
Built-in rules were then made from this "Outline of the *Service Provider* side".

Built-in rules for Service Provider

1. Built-in rules that receives transmitted message
Rules name: soap:decodeData
Shape of rules: (soap:decodeData *Mess)
Explanation of argument:
*Mess: Data that has been transmitted (SOAP message)

2. Built-in rules that pulls out necessary data from SOAP message
Rules name: soap:selectData
Shape of rules: (soap:selectData *Mess *pathList *data)
Explanation of argument:
* Mess: SOAP message
*pathList: Passing to data that wants to be pulled out
*data: Data that has been pulled out

3. Built-in rules that converts processing result into SOAP message, and replies
Rules name: soap:makeMessage
Shape of rules: (soap:makeMessage *Ans *Name *return)
Explanation of argument:
* Ans: Result of processing (list)
*Name: Name of Web Service (String)
*return: Message of replies (SOAP message)

As an example, the Web Service determines whether the given value (integer) is a prime number or not. If it is a prime number then TRUE is returned, otherwise a Web Service that returns FALSE is constructed.

Building of Service Provider

```
?- (chdir "C:/webs/eti-bin/webservices/primes").
// 1. loading built-in file
?- (loadModule "./dll/providerblt.dll" ?) .
(as (main)
// 2. Build-in rules soap:decodeData
(soap:decodeData *message)
:
// 3. Build-in rules soap:selectData
(soap:selectData *message ((soap:Body) (Prime (xmlns
"http://fox19.hucc.hokudai.ac.jp/webs/eti-bin/
webservices/primes")) (Element) dd:content) *data)
(atoi *data *Data)
// 4. judged whether it is a prime number
(Prime *Data *Ans)
// 5. Build-in rules soap:makeMassage
(soap:makeMassage (*Ans) "Prime" *return) )
//-------------------------------------------------------//
//Rules Prime
// Program that judges whether given positive integer is prime number
(as (Prime *N *Ans) (Prime *N) : (= *Ans "TRUE") )
(as (Prime *N *Ans) (not (Prime *N)) : (= *Ans "FALSE") )
(as (Prime 1) : (false) )
(as (Prime *N) (> *N 1) : (:= *N1 (- *N 1)) (Prime2 *N *N1) )
(as (Prime2 *N 1) )
(as (Prime2 *N *N1)
(> *N1 1)
(:= 0 (mod *N *N1))
: (false) )
(as (Prime2 *N *N1) : (:= *N2 (- *N1 1)) (Prime2 *N *N2) )
?- (rebuildRuless)(main) .
?- q .
```

The Soap message cannot be generated from WSDL because WSDL is not automatically generable and does not exist. The SOAP message has to be prepared. The Soap message is transmitted to *Service Provider* when executing it by ETI [5] and the result of the reply is shown as follows.

Result from Service Provider

```
[D]>(main 3)
------------------------D execution ---------------------
((soap:Envelope (xmlns:xsi "http://www.w3.org/2001/XMLSc
hema-instance") (xmlns:xsd "http://www.w3.org/2001/XMLSc
hema") (xmlns:soap "http://schemas.xmlsoap.org/soap/enve
lope/")) ((soap:Body) ((PrimeResponse (xmlns "http://fox
19.hucc.hokudai.ac.jp/webs/eti-bin/webservices/primes"))
((PrimeResult) (dd:content "TRUE")))))
Answer = TRUE
-----------------------------------------------------------
succeeded.
(main 3)
execution time: 213 [msec]
[D]>(main 20)
------------------------D execution ---------------------
((soap:Envelope (xmlns:xsi "http://www.w3.org/2001/XMLSc
hema-instance") (xmlns:xsd "http://www.w3.org/2001/XMLSc
hema") (xmlns:soap "http://schemas.xmlsoap.org/soap/enve
lope/")) ((soap:Body) ((PrimeResponse (xmlns "http://fox
19.hucc.hokudai.ac.jp/webs/eti-bin/webservices/primes"))
((PrimeResult) (dd:content "FALSE")))))
Answer = FALSE
-----------------------------------------------------------
succeeded.
(main 20)
execution time: 213 [msec]
```

6 Conclusion

In this paper, we proposed a method for implementing a Web Service in the equivalent transformation programming language *ET*. Also, we did actual experimentation on the provision and use of the Web Service by making the proposed Web Service and built-in rules (built-in predicate) for the *ET* program creation. In addition, *Service Request* System- SRSET (*Service Request* System in *ET*) was constructed.
We will make the mechanism for the automatic generation of WSDL document in the *Service Provider* in the future. Also, taking OWLS into consideration, we plan to construct more complex Web Services.

References:

[1]http://www.atmarkit.co.jp/fxml/tanpatsu/21websvc/websvc01.html
[2] Kiyoshi Akama etc., A Class of Rewriting Rules and Reverse Transformation for Rules-Based Equivalent Transformation, Electronic Notes in Theoretical Computer Science, 59 No.4 pp.1—16, 2001.
[3] K.Akama, H.Koike and H.Mabuchi, A Theoretical Foundation of Program Synthesis by Equivalent

Transformation, Perspectives of System Informatics, Lecture Notes in Computer Science, Vol. 2244, Springer Verlag, Heidelberg, 2001, pp. 131-139.

[4] W3C Web Services Architecture Working Group Note, http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/, 11 February 2004.

[5] ETI (Equivalent Transformation rule Interpreter). http://assam.cims.hokudai.ac.jp/eti/ http://assam.cims.hokudai.ac.jp/et/indexj.html (Japanese Edition)

.