Template based High Performance ALE-TSOAP Message Communication

Suhyun Kim, Daeyoung Kim, Jongwoo Sung and Tomás Sánchez López
Information and Communications University (ICU)
Daejeon, Republic of Korea
{lure1214, kimd, jwsung, tomas}@icu.ac.kr

Abstract

Recently, the RFID technology has become essential for ubiquitous computing. As the deployment of mega SCM (Supply Chain Management) environments starts at the largest companies in the distribution industry, efforts tend to concentrate on a variety of performance improvements for the RFID middleware aiming to give quality of service for large RFID-data transmission. Web services, cornerstone of the RFID networks, require high performance, security and extensibility. Since SOAP (Simple Object Access Protocol) inherits the poor performance of XML, it is not easy for the RFID middleware to support high performance web services. The ALE (Application Level Event) communication interface from the RFID middleware sends a response message. Its serialization, which includes the conversion of common language runtime objects to the XML documents and streams and packing of this data into a message buffer, has been proven as the bottleneck for SOAP's poor performance. In this paper, we propose ALE-TSOAP, based on ALE templates, that provides an increase in the performance of the RFID middleware when generating response messages. We analyze SOAP messages to classify the various ALE template formats, to design and implement our ALE templates and, finally, to evaluate the performance of the ALE-TSOAP processing. The use of ALE-TSOAP does not change the SOAP protocol or the ALE communication interface of the RFID middleware. Through our experiments, we observed that our approach obtains up to a 197.8% performance gain by only using ALE templates for the serialization of SOAP message.

1. Introduction

RFID (Radio Frequency Identification) technology is one of the most influencing and powerful technologies to realize the ubiquitous computing society. The RFID system [1] is automatic identification technologies. RFID tags can identify a product, animal or person by using radio waves and monitor their physical data in real-time.

Traditionally, the RFID technology research has focused on the development of hardware such as antennas, transponders and single chip RFID tags. Recently, however, the interest has shifted to the development of middleware solutions that can support the RFID technologies integrating them into existing enterprise application systems. The resulting RFID middleware[2] must aggregate and filter the data from RFID tags, and must support the management of tags by using various RFID readers in an Internet based network.

Among all the challenges involving the RFID middleware, its performance is one of the most essential factors. For every service request from clients, the middleware should process hundreds of thousands of tag codes per second. The increase in the concerns about the RFID technology is also becoming more and more coupled with the increase in the RFID standardization activity by EPCglobal; a organization that leads the development of industry-driven standards for the EPC (Electronic Product Code) to support the use of RFID in today's fast-moving, information rich, trading networks. In [1] the EPCglobal defines and describes the EPC Network as a collection of technologies to build an 'Internet of Physical Objects'.

This research was supported by the MIC(Ministry of Information and Communication), Korea, under the ITRC(Information Technology Research Center) support program supervised by the IITA(Institute of Information Technology Advancement)" (IITA-2006-(C1090-0603-0047))

This work was supported by "Development of Korean u-SCM platform and wireless identifying application technology" project of MOCIE, Korea.

In addition to the means for supporting multifarious services as described previously, the RFID middleware in EPC Networks provides the ALE interface [2] as the communication interface through which clients such as EPCIS may obtain filtered, consolidated EPC data from a variety of sources. The client may be new software designed expressly to carry out an EPC-enabled business process, EPCIS (Electronic Product Code Information Service), The ALE interface enables platform independent services using the SOAP protocol to communicate with the EPCIS.

SOAP [3] is a protocol for exchanging XML-based messages over computer networks, normally using HTTP. SOAP forms the foundation layer of the web services stack, providing a basic messaging framework that more abstract layers can build on. SOAP is the most popular protocol to exchange message among heterogeneous systems due to its support for interoperability, language and platform independence, simplicity, extensibility and robustness. SOAP has less performance than binary protocol such as RMI or CORBA due to its inheritance from XML [4]. For this reason, the EPC middleware, which uses SOAP to generate request/response messages, has a limitation of the RFID data it can process in real-time. In concrete, some studies of the SOAP performance [5] show that the encoding of the serialization takes about 90% of the time during the SOAP service. Particularly, the encoding time increases exponentially with the size of the SOAP message. This paper proposes a performance improvement to the generation of SOAP messages by using ALE-based templates for the ALE interface. This approach increases the performance of the RFID middleware without the need of making any change in the SOAP protocol. In particular, we optimize the RFID middleware server-side processing of an ALE-TSOAP request.

This paper is organized as follows. In Section 2, we discuss works related to the SOAP processing optimization. In section, 3, we explain the background of the EPC Network. In section 4, we design the RFID middleware platform. In section 5, we design the templates for the RFID middleware, and discuss the use of caching to increase the performance of the RFID middleware. Section 6 presents our experimental result for the execution time required to process SOAP messages. Finally, in Section 7 we conclude the paper with future work and the summary of our contribution.

2. Related Work

Several studies have proposed versatile approaches for analyzing and optimizing the SOAP performance, comparing SOAP with binary protocols such as JavaRMI and CORBA. In this section, we show how all these studies have proven that SOAP is inefficient in distributed computing due to the requirement of formatting all its messages in ASCII. [4][5] [6].

[4] identifies the sources of inefficiency in the current implementations of SOAP and analyzes the latency performance of several SOAP implementations such as MS SOAP Toolkit +, SOAPRMI, SOAP::Lite, etc. It also compares them with the performance results of binary protocols such as JavaRMI and CORBA. This study shows that the performance of JavaRMI and CORBA are comparable, but that the latencies of SOAPRMI, Apache SOAP, SOAP::Lite, and Microsoft SOAP Toolkit are much worse. One large source of inefficiency in SOAP is the use of multiple system calls to send one logical message. Another source of inefficiency is the XML parsing and formatting time, that incurs in processing penalizations when converting SOAP messages from binary representation into ASCII representation.

In addition to these costs, it should be considered that the ASCII encoded record in each SOAP call is larger than the original binary. This means that SOAP also affects the higher network transmission costs. The performance analysis of SOAP in [5] identified the most critical bottleneck to be the conversion cost of SOAP from binary representation to ASCII representation. The conversion processing takes up to 90% of the time for the end-to-end message processing. To solve the serialization problems of SOAP, [6] proposed bSOAP, which uses a saved message that is sent last so that it can be used as template for later SOAP calls. Each saved message has its own DUT (Data Update Tracking) table, each of whose entries corresponds to a data element in the SOAP message. This approach can bypass the generation time of an equal previously sent message when it is called next. Applications that repeatedly send similar messages achieve performance improvement by applying bSOAP processing.

Because this approach should maintain the DUT table, it becomes a problem to guarantee enough memory space to maintain such table for each template. Shifting is also necessary when the serialized form size of the new data exceeds the field width value in the DUT entry. In this case, the performance of the system is much poorer, meaning that the performance of a 100% value re-serialization without any shifting is better than the shifting worst case.

The limitations of the system performance in [6] are more evident in EPC networks, where most of the RFID middleware generates different structure and size messages in the SOAP protocol used by the ALE communication interface.

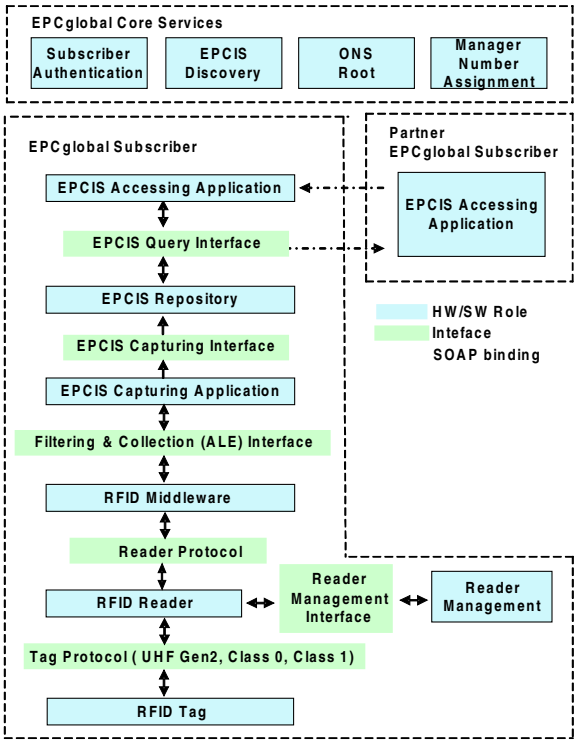

Fig. 1 Communication interface using SOAP protocol in EPC Network Architectures

In order to increase the performance of the DUT table, [6] also proposes to "steal" memory space when only a few contents require modification in the data fields of the table after the serialization. In this case, it is profitable to adjust the data fields of the DUT table by stealing memory space from neighbor fields. However, this approach does not provide much benefit to the RFID middleware in EPC networks. RFID middleware generate an Event Cycle Report which is a response sent to the ALE client at the conclusion of an event cycle. Since most of all the SOAP messages, such as the Event Cycle Report generated every event cycle, have different XML tree structures with different contents, the expansion and shrinking of tags in the XML tree structures would always be necessary. Also since the DUT table can not exceed the maximum size of the predefined memory space, DUT tables for Event Cycle Report templates should be assigned enough space in the initial stage of the memory allocation to store the RFID data in the data fields of the RFID middleware. It is clear that we should avoid this way for optimized SOAP processing. since this kind of memory allocation before starting the SOAP service calls wastes memory space.

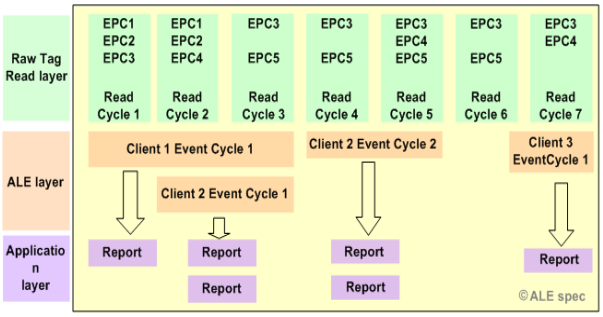

Fig.2 Event data processing of ALE communication interface

3. SOAP based ALE Services in EPC Networks

The EPC Network (Figure 1) architecture consists of an infrastructure which aggregates raw EPC data in the physical layer, a RFID middleware (ALE) which performs filtering and collection (grouping) of EPC data, and a repository and business gateway (EPCIS) which provides application services using EPC business data. All the components in the EPC Network architecture intercommunicate through interfaces such as tag protocols, the Reader protocol, the ALE interface, the EPCIS Capturing application and the EPCIS Query interface. However, the EPC network suffers from huge data processing at each component since the RFID data is periodically returned to the upper layers every read cycle and event cycle. The event cycle is an interval of time over which an ALE server carries out interactions with one or more Readers on behalf of an ALE client. The read cycle is communication unit of interaction with a Reader and represents iteration of the RF protocol used to communicate with RFID tags. Additionally, the fact that all the interfaces are expressed in XML and most of the communication in the network is done via the SOAP protocol pose additional overhead limitations.

The RFID middleware serves service requests from the EPCIS through the ALE interface over HTTP. When an EPCIS client requests a service to an ALE server, it sends an Event Cycle Spec within the request. The Event Cycle Spec holds information about the service start/stop time, the event/read cycle period, the destination URI of the response message, a list of readers or logical readers which is an ALE client uses to refer to a physical reader with multi-antenna, etc.

The RFID middleware (ALE) of the EPC Network processes the raw EPC data that travels from the reader layer to the application layer (Fig. 2). When the ALE server calls

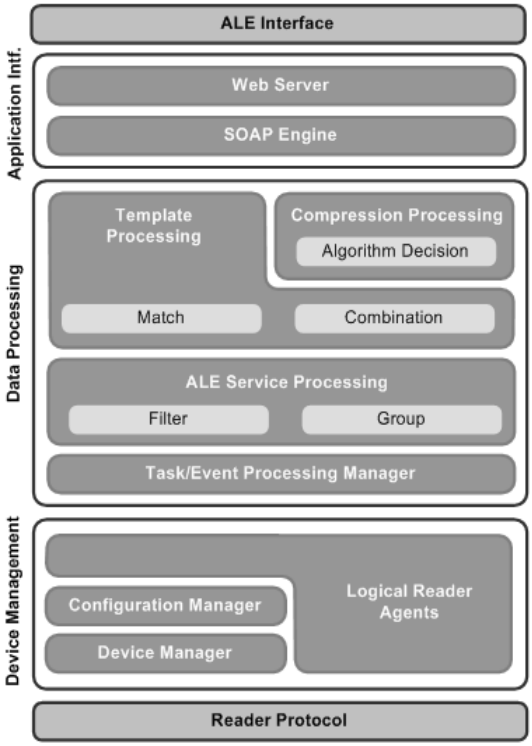

Fig.3 Template based ALE-TSOAP Middleware Architecture

an ALE service, it first analyzes the Event Cycle Spec, and then starts to receive EPC data from data sources such as an RFID reader. This data is accumulated in every read cycle, filtered by removing duplicated or irrelevant EPC data and organized in groups. Finally, the middleware generates an Event Cycle Report in the form of a SOAP message.

The RFID middleware must contain characteristics typical of the RFID networks such as real-time, repetitiveness of frequent service calls, mass data processing or distributed environment. To support these characteristics, the ALE interface provides component independence for the EPC data acquisition, filtering and business applications. This way, the RFID middleware can maintain its internal data processing by aggregating RFID data from logical readers independently without being affected by the physical devices. Thus this independence separates the infrastructure in the physical, service and application layers.

4. Design of a Template-based RFID Middleware Architecture

This paper proposes the light-weight ALE-TSOAP RFID middleware architecture. ALE-TSOAP is an ALE Template based SOAP processing scheme that enables a high performance RFID middleware by using a Template Processing component and a Compression Processing component in order to overcome the poor performance of SOAP.

Our ALE design is a layered structure formed by a Device Management layer, a Data Processing layer and an Application Interface layer that provide independent data.

The Device Management layer aggregates EPC data every read cycle through devices such as RFID readers by sending requests to the Event Processing Manager and transferring periodically the EPC data to the Data Processing layer. The Device Management, consisting of the Logical Reader Agents, the Configuration Management and the Device Manager, must hold a reader ID, a sensor ID and a device profile ID in order to read EPC data. The device profile contains the reader name, antenna and the protocol of the physical reader devices. The Configuration Management should also have the MAC and IP addresses of each device, and their related public/private keys for security purposes. A reader with more than a single antenna recognizes each antenna as an independent device, called a Logical Reader.

The Logical Reader Agent plays the role of a reader adapter and driver instance. The Data Processing layer processes service requests called by the Application Interface layer and responds an Event Cycle Report message for every request. The Data Processing layer consists of the Event/Task Process Manager, the ALE Service Processing, the Template Processing and the Compression Processing modules. The Event Process Manager produces event cycles by using the Event Cycle Spec sent from the EPCIS. The Event Process Manager accumulates reading data from a collection of Logical Readers during every event cycle.

The ALE Service Processing component filters the EPC data returned from the Event Process Manager by using EPC code patterns, time and logical reader, etc. as defined in the Event Cycle Spec, making separate groups of classified reports. The Template Processing module carries out a new approach using ALE templates WSDL based to generate Event Cycle Spec and Event Cycle Report between a RFID middleware and an EPCIS service. The Template Processing component aggregates reports returned from the ALE Service Processing during the Event Cycle and translates them into SOAP messages. This way, the ALE-TSOAP middleware bypasses the serialization of SOAP messages using the ALE-based Template Processing.

The RFID middleware finally generates a complete Event Cycle Report and sends it to the requester, the EPCIS, through the ALE server. Furthermore, if the size of the SOAP message that includes the Event Cycle Report is big enough for compression, compression is applied. The Compression

Processing component dynamically chooses the optimum compression algorithm by the size of the SOAP messages in run-time, such as gzip and XMill, to reduce the transmission size of the response message. The compression technology is essential in order to deal with huge amounts of RFID data especially that included in the Event Cycle Reports.

When an EPCIS calls the RFID middleware through its ALE Interface over HTTP, SOAP messages are sent by the web server to the SOAP engine. If the SOAP engine finds the same service in its own service list first, it sends the message to the Application Interface layer. When the Application Interface layer gets the service, it sends back a response message to the client accepting the request. Finally, when the Application Interface layer starts the service, it calls the Data Processing Layer and waits for the the Event Cycle Report message returning from it.

5. ALE-TSOAP Optimization Scheme using ALE Templates

This paper bypasses the full serialization of SOAP messages to overcome the serialization problem where 90% of the processing time in SOAP service calls is used for message serialization. Our RFID middleware design makes use of ALE-TSOAP to avoid this problem, predefining various types of ALE templates and dynamically choosing the appropriate one. The selection is based on the services that are called to generate SOAP formatted response, and the final complete SOAP message may be a combination of multiple ALE templates together. Fig. 4 presents the RFID middleware processing EPC data. The ALE middleware reads EPC tags, filters their data and makes groups of reports' collections. Finally, the middleware generates a response message for the service called from a client or a third party.

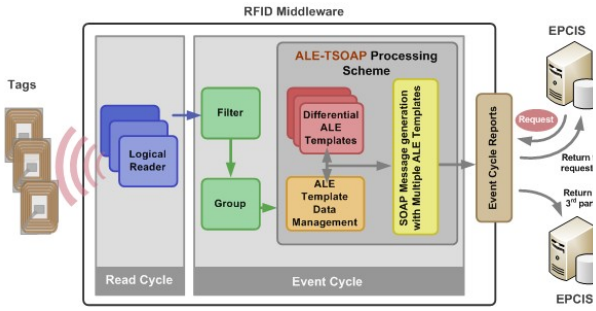

Fig.4 Event Data Processing of ALE Middleware

In this section, we explore the lightweight ALE-TSOAP scheme for SOAP optimization. ALE-TSOAP is based in the use of differential-ALE Templates, ALE TDM (Template Data Management) tables and SOAP message generation with multiple ALE Templates on the RFID server side in order to generate Event Cycle Report messages.

5.1 Differential-ALE Template Design based on the ALE interface

The structure of a SOAP message is fixed when the service is described via WSDL. SOAP is represented in XML elements and attributes, providing a text-based means to describe tree-based structured information in the form of tags and actual values. Regarding the EPC Network and Web Services, the EPCglobal describes the ALE interface as a WSDL with an Event Cycle Spec and an Event Cycle Report schema. Using WSDL for the ALE interface, we design ALE templates in which the tags are a static part of the SOAP messages but the attributes are dynamic.

We started with the WSDL service interface specification that describes the communications between the ALE server and the EPCIS. We then designed the structure of the response messages based on their corresponding XML schema definition. Fig.5 shows the XML schema that we are going to use for the Event Cycle Reports. There is more than one report element in an Event Cycle Report. We analyzed the service calls to the ALE interface and the structure of the response messages to classify them before sending them outside the ALE middleware. Response messages are SOAP message resulting from adding the HTTP protocol header to the service requests. Each SOAP message has information about the SOAP processing instructions and an Event Cycle Report with multiple reports as the result of the particular service that was called.

The ALE template classification derived from the response messages for the ALE interface services is the following;

•Message Fully Matching;

The whole structure of message and its contents is fixed. We define an ALE template of the SOAP header for the SOAP processing instruction; we must cache the serialized SOAP template to avoid duplicated serialization process for its later use.

•Message Contents Matching;

The whole structure and size of the message is fixed, but some of its contents are variable. We define ALE templates

of the SOAP body to add the actual value of variable tags according to a result of services called in Body of SOAP message and update serialized actual value of the only changed contents. If the number of characters in the updated serialized contents is bigger than the field size of the predefined contents, the field size should be adjusted by more than the size of the updated serialized contents. However, the sizes of the variable fields in the response messages are always fixed and so we can make them as templates.

•Message Structure Matching;

The structure and size of the SOAP message should be changed and extended. We define ALE templates of the SOAP body to add both new EPC code tag elements and the actual value of each EPC code. Although the EPC code tag elements and the actual value of the EPC codes have fixed size in the SOAP messages, the structure and size of those messages should be extended to additionally add the number of tags and their contents according to the service results after every event cycle is finished. It is the processing time to adjust the structure of a variable field which dictates the performance improvement of the RFID middleware.

We discussed three classifications for the SOAP messages in the ALE server, giving a much more efficient way to generate response message using different ALE templates. We have found these cases partially appropriate for the Event Cycle Report SOAP messages. The Message Fully Matching classification can produce templates for SOAP headers since they always have the same basic XML and SOAP supporting information. However, the main purpose of the response message in the RFID middleware is to provide EPC information as business event data to the EPCIS as a result of a requested service. The Message Contents Matching and the Message Structure Matching classifications can make a template for the SOAP Body containing reports filled with the list and the total number of EPC codes. The tag field elements presenting the EPC code list must be branched out new child node of XML tree structure, as much as the number of EPC codes in the EPC list returned as the service result. We must also require additional fields for the tag count element and the tag list element according to the service result in every event cycle. The Message Structure Matching classification is useful in the case of expansion of the SOAP messages structure for the additional fields of tag count and tag list elements. Since we can know how many tags are necessary after every event cycle is finished, we can calculate the exact chunk size required for them and reserve the necessary memory space.

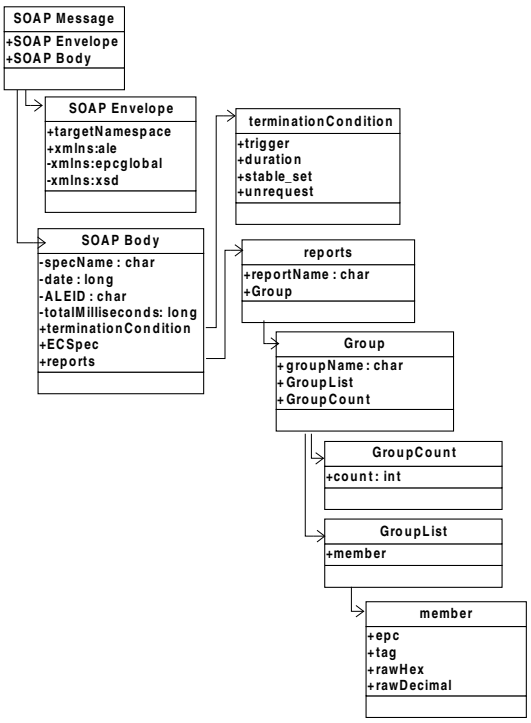

Fig. 5 XML Schema of an Event Cycle Report

Using ALE templates allows avoiding the full serialization of SOAP message by only serializing the contents that are changed, and so reduce the generation time of Event Cycle Report messages and, hence, SOAP generation time, by reusing cached SOAP header templates. Following, we will introduce the ALE templates data management for ALE-TSOAP, that allows maintaining the internal data of the ALE templates while guaranteeing the memory space for caching.

5.2. ALE Template Data Management for ALE-TSOAP

As explained previously, our RFID middleware design dynamically chooses ALE templates from a pool of predefined ALE templates in order to generate response messages. It then fills out the EPC data results for the requested service in the form of serialized contents in each of ALE template. Finally, it combines these ALE templates after applying a differential serialization and makes a complete SOAP.

hen a service is called, the RFID middleware determines whether any of the ALE template messages can be partially reused. To do this, the middleware goes through the ALE TDM (ALE Template Data Management) tables looking at the information about the same service type. Each of the ALE templates has its own ALE TDM table and

continuously maintains its updated status data during service running.

An ALE TDM table contains the following fields.

•**Data Field ID**: a field key to identify additional tag elements and variable content values.

•**Data Field**: a field name for additional tag elements and variable content values.

•**Data Type**: a field data type for additional tag elements and variable content values.

•**Usable Field**: whether cached data exists in the case of tag elements and fixed content values.

•**Change**: whether cached data should change in comparison with the current service result

•**Serialized Length**: the number of characters in the message for the new serialized data

Each result message should be cached in the format of an ALE template after differential serialization and before sending in order to make it available for the next service call. The value of its change field in the ALE TDM table field is set to TRUE and the status of the ALE template is updated. When the same service is called for the second time, the RFID middleware checks the change field in the ALE TDM table and reuses the cached messages in order to avoid full serialization. This way, the response message generator in the ALE server generates a complete SOAP message from each ALE template, using the data ID, data type, usable and change fields in the ALE TDM.

5.3. SOAP Message generation with Multiple ALE Templates

The RFID middleware has a response message generator in the ALE server that creates SOAP message using ALE templates. These templates are created after a differential serialization process that combines dynamically chosen templates among a collection of ALE templates. The response message generator considers the repetition of tag elements and combines SOAP header template messages and SOAP body template messages to finally generate a complete SOAP message. The only difference between this architecture and the original ALE interface in the RFID middleware is thus the response message generator that is replaced by ours which uses ALE templates after differential serialization.

The ALE-TSOAP response message generator is a component that communicates with what we call the "Matching Engine." The Matching Engine processes a variety of ALE templates required for the differential serialization by crawling the repeated XML structures using an automaton. This component dynamically generates an automaton from the response message's XML schema and uses ALE-TSOAP to link the created automaton and the ALE template objects. Finally it tries to match the linked ALE template objects with the service program and if it succeeds, it invokes the differential serialization in order to give the linked ALE template objects the actual values of the service after every event cycle is finished.

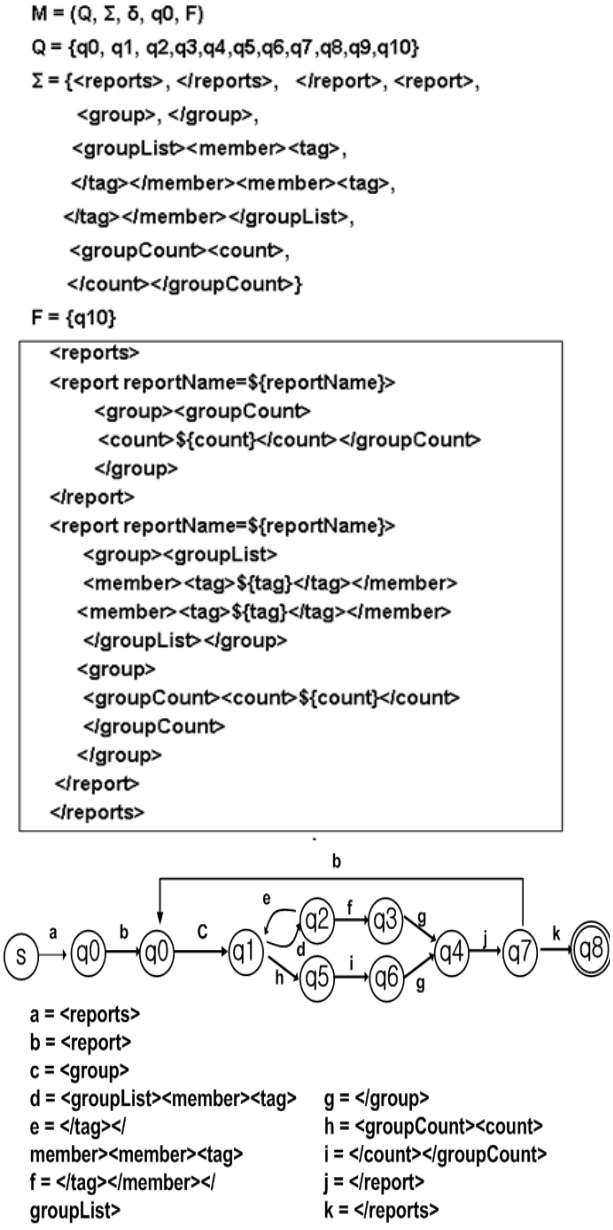

Fig. 6 Automata of combination of report templates to generate complete SOAP Message

```
<reports>{
   FOR $reportName IN ECREPORT
      FOR $groupName || $tag || $count IN REPORT
       IF $reportName
          <report reportName= $reportName >
          IF $groupName
          <group name=$groupName>
          ELSE  <group>
          FOR $tag IN REPORT
             <groupList>
             <member><tag>$tag</tag></member>
             </groupList>
          IF $count IN REPORT
             <groupCount>$count</groupCount>
             </group>
        </report>
}</reports>
```

Fig.7 Pseudo-code for combination of report templates according to the reportName element in an Event Cycle Report

The automaton consists of a fixed state and a variable state. The fixed state contains the tag sequence that is not changed, such as the XML prolog, start and end tags defined by the XML namespace, and constant values of the contents. The variable state contains the tags sequence that is changed, such as the variable content values and additional tags extending XML structure. While creating a new automaton, the Matching Engine collects information about the variable states and creates an ALE TDM table for maintaining them.

The repeated structure of tags element in a SOAP message is required for those services that have two or more pieces of the same ALE template with different contents. This approach is shown by the definition of automaton in Fig. 6. For example, let Q be a finite set of internal states for the fixed part, Σ a finite set of states for the input symbols, δ a transition function to determine the next state and F the set of the final states.

We want to generate a report tag which consists of two report tags. The each report tag has its own child tags hierarchically in the response message. Each report tags should create its child tags step by step and independently. Our goal is to avoid this repeated step by using ALE templates. Our approach is to fill out the report related elements in the report template with the new contents of the variable fields that come as a result of the particular service that was called. Finally we can create a higher reports tag with the combination of multiple report templates. Before making a complete SOAP message, we must cache this report template message for its use in a next service call. Figure 7 presents the pseudo code for creation of reports tags using report templates into an Event Cycle Report.

6. Implementation and Performance Evaluation

In this section we illustrate the operation of ALE-TSOAP and demonstrate the whole process with a simple application. In our experiments we use and open source RFID middleware software, the ALE Server 1.0.4, released by Logicalloy [11] as part of their Web Service suite. The software uses Codehause XFire, a java SOAP framework that supports the most important Web Service standards such as SOAP and WSDL. We used Java 1.4 to test our ALE-TSOAP application on a Jetty Web Server, using Hsql DB and the dom4j library for XML, providing full support for DOM, SAX and JAXP. The application was tested on a Windows XP, 2.40 GHz machine with 512 MB of RAM.

We implemented an alternative response message generator in order to support ALE-TSOAP, that is used in the ALE server when subscribe and immediate/poll services are called in the ALE interface. To experiment with the response message generator, we also implemented a simple application which yields SOAP messages with a payload of a string defined (name, ECSpec) with 100 EPC tags. The purpose of

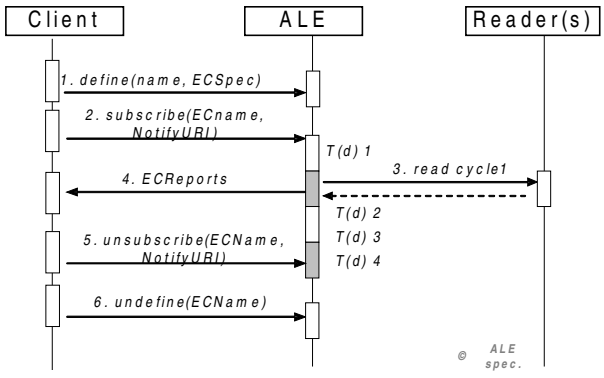

Fig. 8 A subscribe method when a client ask to ALE server to start a service

Table I. The comparison result of processing time for an Event Cycle Response message with 100tags

Codehause XFire	ALE-TSOAP
38.75	19.5

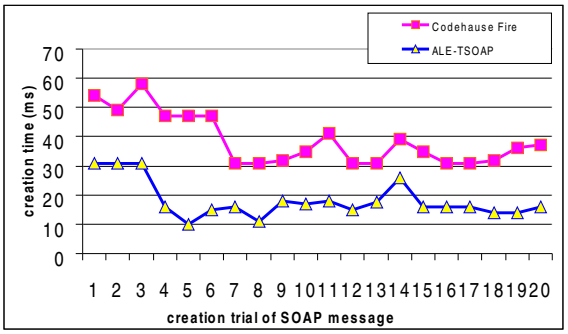

Figure 9. The experiment result of creation time of SOAP message between Codehause Fire and ALE-TSOAP.

the experiment is to compare the performance of the ALE-TSOAP with the performance of the original SOAP (Codehause XFire) regarding the generation time of the SOAP response messages, especially for the ECReports (Event Cycle Reprots) method (Fig. 8). For the performance metric, we used the total number of tags instead of the size of the SOAP messages because even when two messages have the same XML tree structure they may have different sizes due to white spaces such as carried returns, new lines, tab and a blank space. At first, our study focused on the ALE-TSOAP after the differential serialization since the most critical bottleneck of SOAP is the serialization of the XML files. We saved to files the SOAP response messages for each operation in order to analyze each of their generation times according to the total number of EPC tags. Results in Fig.9 show that our ALE-TSOAP boosts the performance of the ALE server when compared with the original SOAP, Codehause XFire. It presents the average time for SOAP processing for 20 trials for both the Codehause Fire and ALE-TSOAP.

We show evaluation results in Table I. When the Codehause XFire in the ALE server generates response messages with 100 tags, the time taken before file I/O grows up to 38.75 ms. When using the ALE-TSOAP, however, it only takes 19.5 ms. Therefore the performance of the ALE-TSOAP is better than that of the Codehause XFire by a 197.8%.

7. Conclusion

This paper aims to solve the problem of the poor performance of the SOAP serialization of XML messages in the RFID middleware. We implemented the light-weight ALE-TSOAP architecture based on ALE templates that applies differential serialization of SOAP messages in the EPC Network. We defined ALE templates to used them with WSDL and the XML schema of Event Cycle Reports, mapped the ALE templates with the ALE TDM tables and combined pieces of ALE template messages to generate complete SOAP messages. Our study compares the performance of our ALE-TSOAP, based on ALE templates, with the performance of the original SOAP, Codehause XFire, regarding the generation time for response messages. The evaluation result presents a 197.8% performance improvement of the ALE server when using ALE-TSOAP.

For our future work, we plan to improve and enhance ALE-TSOAP using caching of ALE template messages. First, we will take advantage of the fact that a major portion of the bottleneck is the serialization time. By using caching, the performance of the RFID middleware could improve due to the reduction of the system calls processed by SOAP.

Reference

[1] Ken Traub,Greg Allgair , “The EPCglobal Architecure Framwork” ,EPCglobal Final Version of 1 July 2005,

[2] EPCglobal, ”The Application Level Events (ALE) Specification, Version 1.1 “EPCglobal ,13 July 2006

[3] W3C, "SOAP Version 1.2" , June, 2004

[4] D. Davis and M. Parashar, “Latency Performance of SOAP Implementations”, Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, pages 407-412, 2002.

[5] K. Chiu, M. Govindaraju, and R. Bramley, “Investigating the Limits of SOAP Performance for Scientific Computing”, Indiana University. Accepted for publication in the Proceedings of HPDC 2002.

[6] Abu-Ghazaleh, N., Lewis, M.J., Govindaraju, M., "Differential serialization for optimized SOAP performance," 13th IEEE International Symposium on High performance Distributed Computing, pp. 55-64 , June 2004

[7] K. Devaram and D. Andresen, “SOAP Optimization via Client-Side Caching”, in the Proceedings of the First International Conference on Web Services (ICWS'03), pages 520-524, LasVegas, NV, June 2003.

[8] F. E. Bustamante, G. Eisenhauer, K. Schwan, and P. Widener, “Efficient wire formats for high performance

computing". In Proceedings of the 2000 conference on Supercomputing, 2000.

[9] K. Chiu, M. Govindaraju, and R. Bramley, "Investigating the Limits of SOAP Performance for Scientific Computing", HPDC-11, pages 246-254, Edinburgh, Scotland, July 23-26, 2002.

[10] Daniel Andresen , David Sexton, Kiran Devaram, Venkatesh Prasad Ranganath, "LYE: a high-performance caching SOAP implementation" ICPP'04, pages 143- 150 vol.1, Montreal, Canada, Aug. 2004

[11] ALE Server 1.0 Final Released 2006-11-19 , logicalloy RFID system, http://sourceforge.net/projects/logicalloy/