International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:4, No:12, 2010

Web Service Providing Using Web Service Transformation

Youngmee Shin, Hyunjoo Bae

***Abstract*—**In order to provide existing SOAP (Simple Object Access Protocol)-based Web services with users who are familiar with REST (REpresentational State Transfer)-style Web services, this paper proposes Web service providing method using Web service transformation. This enables SOAP-based service providers to define rules for mapping from RESTful Web services to SOAP-based ones. Using these mapping rules, HTTP request messages for RESTful services are converted automatically into SOAP-based service invocations. Web service providers need not develop duplicate RESTful services and they can avoid programming mediation modules per service. Furthermore, they need not equip mediation middleware like ESB (Enterprise Service Bus) only for the purpose of transformation of two different Web service styles.

***Keywords*—**REST, SOAP, Web Services, Web Service Transformation.

I. Introduction

WEB services can be accessed by service consumers through abstract and easy interfaces and can be executed on remote systems hosting the requested services. Web services are based on SOAP (Simple Object Access Protocol), WSDL (Web Service Description Language), and UDDI (Universal Description, Discovery and Integration) standards [2]–[5]. Web services have been gaining attentions and recently are widely used because they are regarded as the default underlying technology for SOA (Service Oriented Architecture) [1]. Meanwhile, REST (REpresentationl State Transfer)-style Web services are also gaining popularity together with success of Web 2.0. A RESTful service is a simple Web service implemented using HTTP and the principles of REST [6][7]. RESTful services have several merits such as light-weight and easiness for users to use, as compared with SOAP-based Web services which are based on SOAP, WSDL, and WS-* stack .

As stated above, there are two technologies to provide Web services. Recently because of the popularity of RESTful Web services, existing SOAP-based service providers need to provide their services with users who are familiar with RESTful services.

Youngmee Shin is with the Service Platform Research Department, ETRI (Electronics and Telecommunications Research Institute), Daejeon 305-700 Korea (corresponding author to provide phone: 82-42-860-1314; fax: 82-42-861-1342; e-mail: ymshin@ etri.re.kr).

Hyunjoo Bae is with the Service Platform Research Department, ETRI (Electronics and Telecommunications Research Institute), Daejeon 305-700 Korea (e-mail: hjbae@etri.re.kr).

The simplest way to realize this requirement is that service providers develop the same RESTful services as existing SOAP-based services. But this has demerits such as duplication, developing cost, and maintenance.

ESB (Enterprise Service Bus) can be a solution for the requirement. ESB is a mediation middleware to integrate a variety of legacy applications of an enterprise using SOAP-based Web services [8]. Recently ESB tries to incorporate REST-style Web services [9]–[13]. But because most of ESB products are based on SOAP-based Web services, these can't meet the requirements for REST-style service users. Meanwhile, a few ESB products support exposing RESTful services in order to integrate many backend services [10][11][13]. These products are good solutions for REST users, but these are expensive solutions and they require lots of effort for adaptation and maintenance. Furthermore, these originally aim at integrating a variety of legacy applications, so it is not a cost-effective method to use ESB only for the purpose of conversion of two different Web services styles.

Policy Studio and XML Gateway of Vordel pursues the same objective as this paper [13]. Users create a policy which reads parameters from the REST URL and then inserts those parameters into a SOAP message which it creates on-the-fly. But Vordel makes users handle SOAP message directly and it covers mapping from REST URL to SOAP message. Furthermore, Vordel does not support for users to customize error messages and a variety of data representation such as RSS (Really Simple Syndication), JSON (JavaScript Object Notation), and etc.

In this paper, in order to provide existing SOAP-based Web services with users who are familiar with RESTful Web services, this paper proposes the method for Web service providing method using Web service transformaiton.

The paper is organized as follows: in section 2, it reviews the concept of Web services. Section 3 and 4 describes the Web service transformation which is main part of this paper. Finally it summarizes and comments on future work in section 5.

II. Web Services

A. SOAP-based Web Services

A Web service is defined by the W3C as "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 services 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" [2]. A Web service is frequently just Internet Application Programming Interfaces that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services.

Fig. 1 shows the general Web services architecture. A service provider publishes his/her Web services into UDDI registry using WSDL. A service requester looks for required services in UDDI registry. After finding the required service, the service requester gets WSDL of the found service from UDDI registry and then invokes the required service through sending SOAP-based request message to the service provider who provides the required service.

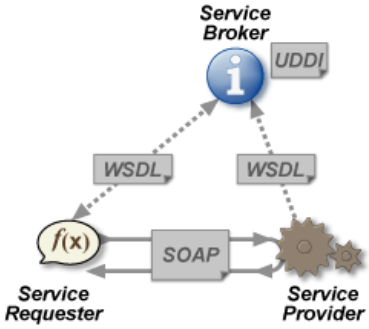

Fig. 1 Web services architecture

B. RESTful Web Services

RESTful Web services have gained widespread acceptance across the Web as a simpler alternative to SOAP-based Web services. Key evidence of this shift in interface design is the adoption of REST by mainstream Web 2.0 service providers, including Yahoo, Google, and Facebook, for an easier-to-use and resource-oriented model to expose their services.

REST defines a set of architectural principles by which developers can design Web services that focus on a system's resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages. REST Web services follow four basic design principles [6][7]:

- Use HTTP methods explicitly : GET, POST, PUT, DELETE
- Be stateless.
- Expose directory structure-like URIs.
- Transfer XML, JSON, or both.

If measured by the number of Web services that use it, REST has emerged in the last few years alone as a predominant Web service design model. In fact, REST has had such a large impact on the Web that it has mostly displaced SOAP-based interface design because it's a considerably simpler style to use.

III. Web Service Transformation

A. Web Service Providing System

Fig. 2 shows the Web service providing system which this paper is applied to. It consists of Web server, Web service converter, SOAP engine, and SOAP-based Web services.

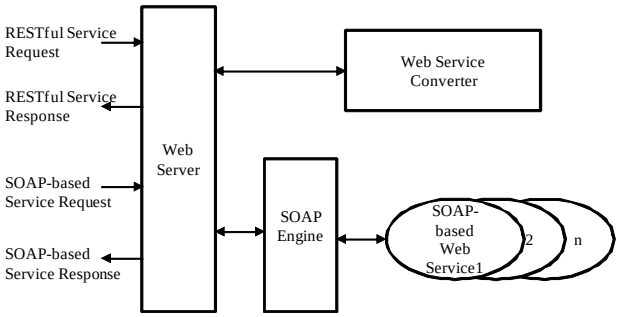

Fig. 2 Web service providing system using Web service transformation

The Web server receives and sends HTTP messages. It dispatches the received messages. It sends RESTful service requests to the Web service converter. And it sends SOAP-based service requests to the SOAP engine.

The Web service converter transforms a RESTful service request into a SOAP-based service invocation statement and then executes the generated statement. It also receives a result from the invoked SOAP-based service and transforms the received results into a RESTful service response.

The SOAP engine such as Apache Axis provides SOAP-based Web service framework. It includes implementation of the SOAP server, and various utilities and APIs for generating and deploying Web service applications. It handles SOAP and activates proper web services depending on request messages.

The SOAP-based Web Services provides services exposed to users through WSDL.

B. Web Service Converter

Fig. 3 shows the structure of the Web service converter depicted in Fig. 2. It consists of request receiver, conversion process manager, a number of conversion processes, and mapping rules.

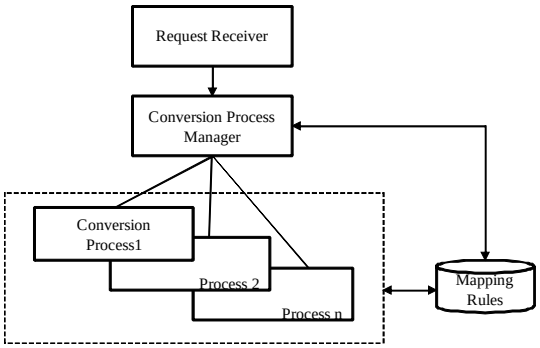

Fig. 3 Structure of the Web service converter

The request receiver receives RESTful service requests and validates the received requests.

The conversion process manager creates or kills conversion processes which are in charge of converting a RESTful service request into a SOAP-based service invocation statement. In case of abrupt stop of a conversion process caused by exception during processing, the conversion process manager generates a RESTful service response including description of error occurrence on behalf of the conversion process stopped abruptly and sends the generated message to the user who requested the service. The process of generating response including description of error occurrence is as follows. The conversion process manager makes a decision what kind of error occurs depending on exception types. And it determines HTTP status code corresponding to the occurred error. It obtains mapping rules corresponding to the determined status code and finally generates a RESTful service response which consists of HTTP status code, entity-headers, and an entity-body describing error types and reasons of errors.

The conversion processes transform the received RESTful service request into a SOAP-based service invocation statement by using mapping rules and then executes the generated invocation statement. These also receive the result returned by the invoked Web service and generate RESTful service responses by using mapping rules. And then these send the generated responses to the users who requested the services.

The mapping rules include rules for error handling and for transforming input parameters and results. These rules are described in next section.

Fig. 4 shows the structure of the conversion process at position *i* depicted in Fig. 3.

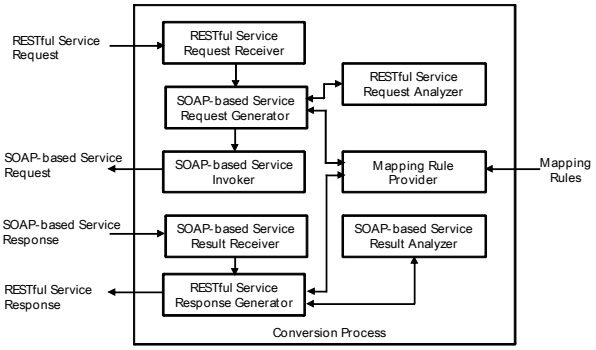

Fig. 4 Structure of the *i*th conversion process

The *i*th conversion process is created by the conversion process manager and terminates by itself in case of completing its processing. When some errors occur, it throws exceptions to the conversion process manager and terminates.

The *i*th conversion process consists of RESTful service request receiver, RESTful service request analyzer, SOAP-based service request generator, SOAP-based service invoker, SOAP-based service result receiver, SOAP-based service result analyzer, RESTful service response generator, and mapping rule provider.

The RESTful service request receiver receives request and validates the received request. After validation, it sends the received request to the SOAP-based service request generator.

The SOAP-based service request generator generates SOAP-based service invocation statement with the help of the RESTful service request analyzer and the mapping rule provider. The RESTful service request analyzer parses the received request message and extracts HTTP method, URL resource name, and URL query string from the received message. And it separates tokens from the extracted URL query string based on the delimiter '&' [14][15]. The mapping rule provider finds mapping rules corresponding to the extracted HTTP method and URL resource and provides the found mapping rules with the SOAP-based service request generator.

The SOAP-based service request generator applies tokens separated from URL query string to input parameter mapping rules provided by the mapping rule provider. It finally generates a SOAP-based service invocation statement and the generated statement is sent to the SOAP service invoker. The SOAP service invoker executes the generated invocation statement.

The SOAP-based service result receiver receives a result returned by the invoked SOAP-based service and it sends the received result to the RESTful service response generator.

The RESTful service response generator generates a response with the help of the SOAP-based service result analyzer and the mapping rule provider. The SOAP-based service result analyzer validates if there is an error in a received result. The mapping rule provider provides the applicable mapping rules, which were obtained during generation of invocation statement, to the RESTful service response generator.

If there is no error in the result, the RESTful service response generator generates a HTTP status code representing success such as 2xx [14]. And it generates entity-headers and an entity-body through applying mapping rules provided by the mapping rule provider. If the result has some errors, the REST response generator makes a decision what kind of error occurs and determines a HTTP status code such as 4xx and 5xx depending on the occurred errors [14]. With the help of the mapping rule provider, it obtains mapping rules corresponding to the determined status code and finally generates a response which consists of a HTTP status code, entity-headers, and entity-body representing error type and reason of error.

IV. Mapping Rules

A. Error Handling Rules

The error handling rules are used by the RESTful service response generator when some errors occur in the result returned by the invoked SOAP-based service. In case of sudden stop of the conversion process by exception, the conversion process manager can also use these rules to generate a response message on behalf of the conversion process stopped abruptly. Fig. 5 shows the error handling rules.

HTTP Response Status Code	Response Template
Status Code	Content-type : *content_type_string* Content-length : *length* *Data Representation* to describe error occurrence

Fig. 5 Error Handling Rules

These rules consist of HTTP response status code and response template fields. The HTTP response status code field is used to identify a number of error handling rules. The value of this field can be 4xx to indicate client-side error and 5xx for server-side error.

The response template field has a template consisting of entity-header and entity-body. The entity-header portion of a template can have Content-type and Content-length headers. The entity-body of a template includes descriptions of error types and error reasons. This can be described in a data representation such as XML, RSS, and etc. Special notation marked with %variable can be used in a data representation. The notation is replaced with a proper value by the response generator.

For an example, suppose a response template is as follow.

```
Content-type : application/xml
Content-length:xxx
<error>Error occurred. Error code is %code : %reason.
<\error>
```

In this case, the RESTful service response generator can make a following response. The notation %code is replaced with an appropriate error code.

```
HTTP 1.1 403 Forbidden
Content-type : application/xml
Content-length:xxx
<error> Error occurred. Error code is POL-200: Busy
criteria is not supported </error>
```

B. Service Mapping Rules

Fig. 6 represents rules applicable to transformation from a REST request to a SOAP-based service invocation statement.

HTTP Method	URL Resource Name	Input Parameter Mapping Rule	Result Mapping Rule
GET,PUT, POST, or DELETE	*name*	*Rule references for mapping input parameters*	Content-type : *type_string* Content-length : *length* *Data Representation*

Fig. 6 Service Mapping Rules

A rule consists of HTTP method, URL resource name, input parameter mapping rule, and result mapping rule fields. The HTTP method and the URL resource name fields are used to identify a number of mapping rules. The input parameter mapping rule field has a reference to a rule for input parameter conversion from a REST request to a SOPA-based service invocation shown in Fig. 7. This rule is described in next section. The result mapping rule filed includes a response template and this is also explained in next section.

C. Input Parameter Mapping Rules

In case that input parameters are conveyed in a REST request message, Fig. 7 shows how each of input parameters of a target SOAP-based service is generated. A rule consists of target parameter, target parameter name, target parameter category, and rule reference fields.

Target Parameter	Target Parameter Name	Target Parameter Category	Rule Reference for Mapping Value
parameter[1]	*name*	Basic, Array, or Structure	*reference*
…	…	…	…
parameter[n] (where n is the number of parameters)	*name*	Basic, Array, or Structure	*reference*

Fig. 7 Input Parameter Mapping Rules

The target parameter category field can have one among basic, array, and structure as a value depending on a data type of a target parameter. The rule reference field has a reference to value mapping rule shown in following Fig. 8, 9, and 10.

We consider three cases for the purpose of parameter value mapping; 1) in case that a target parameter has a value of a basic data type such as string, integer, float, and etc., 2) in case that a target parameter has a value of array type, and 3) in case that a target parameter has a value of structure type.

And we also consider how input parameters in a REST request message are delivered. There are two ways; 1) input parameters are delivered in the URL query string of a request message and 2) input parameter are conveyed in the entity-body of a request message.

First, we consider the case that input parameters are delivered in the URL query string of a request message. Fig. 8 shows value mapping rules when a target parameter has a value of a basic data type. A target parameter is mapped from one of tokens of a URL query string.

Target Parameter Data Type	Token of URL Query String
data_type	*token[i].value* *(where 0 < i ≤ the number of tokens)*

Fig. 8 Value Mapping Rules for Basic Data Type from URL Query

Fig. 9 shows value mapping rules when a target parameter has a value of array type. Each array element of a target parameter is mapped from one of tokens of URL query string.

Target Array Data Type	Target Array Length	Target Array Element	Token of URL Query String
data_type	*name*	*element[1]*	*token[i].value (where 0 < i ≤ the number of tokens)*
		…	…
		element[n] (where n is the length of array)	*Token[j].value (where 0 < j ≤ the number of tokens)*

Fig. 9 Value Mapping Rules for Array Type from URL Query

Fig. 10 shows value mapping rules when a target parameter has a value of structure type. Each structure field of a target parameter is mapped by a value mapping rule. The last field of Fig. 10 has a reference to a rule shown in Fig. 8 and 9. This field also has a recursive reference to a rule shown in Fig. 10.

Target Structure Name	Target Structure Field Number	Target Structure Field	Field Name	Field Category	Rule Reference for Mapping Value
name	*number*	*field[1]*	*name*	Basic, Array, or Structure	*reference*
		…	…	…	…
		field[n] (where n is the number of fields)	*name*	Basic, Array, or Structure	*reference*

Fig. 10 Value Mapping Rules for Structure Type from URL Query

Second, we consider the case that Input parameters are delivered in the entity-body of a request message.

Fig. 11 shows value mapping rules when a target parameter has a value of a basic data type. A target parameter is mapped from one of tags of an entity- body. For supporting of nested tags in multi-levels, the notation marked with {} which means repetition zero or more times is used.

Target ParameterData Type	Tag Name
data_type	*{parent_tag_name}.tag_name.value*

Fig. 11 Value Mapping Rules for Basic Data Type from Entity-Body

Fig. 9 shows value mapping rules when a target parameter has a value of array type. Each array element of a target parameter is mapped from one of tags of a message body.

Target Array Data Type	Target Array Length	Target Array Element	Tag Name
data_type	*name*	*element[1]*	*{parent_tag_name}. tag_name[i].value (where 0 < i ≤ the number of tag occurrence)*
		…	…
		element[n] (where n is the length of array)	*{parent_tag_name}. tag_name[j].value (where 0 < j ≤ the number of tag occurrence)*

Fig. 12 Value Mapping Rules for Array Type from Entity-Body

Value mapping rules applied when a target parameter has a value of structure type is the same as one shown in Fig. 10.

For an example, suppose a SOAP-based service has a following API.

1) Operation : getLocation
2) Input message of getLocation
   - requester : string
   - address: string[1..unbounded]
   - requestedAccuracy : integer
   - acceptableAccuracy : integer
3) Output message of getLocation
   - result : LocationInfo structure
4) LocationInfo structure
   - latitude :float
   - longitude: float
   - altitude : float

And we suppose a REST request message is as follows.

```
GET
http://www.example.com/location?address=tel:8601111&addre
ss=tel:8602222&                    requestedAccuracy=
500&acceptableaccuracy=1000 HTTP/1.1
Accept:application/xml
…
```

In this case, the generated SOAP-based service invocation statement is as follow.

```
String requester = null;
String[] address = new String[2];
address[0] = “tel:8601111”;
address[1] = “tel:8601111”;
Integer requestAccuracy = 500;
Integer acceptableAccuracy = 1000;
LocationInfo result = svc.getLocation(requester, address,
requestAccuracy, acceptableAccuracy);
```

D. Result Mapping Rules

The result mapping rule field shown in Fig. 6 includes a response template consisting of entity-header and entity-body. The entity-header portion of a template includes Content-type and Content-length headers. The value of Content-type can be application/xml, application/json, or etc.

The entity-body of a template can be described in a data representation such as XML, JSON, RSS, and etc. The data representation consists of a number of pairs of tag name and value. The tag names are pre-defined in the template and the values are notated with replacement variable marked with %variable. This notation means that the response generator has to substitute the notation with the value of a designated variable.

Return value of invoked service can be categorized into 3 types such as single value, array value, and structure value. In each case, replacement variable marked with %variable is notated differently.

In the first case that the invoked service returns a single value, replacement variable is marked with %result that means direct mapping from the value returned by the invoked service.

For an example, suppose that a response template is as follow.

```
Content-type : application/xml
Content-length:xxx
<TerminalDistance>%result</TerminalDistance>
```

In this case, the response generator can make a following response message.

```
HTTP 1.1 200 OK
Content-type : application/xml
Content-length:xxx
<TerminalDistance>500</TerminalDistance>
```

In the second case that the invoked service returns an array value, replacement variable is marked with %result[i] that means mapping from the value of an element of the returned array. For an example, suppose a response template is as follow.

```
Content-type : application/xml
Content-length:xxx
<TerminalAddress>%result[i]</TerminalAddress>
```

In this case, the response generator can make a following response.

```
HTTP 1.1 200 OK
Content-type : application/xml
Content-length:xxx
<Terminal Address>tel: 8601111</TerminalAddress>
<Terminal Address>tel: 8602222</TerminalAddress>
<Terminal Address>tel: 8603333</TerminalAddress>
```

In the third case of a structure value being returned, replacement variable is marked with %result.fieldName that means mapping from the value of a designated field among fields of a returned structure. For an example, suppose a response template is as follow.

```
Content-type : application/xml
Content-length:xxx
<Terminal Location>
   <latitude>$result.latitude </latitude>
   <longitude>$result.longitude </longitude>
   <altitude>$ result.altitude</altitude>
</TerminalLocation>
```

In this case, the response generator can make a following response.

```
HTTP 1.1 200 OK
Content-type : application/xml
Content-length:xxx
<Terminal Location>
   <latitude>100.23</latitude>
   <longitude>-200.45 </longitude>
   <altitude>85</altitude>
</TerminalLocation>
```

V. Conclusion

This paper showed the Web service mediation method in order to provide existing SOAP-based Web services with RESTful service users. This mediation method provides merits like that the Web service providers need not develop duplicate RESTful services or they don't have to program mediation modules per service.

This paper focused on transformation using mapping rules. But from the viewpoint of service provider, how easy to describe mapping rule is also important. As a next step, we have a plan to design and develop a mapping rule definition tool using user-friendly interface.

Acknowledgment

This research is supported by the IT R&D program of MKE/KEIT of South Korea. [KI002076, Development of Customer Oriented Convergent Service Common Platform Technology based on Network].

References

[1] Liang-Jie Zhang, Jia Zhang, and Hong Cai, Services Computin, Springer, 2007.
[2] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard, "Web Services Architecture", W3C Working Group Note, Feb. 2004 (Available at http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/).
[3] M. Gudgin, M. Hadley, N. Mendelsohn, J-J. Moreau, and H. Nielsen, "SOAP Version 1.2 Part 1: Messaging Framework", W3C Recommendation, Jun. 2003 (Available at http://www.w3.org/TR/2003/REC-soap12-part1-20030624/).
[4] Erik Christensen, Erik Christensen, Greg Meredith, and Sanjiva Weerawarana, "Web Services Description Language (WSDL) 1.1", W3C Note, Mar. 2001 (Available at http://www.w3.org/TR/wsdl).
[5] Luc Clement, Andrew Hately, Claus von Riegen, and Tony Rogers, "UDDI Version 3.0.2", UDDI Specification Technical Committee Draft, Oct. 2004 (Available at http://uddi.org/pubs/uddi-v3.0.2-20041019.pdf).
[6] Leonard Richardson and Sam Ruby, Restful Web Services, O'Reilly Media, 2007.
[7] Alex Rodriguez, "RESTful Web services: The basics", Nov. 2008 , Available at http://www.ibm.com/developerworks/Webservices/library/ws-restful/.
[8] Dave Chappell, Enterprise Service Bus, O'Reilly, Jun. 2004.
[9] Jeff Davies, David Schorow, Samrat Ray, and David Rieber, The Definitive Guide to SOA: Oracle Service Bus, Second Edition, Apress, 2008.
[10] WebSphere Enterprise Service Bus, Available at http://www-01.ibm.com/software/integration/wsesb/.
[11] ORACLE SERVICE BUS, Available at http://www.oracle.com/technologies/soa/docs/service-bus-datasheet.pdf.
[12] Mule ESB, Available at http://www.mulesoft.org/display/MULE2INTRO/Home.
[13] Vordel XML Gateway, Available at http://www.vordel.com/products/vx_gateway/.
[14] R. Fielding, J. Gettys, J. Mogul, and et all, "Hypertext Transfer Protocol -- HTTP/1.1", IETF rfc, Jun. 1999, Available at http://www.w3.org/Protocols/rfc2616/rfc2616.html
[15] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", IETF rfc, Jan. 2005, Available at http://www.ietf.org/rfc/rfc3986.txt