StASOAP: Streaming API for SOAP

Antonio J. Sierra
Department Ing. Sist. Automat,Área de Telemática, University of Sevilla,
C/Camino de los Descubrimientos s/n
Email: antonio@trajano.us.es

ABSTRACT

This paper shows an event-based API for using SOAP over Java platform J2ME, known as Streaming API for SOAP (StASOAP). StASOAP is a streaming and bi-directional API for SOAP over resources-constrained devices based on XML Pull parsing model's extensions. This API uses an event-based XML pull-everything parser. Some uses of SOAP Version 1.2 emphasized the use of a pattern in which multiple message exchanges between two nodes, sender and receiver, as means for conveying remote procedure calls (RPC). Pull-everything, in-situ XML parsers require a complete serialized representation of XML document, to give a more flexible and scalable implementation.

Keywords: SOAP, XML, Pull, J2ME, Java.

1. INTRODUCTION

Two types of interfaces are available for parsing XML documents, tree-based interfaces and event-based interfaces. Tree based XML parser read an entire XML document into an internal tree structure in memory. Each node of the tree represents a piece of data from the original document. It allows an application to navigate and manipulate the parsed data quickly and easily. But a tree-based XML parser can be very memory- and CPU-intensive because it keeps the whole data structure in memory. An event-based XML parser reports parsing events directly to the application. It provides a serial-access mechanism for accessing XML documents. In general, event-based XML parsers are faster and consume less CPU and memory than DOM parser. But, event-based parser allow only serial access to the XML data, we can't go back to an early position or leap ahead to a different position.

Pull-everything, in-situ XML parsers are about as far as you can go, they can do anything. Data-copying XML parsers copy all the information in the parsed XML document into objects, returned to the client. In-situ XML parsers, as much as possible, indicate where data was found in the parsed XML document. In-situ is more powerful than data-copying: if the client doesn't care where the data is, either will do; if the client cares, only in-situ will do. In-situ parsing is a good fit for small-footprint devices. A XML parser's implementation need to mantain a serialized representation (that could be the virtual data source) to move back the cursor, such as is shown Free Cursor Mobility (FCM) [1].

Simple Object Access Protocol (SOAP) Version 1.2 is a lightweight protocol intended for exchanging information decentralized, distributed environment, that uses XML technologies to define an extensible messaging framework providing a message construct that can be exchange over a variety of underlying protocols [3]. However fundamentally SOAP is a stateless, one-way message exchange model, applications can create more complex interaction patterns by combining such one-way exchanges with features provided by an underlying protocol and/or application-specific information. The SOAP is an XML messaging specification that describes a message format along with a set of serialization rules for data types including structured types and arrays. SOAP Version 1.2 can encapsulate remote procedure call functionality using the extensibility and flexibility of XML.

A small-footprint device by definition has an extremely limited amount of memory, and traditional packages are far too large and resource-intensive to work on resources-constrained devices. These are simply too small to be expected to work with packages originally designed for desktop clients and servers. An event-based XML parser uses much less memory than a tree-

based XML parser since it doesn't have to hold the entire document in memory simultaneously. It can process the document in small pieces. This implementation uses an event-based XML pull-eve everything, in-situ parser to give support for a SOAP implementation in a framework of small-footprint.

This paper is organized as follows first we present related work for the same platform. Next, the SOAP protocol and SOAP HTTP binding is shown. Next, we present the detail of the implementation. And finally conclusions are shown.

2. RELATED WORK FOR J2ME

A DOM parser returns a "tree" representation of the XML document. A Push parser calls client's methods with XML events. A Pull parser returns XML events to a client on request. DOM provides APIs that allows access and manipulation of an in-memory using a "tree" representation of the XML document. At first glance this seems like a win for the application developer. However, this perceived simplicity comes at a very high cost, the performance. An implementation for small-footprint devices must not provide any support for DOM, because is generally considered to be too heavy, both in terms of implementation size and runtime memory footprint, to be used on the J2ME platform. In this section we show two implementations for Web Services in the framework of J2ME. Java Specification Request 172 (JSR-172) [6] and kSOAP [7], none of then uses a DOM or like-DOM parser.

kSOAP uses an extremely memory efficient pull J2ME parser, however test realized shown a best memory performance for JSR-172. kSOAP has a class *SoapEnvelope*, with two attributes *public Element []headerIn* (or *headerOut*), also contain another for the Body, `public Object bodyIn/bodyOut`.

To use SAX, a programmer writes handlers, that is objects that implement the various SAX handler APIs, which receive callbacks during the processing of an XML document. The main benefits of this style of XML document processing are that it is efficient, flexible, and relatively low level. An implementation of SOAP for J2ME for Web Services is the JSR-172, which used a subset of *Simple API for XML* (SAX) 2.0 [9] for parsing XML document. JSR-172 proposes an implementation with no server capabilities, which is a strict subset of the JAXP 1.2 specification, which uses SAX 2.0.

A problem of the current implementation is the soap's version. kSOAP and JSR-172 implements SOAP 1.1, and the new SOAP version, 1.2, is not compatible. StASOAP is an implementation of SOAP 1.2, that unlikely to kSOAP and JSR-172 uses the real types, *float* and *double* for data binding.

3. SOAP

SOAP messages are transmitted between applications and may pass through a number of intermediaries as they travel from the initial sender to the ultimate recipient. A SOAP message is one-way transmission between SOAP nodes, from a SOAP sender to a SOAP receiver but SOAP messages are expected to be combined by applications to implement more complex interaction patterns ranging from request/response to multiple. SOAP messages are comprised of an *Envelope* element, with an optional *Header* and a mandatory *Body* child element. The *Envelope* element is the root element for all SOAP messages, identifying the XML as a SOAP message. Child element. A pictorial representation of the SOAP message we can see in Figure 1.

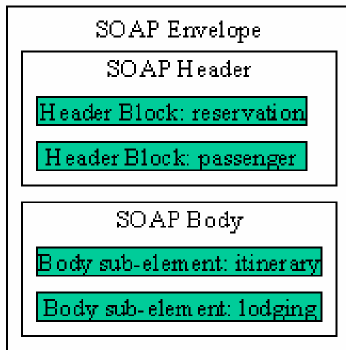

Figure 1: SOAP message.

The header blocks *reservation* and *passenger* must be processed by the nest SOAP intermediary encountered in the message path or, if there is no intermediary, by the ultimate recipient of the message. The *Body* element and its associated child elements, *itinerary* and *lodging*, are intended for exchange of information between the initial SOAP sender and the SOAP node which assumes the role of the ultimate SOAP receiver. The means by which a SOAP node assumes such a role is not defined by the SOAP specification, and is determined as a part of the overall application semantics and associated message flow.

3.1 Header

The *Header* element namespace serves as a container for extensions to SOAP. No extensions are defined by the specification, but user-defined extension services such as transaction support, locale information, authentication, digital signatures, and so forth could all be implemented by placing some information inside the *Header* element. Children of the *Header* element may be annotated with the *mustUnderstand* and/or *actor* attributes.

3.2 Body

The SOAP Body is the mandatory element within the SOAP Envelope, which implies that this is where the main end-to-end information conveyed in a SOAP message must be carried. The SOAP Body provides a mechanism for transmitting information to an ultimate SOAP receiver.

3.3 Fault

The *Fault* element indicates that an error occurred while processing a SOAP request. This only appears in response messages. Child elements are, two mandatory (*Code*, *Reason*), and three optional (*Node*, *Role* and *Detail*).

The *Code* element has one (*Value*) or two (*Value* and *Subcode*) child in order to define a small set of SOAP fault codes covering high level SOAP faults. The *Reason* is a human readable explanation of the fault.

The *Node* element information item is intended to provide information about which SOAP node on the SOAP message path caused the fault to happen. The *Role* element identifies the role the node was operating in at the point the fault occurred. The *Detail* might contain information about a message not containing the proper credentials, a timeout, etc. The presence of the *Detail* element information item has no significance as to which parts of the faulty SOAP message were processed.

3.4 Attributes

Further processing of header blocks and the body depend on the role assumed by the SOAP node for the processing of a given message. SOAP defines the optional *role* attribute that may be present in a header block, which identifies the role played by the intended target of that header block.

In order to ensure that SOAP node do not ignore header blocks which are important to the overall purpose of the application, SOAP header blocks also provide for the additional optional attribute, *mustUnderstand*, which, if "true", means that the SOAP node must process the header with the semantics described in the specification of the header.

SOAP Version 1.2 defines another optional attribute for header blocks, *relay*, which indicates if a header block targeted at a SOAP intermediary must be relayed if it is not processed.

4. TYPE MAPPING

The rules and format of serialization for XML data types are based on the encoding style. XML data types are as defined in the XML Schema Part II-Datatypes [13]. kSOAP serialize type data to mapping to the Java type *Integer*, *Boolean*, *Long*, *String*, *Date*, *Base64* and *Vector*. JSR-172 appends *Short* and *Byte*, and to uses arrays uses the class *TypeVector* that extends *Vector*, to append information about the type supported and information *nillable*. An element with *nillable* attribute set to true for a built-in simple XML data type is mapped to the corresponding Java wrapper class for the Java primitive type. By implementing *KvmSerializable*, a developer can continue to use his own data object in kSOAP. StASOAP appends the type *Qname*, *Double* and *Float*.

5. SERIALIZATION

SOAP defines a set of serialization rules for encoding data types in XML. To implement the serialization at J2ME is different to J2SE. All data is serialized as elements rather than attributes. For simple types such strings, numbers, dates, and so forth, the data types defined in XML Schema Part II-Datatypes [13] are used. For types such as classes or structures, each field in the type is serialized using an element with the same name as the field. For array types, each array element is typically serialized using an element with the same name as the type, although other element names may be used. In both cases, if the field being serialized is itself a structure or an array, then nested elements are used. The top-level element in both the structure case and the array case is namespace qualified. Descendant elements should be unqualified. Serializing data structures, when each field is referred to exactly once, is straightforward. Each field is serialized as an embedded element, a descendant element of the *Body* element, not as an immediate child.

6. SOAP HTTP BINDING

When we use SOAP, we think usually in XML over HTTP. HTTP is an excellent transport for SOAP due to wide use. HTTP is a protocol omnipresent and is a good format of message standard. SOAP defines a binding to the HTTP protocol. This binding describes the relationship between parts of the SOAP request message and various HTTP headers.

In the SOAP 1.2 HTTP binding, the *Content-type* header should be "*application/soap+xml*" instead of "*text+xml*" as in SOAP 1.1.

In the SOAP 1.2 HTTP Binding the *SOAPAction* HTTP header defined in SOAP 1.1 has been removed, and a new HTTP code 427 has been sought from IANA for indicating that its presence is required by the server application. The contents of the former *SOAPAction* HTTP header are now expressed as a value of an (optional) "*action*" parameter of the "*application/soap+xml*" media type that is signaled in the HTTP binding.

SOAP 1.2 provides a finer grained description of use of various 2xx, 3xx, 4xx HTTP status codes. SOAP 1.2 also provides an additional message exchange pattern, which may be used as a part of the HTTP biding that allows the use of HTTP GET for sale and idempotent information retrievals.

The *Content-Length* header for SOAP requests and responses is set to the number of bytes in the body of the request or response.

J2ME uses the interface *HttpConnection* to manage HTTP protocol. With the static methods *open* of the *Connection* class can open HTTP connection. The server and the client have no any information about the connection, so we can uses cookies for identifying the connection, with the *setRequestProperty* of the *HttpConnection.*

7. IMPLEMENTATION

7.1 Introduction

A pull-everything for XML consists of two styles: A low-level cursor API, designed for creating object models and a higher-level event iterator API, designed to be easily extensible access to the serialized representation of the all XML document.

A pull (and pull-everything) XML parser gives parsing control to the programmer by exposing a simple iterator based API and an underlying stream of events. Methods such as *next()* and *hasNext()* allow an application developer to ask for the next event (pull the event) rather than handling the event in a callback. This gives a developer more procedural control over the processing of the XML document. Pull (and pull-everything) XML parser allows the programmer to stop processing the document, skip ahead to sections of the document, and get subsections of the document. With a parser that uses pull- everything parsing, we implement this API to uses SOAP to look for more efficiency in memory, in an event. This API consists of two styles, a low-level cursor API, designed for creating object models and a higher-level event iterator API, designed to be used in pipelines and be easily extensible.

SOAP is an XML message format that can be used as a transport for remote procedure calls (RPC). From a Java server-side perspective, processing an RPC-encoded SOAP message requires the message to be parsed and a reply to be constructed. During the inbound processing the application must identify the proper method to invoke, unmarshal the arguments to the method from XML to Java, invoke the method and marshal the output of the method into the SOAP response envelope.

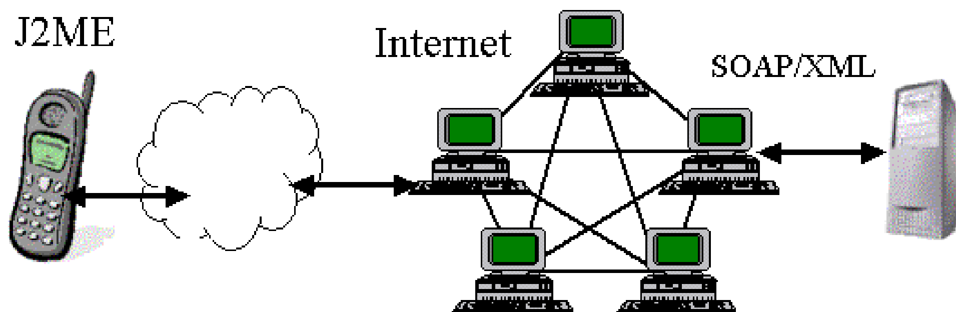

Figure 2: Scenario.

7.2 Implementation's Classes

Under the consideration of resources-constrained devices, the implementation must use less number of class and interfaces. The directory *com.sierra.io* contains the Input/Output operation to read XML data in the different encoding schemes. The directory *com.sierra.xml.fcm* contains the classes related to XML parser.

The directory *com.sierra.xml.soap.fcm* contains the core of soap's implementation. This API uses the interfaces *SOAPFCMReader* and *SOAPFCMWriter*, *SOAPFCMException* and *SOAPFCMConstants*, to management the event.

The directory *com.sierra.xml.soap.rpc* contain information of the XML datatype supported. The directory *com.sierra.xml.soap.transport* contain the class *HttpTransport* to establish the connection.

```
+---com
    \---sierra
        +---io
        |   |   ArrayInputStream.java
        |   |   XMLReader.java
        |   |   XMLWriter.java
        |   |
        |   \---i18n
        |           ReaderUCS4.java
        |           ReaderUTF16.java
        |           ReaderUTF8.java
        |           WriterUCS4.java
        |           WriterUTF16.java
        |           WriterUTF8.java
        |
        \---xml
            |   XMLConstants.java
            |
            +---fcm
            |       Location.java
            |       XMLFCMConstants.java
            |       XMLFCMException.java
            |       XMLFCMReader.java
            |       XMLFCMWriter.java
            |
            +---namespace
            |       NamespaceContext.java
            |       QName.java
            |
            \---soap
                |   ConstantsType.java
                |   SOAPConstants.java
                |   SOAPFCMConstants.java
                |   SoapMessage.java
                |
                +---fcm
                |       SOAPFCMException.java
                |       SOAPFCMReader.java
                |       SOAPFCMWriter.java
                |
                +---rpc
                |       Type.java
                |
                \---transport
                        HttpTransport.java
```

Figure 3: Classes of the implementation.

The class *HttpTransport* contains the method *call* to establish communication to the server. The method call need as parameter the remote soap method to be executed.

The *SoapMessage* class configures the parameter to the method called.

8.2 SOAPFCMWriter

SOAPFCMWriter is used for writing a SOAP document, and is based on *XMLFCMWriter*, to send the XML document. This interface has methods to write well-formed SOAP message, flush o close the output and write qualified names.

SOAPFCMWriter provide all *XMLFCMWriter*'s method, because this class is provided in the constructor the *SOAPFCMWriter* class, and append the methods described in the Table 4.

Method	Description
public void ***writeStartEnvelope()***	Writes the start of SOAP message's envelope to the output. This method writes the prefix. (provided as parameter) and the namespace.
public void ***writeEndEnvelope()***	Writes an envelope's end tag to the output relying on the internal state of the writer to determine the prefix and local name of the envelope.
public void ***writeStartHeader()***	Writes the start tag of header. This method writes the prefix.
public void ***writeEndHeader()***	Writes a header's end tag to the output relying on the internal state of the writer to determine the prefix and local name of the envelope.
public void ***writeStartBody()***	Writes the start tag of body. This method writes the prefix.
public void ***writeEndBody()***	Writes a body's end tag to the output relying on the internal state of the writer to determine the prefix and local name of the envelope.
public void ***setMethodAndParameter()***	This method needs as parameter a *SoapMessage* object, and a *prefix* (if we don't uses the *prefixDefault*).
public void ***flush()***	Write any cached data to the underlying output mechanism.

Figure 4: Methods of *SOAPFCMWriter*.

8.2 SOAPFCMReader

The class *SOAPFCMReader* is based on *XMLFCMReader* and provides methods to get event's information. Similarly to *XMLFCMReader*, *SOAPFCMReader*, has the *next()*, and *hasNext()* methods to obtain different events from the XML data. *SOAPFCMReader* provides more methods associated to the different events obtain.

Event in StSOAP	Description
SOAPFCMConstants.START_SOAP	*public boolean* ***isFaul()***
SOAPFCMConstants.CODE	*public String* ***getCode()***
SOAPFCMConstants.REASON	*public String* ***getReasonText()***
	public String ***getReasonLanguage()***
SOAPFCMConstants.NODE	*public String* ***getNode()***
SOAPFCMConstants.DETAIL	*public String* ***getDetail()***
SOAPFCMConstants.ROLE	*public String* ***getRole()***
SOAPFCMConstants.VALUE	*public String* ***getValue()***
SOAPFCMConstants.SUBCODE	*public boolean* ***hasSubcode()***
	public String ***getSubcode()***
	public int ***getSubcodeCount()***
	public String ***getSubcodeValue(*** *int count)*
SOAPFCMConstants.MUSTUNDERSTAND	*public boolean* ***getMustUnderstand()***
	public boolean ***hasMustUnderstand()***
SOAPFCMConstants.RELAY	*public boolean* ***getRelay()***
	public boolean ***hasRelay()***
SOAPFCMConstants.ENCODINGSTYLE	*public String* ***getEncodingStyle()***
	public String ***hasEncodingStyle()***
SOAPFCMConstants.ROLEATTRIBUTE	*public String* ***getRoleAttribute()***
	public String ***hasRoleAttribute()***

Figure 5: Methods of *SOAPFCMReader*.

The value return for the *call* method is returned with *Object* ***getResponse()***.

8.1 SOAPFCMConstants

SOAPFCMConstants extends the events specified in *XMLFCMConstants*. This interface appends the new events that appear in the following table.

Event in StSOAP	Description
SOAPFCMConstants.START_SOAP	Indicates an event is a start document soap
SOAPFCMConstants.END_SOAP	Indicates an event is an end document soap
SOAPFCMConstants.START_HEADER	Indicates an event is a start Header
SOAPFCMConstants.END_HEADER	Indicates an event is a end Header
SOAPFCMConstants.START_BODY	Indicates an event is a start Body
SOAPFCMConstants.END_BODY	Indicates an event is a end Body
SOAPFCMConstants.START_FAULT	Indicates an event is a start Fault
SOAPFCMConstants.END_FAULT	Indicates an event is a end Fault
SOAPFCMConstants.CODE	Indicates an event is the Fault's Code
SOAPFCMConstants.REASON	Indicates an event is the Fault's Reason
SOAPFCMConstants.NODE	Indicates an event is the Fault's Node
SOAPFCMConstants.DETAIL	Indicates an event is the Fault's Detail
SOAPFCMConstants.VALUE	Indicates an event is the Fault's Value
SOAPFCMConstants.SUBCODE	Indicates an event is the Fault's Subcode
SOAPFCMConstants.MUSTUNDERSTAND	Indicates event is attribute mustUnderstand
SOAPFCMConstants.RELAY	Indicates an event is attribute Relay
SOAPFCMConstants.ROLE	Indicates an event is attribute Role
SOAPFCMConstants.ENCODINGSTYLE	Indicates an event is attribute encodingStyle
SOAPFCMConstants.ROLEATTRIBUTE	Indicates an event is attribute Role

Figure 6: Events of *SOAPFCMConstants*.

All events available through *XMLFCMConstants* can be used in any moment. The SOAP events can be considered as most specialized events.

SOAPFCMConstants, extends events specified in *XMLStreamConstants*, adding new events related to XML soap. *START_SOAP*, *END_SOAP*, START_HEADER, END_HEADER, START_BODY, *END_BODY*, *FAULT*, *END_FAULT*, and the events related with the different element that can be used in a message SOAP 1.2, *CODE*, *REASON*, *NODE*, *DETAIL*, *VALUE*, *SUBCODE*, *MUSTUNDERSTAND*, *RELAY*, *ENCODINGSTYLE*, *ROLEATTRIBUTE*.

8.2 SOAPFCMException

Implementations at J2ME must be less number of exceptions.

Code of SOAPFCMException

```
public class SOAPFCMException extends XMLFCMException {
    /**
     * Default constructor
     */
    public SOAPFCMException (){
        super();
    }
    /**
     * Constructor with a String parameter
     */
    public SOAPFCMException (String info){
        super(info);
    }
}
```

Figure 7: Exception of the implementation, *SOAPFCMException*.

The Fault situation for an StSOAP implementation, as response to the remote call procedure, is treated as an event. kSOAP following uses a class to represent the information related to situations when faults arise in the processing of a message, and uses the *toString()* method to obtain information. Unlikely to kSOAP that uses a class for representing the fault information, and unlikely too the JSR-172 that is treated as an exception (*JAXRPCException* because does not provide the *SOAPFaultException*). However the *SOAPFCMException* is supported is not considered for this point, it is possible throw this exception using the constructor with a *String* parameter, where in the parameter has all information related to Fault situation when we receive the *END_FAULT* event.

8. EXAMPLE

In this section we present XML soap's example of a request and response, to a server that have a database (MySQL). The server to realize request to the database.

9.1 XML SOAP request

```
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body>
      <SOAP-ENV:query>
      <SOAP-ENV:Act xmlns="" xsi:type="xsd:string">
         Autoescuela
      </SOAP-ENV:Act>
      <SOAP-ENV:Pro xmlns="" xsi:type="xsd:string">
         Sevilla
      </SOAP-ENV:Pro>
      <SOAP-ENV:Loc xmlns="" xsi:type="xsd:string">
         Sevilla
      </SOAP-ENV:Loc>
      <SOAP-ENV:Cat xmlns="" xsi:type="xsd:string">
      </SOAP-ENV:Cat>
      <SOAP-ENV:Emp xmlns="" xsi:type="xsd:string">
         Leonesa
      </SOAP-ENV:Emp>
    </SOAP-ENV:query>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 8: Example of a XML SOAP request .

9.2 XML SOAP response

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <SOAP-ENV:Body>
      <ns1:queryResponse>
         <SOAP-ENV:queryReturn
         SOAP-ENV:arrayType="soapenc:string[8]"
         xsi:type="SOAP-ENV:Array">
            <SOAP-ENV:item xsi:type="soapenc:string">
               Autoescuela
            </SOAP-ENV:item>
            <SOAP-ENV:item xsi:type="soapenc:string">
               Sevilla
            </SOAP-ENV:item>
            <SOAP-ENV:item xsi:type="soapenc:string">
               Sevilla
            </SOAP-ENV:item>
            <SOAP-ENV:item xsi:type="soapenc:string">
```

```
                0
            </SOAP-ENV:item>
            <SOAP-ENV:item xsi:type="soapenc:string">
                Leonesa
            </SOAP-ENV:item>
            <SOAP-ENV:item xsi:type="soapenc:string">
                954232506
            </SOAP-ENV:item>
            <SOAP-ENV:item xsi:type="soapenc:string">
                Reina Mercedes, 1
            </SOAP-ENV:item>
            <SOAP-ENV:item xsi:type="soapenc:string">
                166
            </SOAP-ENV:item>
         </SOAP-ENV:queryReturn>
      </SOAP-ENV:queryResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 9: Example of a XML SOAP response .

ACKNOWLEDGEMENT

Thank to Sam Wilmott, the father of Another Fun Language (http://www.wilmott.ca/afl/), for your comments.

CONCLUSION

An event-based XML pull-everything parser is used to implement a SOAP Version 1.2, to use a pattern in which multiple message exchanges between two nodes, sender and receiver, as means for conveying remote procedure calls (RPC). Streaming API for SOAP (StASOAP) is a streaming and bi-directional API for SOAP over resources-constrained devices based on XML Pull parsing model's extensions. Pull-everything, in-situ XML parsers require a complete serialized representation of XML document, to give a more flexible and scalable implementation.

REFERENCES

[1] Antonio J. Sierra, "*A new Paradigm of Parsing XML based on Free Cursor Mobility (FCM)*", Extreme Markup Languages 2005.August 1-5, 2005 Montréal, Canada.
[2] Nilo Mitra, "*SOAP Version 1.2 Part 0: Primer*", w3c Recommendation, 24 June 2003, http://www.w3.org/TR/soap12-part0/.
[3] Martin Gudgin et al., "*SOAP Version 1.2 Part 1: Messaging Framework*", w3c Recommendation, 24 June 2003, http://www.w3.org/TR/2003/REC-soap12-part1-20030624/.
[4] Martin Gudgin et al., "*SOAP Version 1.2 Part 2: Adjuncts*", w3c Recommendation, 24 June 2003, http://www.w3.org/TR/soap12-part2/.
[5] Hugo Haas, et al., "*SOAP Version 1.2:Specification Assertions and Test Collection*", w3c Recommendation, 24 June 2003, http://www.w3.org/TR/soap12-testcollection/.
[6] Jon Ellis and Mark Young, "*Java Specification Requests, JSR-172, J2ME Web Services Specification*", http://jcp.org/en/jsr/detail?id=172.
[7] http://ksoap.objectweb.org/.
[8] Antonio J. Sierra and Antonio Albéndiz, "*Comparative Study of methods of serialization at J2ME*", The 14th IASTED International Conference on Applied Simulation and Modelling ~ASM 2005, Benalmádena, Spain.
[9] D. Megginson et al., "*SAX 2.0: The Simple API for XML*", http://www.saxproject.org.
[10] "Document Object Model", http://www.w3.org/DOM.
[11] Paul V. Biron and Ashok Malhotra, "*XML Schema Part 2: Datatypes*" w3c Recommendation 02 May 2001, http://www.w3.org/TR/2001/REC-xmlschema-2-2001/REC-xmlschema-2-20010502/.
[12] Tim Bray, Jean Paoli et al., "*Extensible Markup Language (XML) 1.0 (Third Edition)*", w3c Recommendation, 4 February 2004, http://www.w3.org/TR/REC-xml.