Efficient SOAP Message Exchange and Evaluation Through XML Similarity

Ernesto Damiani
DTI - Università degli Studi di Milano
via Bramante, 65
26013 Crema (CR), Italy
damiani@dti.unimi.it

Stefania Marrara
DTI - Università degli Studi di Milano
via Bramante, 65
26013 Crema (CR), Italy
marrara@dti.unimi.it

ABSTRACT

This paper investigates the possibility of using an XML tree similarity based approach in order to individuate similar SOAP messages, that can be aggregated by the Web Service Provider in a single message before checking it according to WS-Policy, saving time and computational costs.

Categories and Subject Descriptors

D.4.6, K.4.2 [**Security and Protection**]: [Cryptographic control]

General Terms

Security, Performance

Keywords

Web-Services, SOAP messages, XML Encryption, XML similarity

1. INTRODUCTION

In *Services-Oriented Architectures* (SOA) functionality is grouped around business processes and packaged as interoperable services, which communicate using the SOAP protocol. SOA technology enables more dynamic, loosely-coupled and asynchronous interactions between inter-domain applications, compared to traditional approaches [2, 1, 3]. Since SOAP messages may carry vital business information, their integrity and confidentiality needs to be protected and SOAP Message security assurance is a challenging part of SOA integration. Using SOAP header to host security related information is a time-honored technique (see [7]); but while protecting messages confidentiality is of paramount importance, it is often desirable to encrypt only parts of a message that is being sent from one entity to another, so that intermediate nodes between the two entities can process the message appropriately. For example, a purchase order service may contain a customer's credit card details, which must be accessible to authorized parties only. However other information in the same message, such as product description and purchase quantity, is provided as clear text in order to be accessible by all organizational routing services that need to process the purchase order message. SOAP is based on XML and therefore inherits all XML's verbosity disadvantages. When there are many transactions involving similar messages, one by one differential encryption, decryption and signature checking of SOAP messages can generate a very large amount of work overload for the service requester and provider. In the work [10], the authors propose a multicast protocol to aggregate messages on the basis of XML tree similarity in order to reduce the transmission traffic in low bandwidth environments. However, while a great effort has been made to optimize SOAP performances in transmission, less attention has been given to the problem of reducing the overhead of applying security policy rules to a large number of messages to be transmitted. This paper investigates the possibility of using a similarity-based approach in order to identify similar SOAP messages to be aggregated in a single message at the sender side before performing encryption and signing. Such an aggregation would also support collective checking of WS-Policy conditions at the receiver side.

SWS'08, October 31, 2008, Fairfax, Virginia, USA.

1.1 Related Work

Recently several solutions have been proposed to improve SOAP performance, either using binary encoding for XML (as opposed to text encoding) [20, 12], caching (at the client side by increasing the locality of objects) [11], compression (by reducing the size of XML payload) [13] or optimizing SOAP run-time implementation (e.g. by efficient optimization of the kernel).

Several techniques exist for comparing XML data items [8, 23]. Some of them try to find the least edit-distance between XML data items A and B by identifying the number of edit operation required to turn A into B. Many papers are available on structure-based similarity of semi-structured documents [19]; structure can also be used to find the similarity between XML DTDs or Schemata [18]. The paper [4] summarizes three approaches on structure similarity: *Tag Similarity*, *Tree Edit Distance* (TED), and *Fourier Transform Similarity*. In [5] a fast approach is proposed using some properties of trees (such as height, degree, label histograms, etc.) to assess their similarity. The work [9] proposes an approach to evaluate the similarity between complex patterns. In this case the similarity between two simple patterns of the same type is an aggregation of the similarity of structure and the similarity of measurement (content).

2. WEB SERVICES

Web services are based on the interaction of three types of entities: the service requestor, the service provider and the service registry. Generally speaking, the service provider advertises its services in a service registry. The service requestor finds a suitable service from the service registry, and subsequently interacts with the associated service provider. This architectural model is shown in Figure 1. In order to implement web services based on this architecture, three core building blocks are used: SOAP [21], WSDL [22] and UDDI [17]. SOAP was designed as a uniform, unidirectional messaging mechanism to transport XML messages from one node to another. It is a protocol that underlies all interactions between web services. WSDL is an XML-based interface description language used to describe services in a standardized way. UDDI is used as a service registry to support a standard way of publishing and locating services.

2.1 Web Services Security

We now give a short summary of key security standards adopted for securing web services, in useful as basis of the comprehension of this work.

2.1.1 WS Security

The WS-Security specification [16] defines a common format for securing SOAP messages using XML Encryption and XML Signatures to protect message confidentiality and integrity. It also provides a way of passing security tokens, such as X.509 certificates [16] or Kerberos tickets [16], through SOAP headers. Offering message-level security, WS-Security is instrumental in providing end-to-end web services security as each message can be encrypted or signed independently, and thus self-protected. The SOAP header block is used for attaching security related information targeted at a specific recipient in the form of a SOAP actor/role. Conversely, on the receiver side or Web Service provider, a SOAP envelope is accepted as valid and passed to the application if its policy is satisfied for this envelope. This technique is in contrast to transport-level security which provides point-to-point security through, for example, a secure channel established using the SSL/TLS protocol (see [14] for a detailed description of differences). Although message-level security provides finer granularity than transport-level security in terms of selective message protection, this granularity potentially causes significant performance issues. This is because each message needs to be processed separately and different security tokens may be used within the same message or from message to message.

2.1.2 SAML

This standard defines methods for specifying trust assertions in XML [15]. These methods enable *portable trust* in the sense that assertions applied to an individual are attached to a message and they can be transported from one point to another with the message. SAML assertions take the form of authentication, authorization or attributes of entities. The SAML authorization assertion request/response protocol is usually run between a *Policy Enforcement Point* (PEP) and a Policy Decision Point (PDP), typically with to support enforcement of XACML policies [16]. It is also often used by a PEP to request attribute assertions from a *Policy information Point* (PIP).

2.1.3 WS-Policy

WS-Security describes how to attach signature and encryption headers to SOAP messages as well as how to attach security tokens. While WS-Security provides a framework to secure a SOAP message using existing techniques (e.g. encryption, signature), WS-Policy and WS-SecurePolicy describe the capabilities and constraints of the security (and other business) policies on intermediaries and endpoints (e.g. required security tokens, supported encryption algorithms). For example, a service provider may only accept a X.509 security token which can be described using the declarative syntax of WS-Policy and WS-SecurePolicy.

3. SMP OVERVIEW

The basis of our approach is [10] which proposes a SOAP multicast technique, called *Similarity-based Multicast Protocol* (SMP), which takes into account the similarity of SOAP messages. SMP was designed to deal with SOAP performance issues by exploiting the similar structure of SOAP messages. The goal is to reduce the total traffic generated over a network when sending SOAP responses from servers to clients. SMP allows similar SOAP messages that share some parts of the SOAP template to be sent as a single customized SMP message instead of being sent as multiple copies. Clients' addresses are represented as strings and stored in the SMP header, which is encapsulated inside the SOAP message body. The SMP body is also embedded inside the SOAP message body.
There are two sections in the SMP body:

1. the *Common* section containing common values and structures of all messages addressed to clients encoded in the SMP header;

2. the *Distinctive* section containing individual different parts for each response message.

The outermost envelope is referred to as an SMP message. The destination of an SMP message, which is specified in the SOAP header, is the next router in a network when the message is forwarded to all clients given in its SMP header. Figure 2 explains the operations of SMP through a sample network of 1 sender and 2 receivers. Two SMP messages are sent out from the source s to its next-hop router r_1. At r_1, the SMP message header is parsed to find out what clients the message is addressed to and the two messages are divided and sent to their final destination. Despite its advantage of saving network traffic, SMP has a remarkable disadvantage: it uses a conventional routing protocol (OSPF) to deliver messages to clients. Since OSPF uses Dijkstra's algorithm, SMP messages are routed along their shortest paths to destinations. Two nodes of a network are often connected with multiple paths. Therefore, sending messages just along least hop paths does not maximize the saving of traffic resulted from the similarity of messages. In addition, SMP has a user-configured time frame. During this time period, outgoing SOAP response messages will be lined in a queue if their similarity level falls within a threshold limit. When a new request message arrives at the server, the server generates its corresponding SOAP response message and computes its similarity against existing on-queue messages. If the computed similarity satisfies the threshold then it is inserted into the queue. If not, the messages that already reside in the queue are sent out as an aggregated SMP message. As

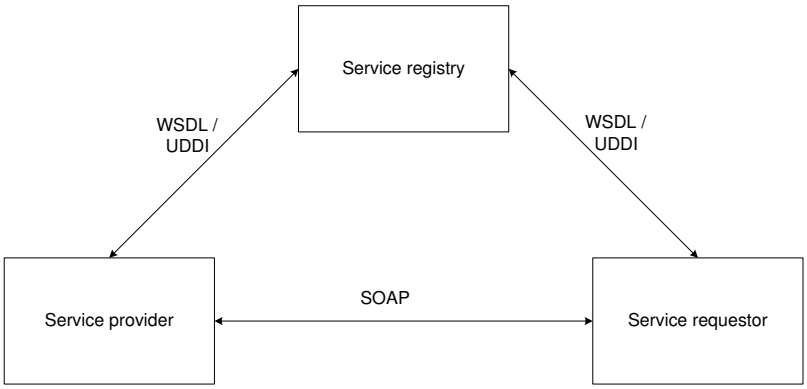

Figure 1: The basic Web Services architecture

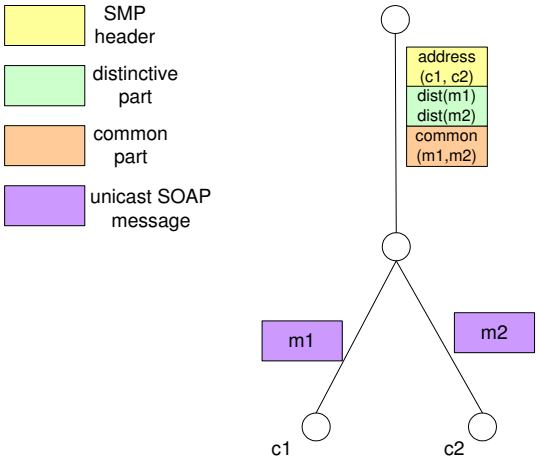

Figure 2: SMP protocol behavior

a result, the queue is empty for new requests and the above aggregation steps can be repeatedly carried out again. Messages in the queue can also be dispatched automatically after the defined time period. It is important to note that to deploy SMP in a real network, all routers in the network need to be SMP-compatible. This can be done by installing an SMP software, which is an implementation of the proposed SMP, on each router to enable it to interpret SMP messages. The SOAP header in an SMP message specifies the next hop router as the message's destination. Therefore, when an intermediary router receives an SMP message, it processes the message as if it is the final destination of the message. Since an SMP-compatible router operates on the application layer, it has full access to the message's envelope and parses the SOAP body to get the list of clients encoded in the SMP header and the actual payload in the SMP body.

4. OUTLINE OF THE APPROACH

Our proposed approach is based on the SOAP-based Multicast Protocol (SMP) proposed in [10] and described in the previous section. The idea is to use XML similarity to recognize SOAP messages that contain common parts and therefore can be grouped by the service provider together before checking them against the security policy rules. In this paper, for the sake of simplicity, we only deal with a single sender and a single receiver, focusing on the problem of aggregating similar messages; however, the approach can be straightforwardly extended to the case of multiple receivers. Figure 3 shows the general architecture of a SOAP message exchange. On the sender side or Web Service Requester in Figure 3, the Requester will first acquire the required security token from the Security Token Service and then the protocol stack generates SOAP envelopes that satisfy its policy. It adds integrity and confidentiality credentials under the <`Security`> header that is defined according to the WS-Security standard.

The basic idea of the approach consists in aggregating the common message structure and data values of multiple messages and their distinctive parts in a single message in order to apply once the rules of the security policy instead of repeating the same procedure on duplicated parts in many messages. In other words, the signed message at step 5 in Figure 3 is not a single message but an aggregate of similar SOAP messages. To do this the idea is to measure the similarity level between the messages to determine if they have sustainable parts in common that can justify the extra processing time at the web service provider. The following sub-sections will describe the idea for similarity measurements and message generation.

4.1 Using XML similarity to aggregate SOAP messages

This section presents the data model for the formulation of the similarity to be used for message aggregation. Since a SOAP message has an XML format, it is a hierarchical document with nested tags. As such, a SOAP message can be modeled as a rooted ordered tree, as shown in the definition below.

Definition 1. A SOAP message tree T is a rooted ordered tree, where each node n is a pair $n = (tag, content)$; tag is the node name and $content$ is the node value. [1]

Most SOAP messages that are generated by the same service provider will have some common message structure (or

[1]For the sake of simplicity in this paper we suppose that content can be of primitive type only. An effective similarity measure involving complex types is target of the next step of our research.

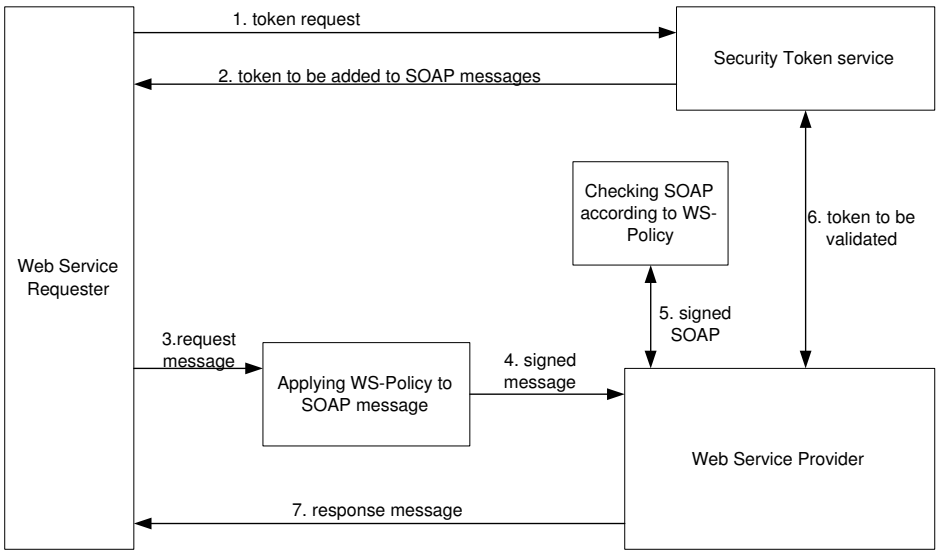

Figure 3: General architecture of a secure SOAP transmission

template). Specifically, they will have similar information provided in the header and the same structure in the body if they are generated to query the same operation. Based on this observation, it is possible to define the similarity between two SOAP messages $Soap_1$ and $Soap_2$ labeled as $sim(Soap_1, Soap_2)$ as a function of the similarity between the two message templates and the similarity between the data values of the messages, as described below. The function adopted to aggregate similarities is a T-norm (*Triangular norms*) function. Triangular norms are operations which generalize the logical conjunction in the semantics of mathematical fuzzy logics and they are commonly used to combine criteria in multi-criteria decision making. Examples of T-norms are the *minimum* or *Gödel* t-norm, the *product* t-norm, and the *Lukasiewicz* t-norm. The choice of the most suitable t-norm can be done on the basis of application dependant considerations. As instance, the *min* function has the property to guarantee that aggregate messages have both kind of similarity higher that a certain threshold.

Next, it is further explained how each type of similarity is computed. SOAP messages are based on XML templates which are formed by a number of node tags definitions, the assumption is that both sender and receiver knows this template; therefore, node tags can be used as a basic metric for measuring the similarity between SOAP message templates which is defined as follows.

Definition 2. $sim_{temp}(Soap1, Soap2) = |\frac{N}{max(N1,N2)}|$
where: N is the number of common nodes tags (excluding closing tags) between Soap1 and Soap2; N_1 is the number of node tags that Soap1 has; N_2 is the number of node tags that Soap2 has.

A major limitations of existing similarity measurement methods used in XML database querying [6] is that they usually compare the values of two XML documents that have the same template (the same path from a node to the root). On the contrary, the technique proposed here allows any XML documents be compared by counting the number of common node tags shared in the documents. This simplification holds because we assume that both sender and receiver knows the message templates, and hence can recognize different messages sharing the same template. Instead, the similarity between the values of two nodes is measured based on the Levenshtein distance (or edit distance) method. The edit distance is a well established method for weighting the difference between two strings. It measures the minimum number of token insertions, deletions, and substitutions required to transform one string into another using a dynamic programming algorithm. For example, the edit distance between the two strings `cat` and `bat` equals 1 because one substitution is required to change the string `cat` into `bat`. Hence we can compute the similarity between the two strings as $sim_{val_1,val_2} = 1 - \frac{c}{max(c_1,c_2)} = 1 - 1/3 = 0,67$ where c is the number of required substituted characters, and c_1 and c_2 are the number of characters composing val_1 and val_2 respectively.

Two SOAP messages are considered syntactically similar if their similarity degree $sim(Soap1, Soap2)$ is equal or greater than a parameter ρ, called the *similarity threshold*, which is dependent on the application domain.

For example if we consider the two messages in 4 that refer to two different purchases of the same person using once a credit card (a string) and the other time using cash (an integer), we will see that the messages share 7 node tags on 8. Hence we can say that $sim_{temp}(Soap1, Soap2) = 7/8 = 0.875$. From the content point of view, we see that only two leaf nodes are different. Hence we compute the string similarity only on these nodes. The first node presenting differences in the content value of messages (a) and (b) is <`OrderDate`>. The edit distance between the two content values is 1, hence $sim_{val_1,val_2} = 1 - 1/10 = 0.9$. The second difference between messages (a) and (b) is given by the nodes <`CreditCard`> and <`cash`> that appear only in one message and not in the other. In this case the two nodes are ignored in the similarity value measure because it is unlikely that two nodes of different data types have similar values.

At the end of the computation $sim(Soap1, Soap2) = min(sim_{temp}(Soap1, Soap2), sim_{val_1,val_2}(Soap1, Soap2)) = 0.875$ that is likely above a common predefined *similarity threshold* (e.g., 0,75).

5. SOAP AGGREGATE MESSAGE STRUCTURE

This section explains the structure of a *SOAP aggregate message* supporting optimization of security policy rules checking.

Figure 5 shows the structure of the *aggregate message*. The original messages ids are stored in the aggregate header which is encapsulated inside the SOAP message body as shown in figure. There are two sections in the *aggregate message* body:

- the *common* section containing common values and structures of all messages to be aggregated;
- the *distinctive* section containing individual different parts for each original message.

The aggregate message generation mechanism is explained through an example. Let assume we have two messages containing the credit cards details of Alice coming from two different orders. Figure 4 illustrates a SOAP message containing Alice's details. The security policy asks that credit cards numbers should be signed and encrypted before transmission.

Upon receiving similar messages to be secured and determining that the similarity degree of the messages is equal or greater than the *similarity threshold* ρ (see section 4.1), the server will generate an aggregate message to capture all information to be secured. In our example, the aggregate message should include the <OrderDetails> structure (as presented in Figure 4) and the data values of the <CustomerInfo> and <CreditCard> nodes in the <Common> tag of the aggregate message since the two orders belong to the same customer. And in the <Distinctive> tag, there should be two <part> elements, each <part> element contains the values of each order, their order dates and the products bought. In this way the signature could be applied directly in the common part once.

5.1 Discussion on times and costs

In the traditional multicast scenario, when a client sends a request to the server, the server will not respond to it promptly but rather it will wait for a number of other requests from other clients who are requesting the same service operation. This is because the server needs to get a reasonable number of the same requests from the clients before sending a multicast message out. This waiting period, denoted by $t_{procServer}(multicast)$, is significant and does not exist in the unicast scheme. Our approach works in a similar way: when the service provider receives a message it does not send it immediately to the policy evaluator, but waits for a number of other messages that can be aggregated in a single message before checking it against the ws-policy. Hence also in our context we must take into account a waiting time $t_{procServer}(message)$, and it is necessary to evaluate under which conditions our approach pays well w.r.t the traditional unicast approach (where each message is received, policy checked, and processed, one by one). Generally our multicast approach shows its benefits when:

```
<soap:Envelope>
<soap:Header>
Address of the receiver </soap:Header>
<soap:Body>
<aggregate:Header>
ids of messages that compose this
aggregate
</aggregate:Header>
<aggregate:Body>
<Distinctive>
<part>
<messageIDs>
messages that share this distinctive part
</messagesIDs>
<data>
Data values of the distinctive part
</data>
</part>
<part>
<messageIDs>
messages that share this distinctive part
</messageIDs>
<data>
Data values of the distinctive part
</data>
</part>
</Distinctive>
<Common>
Common structure and data values
</Common>
</aggregate:Body>
</soap:Body>
</soap:Envelope>
```

Figure 5: The aggregate message structure

Definition 3. $t_{procServer}(message)+t_{procServer}(aggregate)+t_{evaluator}(aggregate) < \sum_i^N (t_{procServer}(unicast) + t_{evaluator}(unicast))$ where

- $t_{procServer}(message)$ is the time the server wait to collect the N messages to be aggregated;
- $t_{procServer}(aggregate)$ is the time required to create the aggregate message;
- $t_{evaluator}(aggregate)$ is the time required by the policy evaluator to check the aggregate message
- $t_{procServer}(unicast)$ is the time required to receive a message and $t_{evaluator}(unicast)$ is the time required to evaluate it in the unicast approach.

Hence if we suppose that $t_{evaluator}(unicast) \approx t_{evaluator}(aggregate)$, it is clear that the approach bottleneck lays in the algorithm used to compute the messages similarity and to create the aggregate message. The algorithm proposed in [10], which combines the complexities of the SMP aggregation and splitting processes at the client side, has a complexity of $O(d^2 log(d) + N)$, where d is the length of a data value in a SOAP message and N is the number of service requestors.

```
<soap:Envelope>
<soap:Body>
<OrderDetails>
<OrderDate>2006-06-12</OrderDate>
<CustomerInfo>
<CustomerName>Alice</CustomerName>
<CustomerCity>New York</CustomerCity>
</CustomerInfo>
<Good>
<Product>Blue polo</Product>
<Price>49$</Price>
</Good>
<Good>
<Product>Women shoes</Product>
<Price>69$</Price>
</Good>
<CreditCard>VISA 1324 6487 2265</CreditCard>
</OrderDetails>
</soap:Body>
</soap:Envelope>
```

(a)

```
<soap:Envelope>
<soap:Body>
<OrderDetails>
<OrderDate>2006-06-13</OrderDate>
<CustomerInfo>
<CustomerName>Alice</CustomerName>
<CustomerCity>New York</CustomerCity>
</CustomerInfo>
<Good>
<Product>shirt</Product>
<Price>110$</Price>
</Good>
<Good>
<Product>Prada handbag</Product>
<Price>842$</Price>
</Good>
<cash>952</cash>
</OrderDetails>
</soap:Body>
</soap:Envelope>
```

(b)

Figure 4: Sample SOAP messages

6. CONCLUSIONS AND FUTURE WORK

This paper proposes an XML similarity approach to aggregate SOAP messages in order to improve the performance of a policy rules evaluator. The idea develops on SMP, a similarity based protocol studied to improve transmission performances. Future work of this idea include extensive experimentation of the performance gain of the policy evaluator, and a comparative analysis of XML similarity for SOAP messages templates.

Acknowledgements

This work was supported in part by the European Commission within the SecureSCM project in the FP7-ICT Programme under contract n.AOR 213531, and by contract/grant sponsor FIRB research fund of MIUR, research project TEKNE (contract/grant n.RBNE05FKZ2).

7. REFERENCES

[1] Corba official web site, http://www.corba.org/.

[2] Dce portal, http://opengroup.org/dce/.

[3] Eai and web services overview, http://www.bitpipe.com/data/web/bpmd/eai/eai_overview.jsp.

[4] A. Guitton A. Boudani and B. Cousin. Gxcast: Generalized explicit multicast routing protocol. In *In Proceedings of the ninth international Symposium on Computers and Communications, pages 1012Ű1017,Piscataway, USA, 2004. IEEE Press.*

[5] C. Buschmann C. Werner and S. Fischer. Compressing soapmessages by using differential encoding. In *In Proceedings of the IEEE International Conference on Web Services, page 540,Washington, DC, USA, 2004. IEEE Computer Society.*

[6] Ernesto Damiani and Letizia Tanca. Blind queries to XML data. In *Database and Expert Systems Applications*, pages 345–356, 2000.

[7] Stefano Paraboschi Ernesto Damiani, Sabrina De Capitani di Vimercati and Pierangela Samarati. Securing soap e-services. *International Journal of Information Security*, 1(2):100–115, 2002.

[8] S. Abiteboul G. Gobena and A. Marian. Detecting changes in XML documents. In *In International Conference on Data Engineering, pages 41Ű52, 2002.*

[9] Paolo Ciaccia Ilaria Bartolini and Marco Patella. A framework for the comparison for complex patterns. In *In Proc. 1st InternationalWorkshop on Pattern Representation and Management (PaRMaŠ04), Crete, Greece, March 2004.*

[10] Zahir Tari Khoi Anh Phan and Peter Bertok. Optimizing web services performance by using similarity-based multicast protocol. *Web Services, 2006. ECOWS '06. 4th European Conference on*, pages 119–128, Dec. 2006.

[11] X. Liu and R. Deters. An efficient dual caching strategy for web service-enabled pdas. In *In Proceedings of the 22nd Annual ACM symposium on Applied Computing, pages 788-794, Seoul, Korea, 2007. ACM Press.*

[12] J. Marsh M. Nagy M. Fernandez, A. Malhotra and N. Walsh. Xquery 1.0 and xpath 2.0 data model (XDM). In *World Wide Web Consortium (W3C) January 2007.*

[13] T. Naumowicz H. Ritter M. Tian, T. Voigt and J. Schiller. Performance considerations for mobile web services. In *Computer Communication, 27(11):1097-1105, July 2004.*

[14] A. Schaad M.A. Ramahan and M. Rits. Towards secure SOAP message exchange in a SOA. In *In Secure Web Services 2006, Alexandria, Virginia, USA.*

[15] OASIS. Oasis security services (SAML) TC http://www.oasis-open.org/committees /tc_home.php/wg_abbrev=security.

[16] OASIS. Oasis standards and other applications, http://www.oasis-open.org/specs/index.php#wssv1.0.

[17] OASIS. Oasis UDDI specification tc http://www.oasis-open.org/committees /tc_home.php?wg_abbrev=uddi-spec.

[18] S. Castano S. Bergamaschi and M. Vincini. Semantic integration of semistructured and structured data sources. In *In SIGMOD Record, vol. 28, 1999.*

[19] Klaas-Jan Winkel Theodore Dalamagas, Tao Cheng and Timos Sellis. Clustering XML documents using structural summaries. In *In Proceeding of ClustWeb - International Workshop on Clustering Information over the Web in conjunction with IntŠl Conference on Extending Database Technology (EDBT 04), Heraklion - Crete, Greece, 14 Mar. 2004.*

[20] K. Chiu W. Lu and D. Gannon. Building a generic soap framework over binary XML. In *In Proceedings of the 5th International Conference on Computer and Information Technology, pages 136-140, Shangai, China, September 2005. IEEE Computer Society.*

[21] W3C. Soap version 1.2 part 2: Adjuncts (second edition). World Wide Web Consortium (W3C) January 2007.

[22] W3C. Web services description language (WSDL) 1.1. World Wide Web Consortium (W3C) March 2001.

[23] D. Dewitt Y. Wang and J. Y. Cai. An effective change detection algorithm for XML documents. In *2003.*