Differential SOAP Multicasting*

Joe Tekli#, Ernesto Damiani
Department of Information Technology,
Università Degli Studi Di Milano, Crema, 65 – 26013, Italy
{joe.tekli, ernesto.damiani}@unimi.it

Richard Chbeir
LE2I Laboratory UMR-CNRS
University of Bourgogne, 21011 Dijon, France
richard.chbeir@u-bourgogne.fr

***Abstract*— SOAP has been widely adopted as a simple, robust and extensible XML-based protocol for the exchange of messages among web services. Unfortunately, SOAP communications have two major performance-related drawbacks: i) *verbosity*, related to XML, that leads to increased network traffic, and ii) *high computational burden* of XML parsing and processing, that leads to high latency. In this paper, we address these two issues and introduce a novel framework for Differential SOAP Multicasting (DSM). The main idea consists in identifying the common pattern and differences between SOAP messages, modeled as trees, so as to multicast similar messages together. Our method is based on the well known concept of Tree Edit Distance, built upon a novel *filter-differencing* architecture to reduce message aggregation time, identifying only those messages which are relevant (i.e., similar enough) for similarity evaluation. In addition, our technique exploits a dedicated differencing output format specifically designed to carry the minimum amount of *diff* information, in the multicast message, so as to minimize the multicast message size, and therefore reducing the network traffic. The battery of simulation experiments conducted to evaluate our approach shows the relevance of our method in comparison with traditional and dedicated multicasting techniques.**

Keywords: *SOAP, XML, Message Multicasting, Differential Processing, SOAP Performance, Web Service Communications.*

I. INTRODUCTION

Web Services (WS) have emerged as a technology that enables machine-to-machine interaction within distributed, heterogeneous computing environments. WS differ from traditional software integration frameworks, such as CORBA [35], DCOM [18] and Java RMI [41], in that they utilize well-established and open Web protocols, chiefly XML [5].

WS rely on two standard XML schemata: WSDL (Web Service Description Language) [10] supporting the machine-readable description of a service's interface, and SOAP (Simple Object Access Protocol) [50] dictating the messages' format. Bindings to existing protocols (e.g., HTTP, FTP, SMTP…) have been provided for SOAP messages' negotiation and transmission.

While carrying most XML's advantages, WS technology has inherited a major XML drawback, *verbosity*, which strongly affects its performance. Indeed, SOAP message exchanges are quite elaborate; the client program has to build the skeleton of the XML message, put the right values in it (serialization), and then send it to the remote service. In turn, the service parses the message, digging out the data it needs (de-serialization), and then goes through the same procedure to generate an XML reply. No wonder, then, that SOAP message processing produces considerable network traffic and causes higher latency than competing technologies [21, 42]. High latency becomes even more critical when handling large volumes of SOAP-based communications such as with emerging e-science [16] and e-business [39] applications.

In this context, similarity and differential encoding have been often proposed to enhance SOAP performance, aiming to i) reduce processing time (in parsing [29, 43, 47], serialization [2, 15], and de-serialization [1, 42]), and to ii) reduce network traffic via compression [49] and multicasting [3, 36, 37]. Similarity-based performance enhancement is based on the observation that SOAP exchanges often involve highly similar messages since those created by the same implementation usually have the same structure, and those sent from a server to multiple clients tend to show similarities in structure and content (e.g., stock quote services [37], online booking and meteorological broadcast services [3], etc.). In this paper, we focus on *SOAP multicasting*, as a technique to save network bandwidth by delivering SOAP messages to a group of destinations simultaneously [52].

To our knowledge, the only approach to SOAP multicasting was described in [37], where the authors introduce SMP (Similarity-based Multicasting Protocol), identifying, indexing and routing similar SOAP messages together (cf. Section II). SMP's main contribution consists in grouping and transmitting together similar SOAP messages, in comparison with identical-only message aggregation of traditional network-layer (e.g., IP) multicasting [52]. SMP's SOAP message aggregation process consists of two steps: i) quantifying the resemblance between SOAP messages using a heuristic XML-based similarity measure [28], and ii) identifying the common part (intersection) and distinctive parts between the most similar messages, to be grouped together in one aggregate multicast message. Nonetheless, careful analysis of [37] led us to pinpoint certain aspects which limit both the effectiveness and efficiency of SMP multicasting. On one hand, while SMP considers the common and distinctive parts of SOAP messages in multicast message encoding, it does not always generate minimum sized aggregate messages (and thus does not guaranty optimal multicast network traffic) since SMP disregards similarities between the SOAP messages' *distinctive* parts (which are repeated multiple times in the aggregate message regardless of their resemblances), as we will see in the motivating examples (Section III.A). On the other hand, the two phase process of i) computing SOAP similarity and ii) identifying message common/distinct parts, induces additional processing overhead, i.e., higher response time, which could be alleviated if both tasks could be integrated together.

In this paper, we propose an improved SOAP multicasting method to address the limitations of SMP [37]. In summary,

* Work supported in part by a *Fondazione Cariplo*.
# The author is currently a visiting researcher with the Department of Computer Science, University of Sao Paulo, Brazil, supported by the *Research Support Foundation of the State of Sao Paulo* (*FAPESP*).

978-0-7695-4463-2/11 $26.00 © 2011 IEEE
DOI 10.1109/ICWS.2011.51

we introduce a framework for Differential SOAP Multicasting (DSM), improving multicasting effectiveness (minimizing network traffic) and efficiency (minimizing processing overhead). Our framework is founded on the well known concept of Tree Edit Distance [4, 6] for comparing and differencing structured XML-based data (which is the case of SOAP messages). DSM is built upon a filter-differencing similarity evaluation architecture, inspired by filter-refinement approaches used in query processing [19, 23]. This allows to identify only those SOAP messages which are relevant (i.e., similar enough) for exact tree edit computations, avoiding computing similarity when it is not necessary[1]. In short, our method allows:

- Encoding the differences between SOAP messages to be multicast, including only their *distinctive* parts, so as to minimize aggregate message size, and thus network traffic,
- Integrating both SOAP similarity computation and message aggregation in one single tree edit distance measure, enhanced via a dedicated filter-differencing technique, so as to reduce multicast processing overhead.

The remainder of this paper is organized as follows. Section II reviews the background in SOAP processing. Section III presents the overview of our approach. Our Differential SOAP Multicasting method (DSM) is developed in Section IV. Simulation experiments are described in Section V. Section VI concludes the paper.

II. BACKGROUND

Several studies have been proposed in the context of SOAP performance enhancement [2, 37, 42, 47, 49], and can be grouped following the kind of SOAP processing they perform.

The authors in [2] address SOAP message serialization, i.e., converting in-memory data types into XML. They identify the main performance bottleneck as that of transforming in-memory data of numeric types into the corresponding ASCII-based XML representation. The authors introduce dedicated indexing tables to track changes between in-memory data and their serialized representations, so as to only serialize the changes to the previously sent message. A comparable approach is introduced in [15], where the authors address client-side SOAP message caching. In [1, 42], the authors target SOAP de-serialization which can be viewed as the inverse function of serialization, i.e., converting XML messages to in-memory application objects. The authors in [42] propose an automaton-based solution creating a link between the defined automaton and the application object. The automaton processes incoming messages, and if matched, returns the linked application object to the SOAP engine after partially de-serializing only the regions that differ from the past messages. In [1], the authors propose to periodically checkpoint the state of the de-serializer, and compute checksums for portions of incoming SOAP messages, in order to de-serialize only those portions which are different. A few studies have proposed dedicated SOAP parsers, in comparison with generic DOM [51] and SAX [32] XML parsers, taking into account the particularities of SOAP messages in order to improve performance. Early approaches, e.g., XSOAP [40], limit the validation scope to those elements specific to SOAP so as to gain in validation time. Recent methods [29, 43, 47] focus on differential parsing, exploiting the similarities between SOAP messages. They make use of predefined templates modeled via dedicated automatons, memorizing the basic structures of the SOAP messages (based on the corresponding WSDL [43], or the messages themselves [29, 47]) and only process those parts of the messages which correspond to variable parts in the templates.

In addition to processing efficiency, a major drawback of using SOAP resides in its demand for bandwidth, critical in various domains such as mobile computing [37] and sensor networks [49]. This problem has been investigated on two levels: i) SOAP compression [49], to reduce message size prior to transmission, and ii) SOAP multicasting [36, 37], to optimize SOAP network traffic. Existing XML compression methods, e.g., [9, 26], could be utilized with SOAP, yet might not always be appropriate since SOAP messages are of relatively smaller sizes, and might yield compression coding tables which require more space than the original SOAP messages themselves [49]. Following this observation, the authors in [49] propose a differential SOAP compression approach, exploiting the WSDL schema definition to generate a SOAP message skeleton describing the structures of corresponding SOAP messages. Consequently, only the differences between the SOAP message and the predefined skeleton are transmitted. Another way to reduce SOAP network bandwidth is to perform multicasting, transmitting the same information destined to multiple clients once, instead of sending multiple replicas [52]. As outlined above, the Similarity-based SOAP Multicasting Protocol (SMP) proposed in [37] groups and transmits together similar SOAP messages, in comparison with identical-only message aggregation with traditional (IP) multicasting [52]. An aggregate SMP message consists of two parts: the *common* part section containing common values of the messages, and the *distinctive* part section containing the different parts of each message. The SMP message is then encapsulated within the body of a classic SOAP message, which header encompasses the address of the next router along the path to all intended recipients. The authors exploit a heuristic similarity measure [28] to quantify the resemblance between SOAP messages, in order to identify the most similar candidates for aggregation and multicasting. Message aggregation (identifying common/distinctive parts) is undertaken in a subsequent dedicated process. In [36], the authors propose an enhanced similarity-based routing protocol, transmitting messages following paths such as there are more shared links between similar messages, to further reduce network traffic. SOAP multicasting has also been recently investigated in the context of SOAP security policy evaluation [3, 13], applying security rules only on distinct parts of the multicast message to improve policy evaluation performance.

To sum up, automaton-based techniques to SOAP message comparison (mainly used with parsing and de-

[1] In addition, we define an XML-based differential output format, SDL (Simple Diff Language), designed to carry the minimum information (in the aggregate multicast message) necessary to regenerate original SOAP messages at multicast end-point. However, due to space limitations and for clarity of presentation, we omit the presentation of SDL here and refer the reader to a dedicated paper (details are provided in [45]).

serialization) [29, 43, 47] focus on messages which strictly correspond to predefined templates. They do not produce a similarity value to quantify the resemblance between SOAP messages, but rather a Boolean result identifying whether the message is valid or not w.r.t. (with respect to) the predefined template. Other approaches [1, 37] usually sacrifice some quality (i.e., comparison accuracy) to gain in performance, such as the error-prone checksum-based measure [1] (exploited for SOAP de-serialization), and the heuristic SMP similarity measure [37] (used for SOAP multicasting). Moreover, neither method allows seamless SOAP message aggregation.

For further details, a comprehensive survey on SOAP performance enhancement techniques is provided in [46].

III. OVERVIEW OF THE APPROACH

In this paper, we address the tasks of similarity evaluation and differential encoding of SOAP messages, to perform SOAP multicasting. As stated previously, SMP [37] aggregates SOAP messages by identifying their *common* and *distinctive* parts. Yet, it disregards certain similarities, mainly between the messages' *distinctive* parts, repeated multiple times in the aggregate message regardless of their resemblances.

A. Motivating Example

To motivate the need for a new approach, let us consider the dummy SOAP messages M_i, $_{i=1...6}$ in Figure 1. In this example, we abstract messages to basic character strings for the sake of simplicity. Figure 1.a shows the expected aggregation result, using SMP. One can see that element *'e'*, which is contained in messages M_3, M_4, M_5 and M_6, is repeated four times in the SMP message *distinctive* section, so as to regenerate the original SOAP messages, such as: $M_i = Common + D_i$.

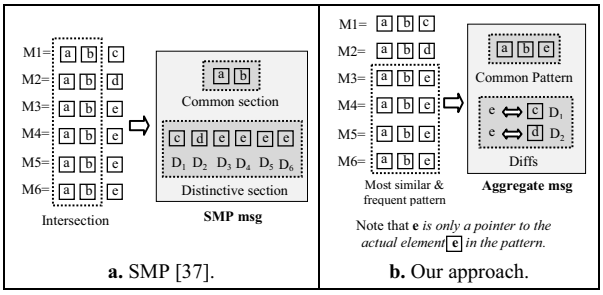


Figure 1. Motivating example to SOAP message aggregation.

However, we argue that such repetitions of identical or similar elements can be eliminated so as to minimize the aggregate message size. To do so, we need to identify the most similar and frequent pattern among SOAP messages (instead of identifying the intersection such as with SMP), and consequently only encode the differences (*diffs*) between each message and the pattern. Hence, only the minimum amount of information necessary to regenerate the original SOAP messages is encapsulated in the aggregate message, eliminating redundancies as shown in Figure 1.b.

B. Underlying Technique

In order to attain our effectiveness (minimizing aggregate message size, and thus network traffic) and efficiency (reducing processing overhead) goals, we exploit the well known concept of tree edit distance (TED) [4, 53] (also known as *tree differencing*), SOAP messages being modeled as Ordered Labeled Trees [51]. A great advantage of using tree edit distance is that along the similarity value, a *diff* is generated (i.e., *edit script*, or *delta*) providing a record of the exact differences, in terms of transformation operations, between the compared trees. This is central to achieve full integration of SOAP similarity evaluation and message aggregation (as opposed to the complex two-step similarity/aggregation process of SMP [37]). In addition, TED methods have been widely used to compare XML-based data [7, 12, 34], and have been proven optimal w.r.t. less accurate (error-prone or heuristic) methods [6]. This is of paramount importance to accurately identify the most common *pattern* minimizing the *diffs* among the SOAP messages being aggregated, and thus minimize overall aggregate message size.

C. Outline of our Proposal

In short, we introduce a framework for Differential SOAP Multicasting (DSM), consisting of two main modules (Figure 2): *Message Multicasting* (MM_{DSM}), and *Message Reconstruction* (MR_{DSM}). Briefly, our multicasting module starts by transforming SOAP messages into their DOM [51] tree representations. SOAP trees are processed for similarity evaluation and aggregation at once, via an integrated tree edit distance measure, to produce multicast DSM messages. Then, our message reconstruction module rebuilds the original SOAP messages. Note that each DSM multicast message consists of a message *pattern* and various *diffs*, describing the differences between the unicast SOAP messages and the multicast message *pattern*. The *pattern* comes down to the SOAP message sharing the maximum similarities to all others being processed in the same multicast, i.e., the message inducing the smallest *diffs*. Thus, message reconstruction consists in patching the *pattern* of the multicast message, with the *diff* corresponding to the SOAP message to be regenerated.

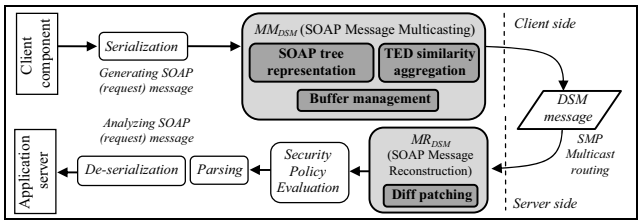


Figure 2. Outline of our approach[2].

DSM builds on the groundwork of the SMP protocol [37], in exploiting the message formatting, indexing and routing facilities provided by SMP.

IV. SOAP MESSAGE MULTICASTING

Our main idea consists in comparing SOAP messages in a pair-wise manner, generating and composing *diffs* accordingly. A single DSM multicast message is generated for each group of SOAP messages such that their similarities are above a given threshold. Here, a user-defined similarity threshold $Thresh_{Sim}$ and time frame T_{Pool} are exploited. When the new outgoing

[2] SOAP *response* message processing is similar to *request* processing, yet the *response* is generated at the server side, and transmitted toward the client.

SOAP message does not satisfy the predefined threshold $Thresh_{Sim}$ w.r.t. all messages in the buffer, it is allocated a new buffer pool, for a period of T_{Pool} time, and constitutes the seed of a new DSM multicast message. When the outgoing message satisfies the similarity threshold, it is appended to the pool corresponding to the in-buffer message with which it shares maximum similarity. Thus, when the T_{Pool} expires for each buffer pool, the latter's buffer space is released and the corresponding multicast DSM message is sent over the wire. The activity diagram of our SOAP multicasting module is depicted in Figure 3. It consists of three components: i) *SOAP Tree Representation*, ii*) SOAP Tree Similarity Evaluation and Differencing*, and iii) *SOAP Buffer Management*.

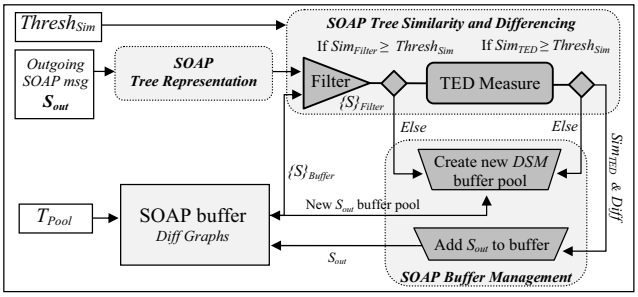


Figure 3. Simplified activity diagram describing our SOAP message multicasting module, MM_{DSM}.

A. SOAP Tree Representation

Definition 1 – SOAP Message Tree: It is a rooted tree S which nodes $n_i \in S$ represent SOAP message elements, ordered and labeled following the corresponding message. Element values mark the nodes of their containing elements ●

Consider an air travel booking service, and the SOAP response message in Figure 4 as an answer to a booking confirmation request. Here, we show the contents enclosed in the SOAP message body, and disregard meta-data in the header. Corresponding tree representation is depicted in Figure 5.

```
<soap:Envelope xmlns:xsd= "…">
 <soap:Header> … </soap:Header>
 <soap:body>
  <BookingConfirmationResp>
   <FlightBooking>
    <FlightInfo>
      <FlightNum>AZ211</FlightNum>
      <SourceHub>Milano</SourceHub>
      <DestHub>Paris</DestHub>
    </FlightInfo>
    <ClientInfo>
      <Name>Paula Olivetti</Name>
      <PhoneNum>+39 3206813826</PhoneNum>
      <CCNum>4511 2326 1121 3432</CCNum>
    </ClientInfo>
   </FlightBooking>
  </BookingConfirmationResp>
 </soap:body>
</soap:Envelope>
```

Figure 4. Sample SOAP message.

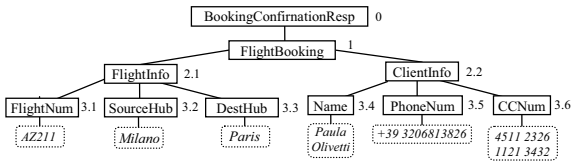


Figure 5. Sample SOAP message tree representation.

Note that for tree node identifiers in the SOAP message tree representation, we follow [37] in using a depth/order Dewey numbering system for trees, which allows to effectively pinpoint the exact location of each node in the SOAP tree (central in consequently encoding the *diffs*).

B. SOAP Tree Similarity and Differencing Evaluation

In short, we propose a two step *filter-differencing* similarity evaluation approach (cf. Figure 3), inspired by *filter-refinement* architectures in query processing [19, 23, 25]. The main idea is to first run a *filter* step, exploiting a fast approximation (Sim_{Filter}) of our main edit distance measure (Sim_{TED}) to compare the outgoing SOAP tree (S_{out}) to all those kept in the SOAP buffer. The filtering step identifies the set of SOAP trees in the buffer which are most similar (following Sim_{Filter}) to the outgoing tree S_{out}. Formally:

$$Filter = \{ S \in Buffer \mid Sim_{Filter}(S_{out}, S) \geq Thresh_{Sim} \wedge \forall S' \in Buffer, Sim_{Filter}(S_{out}, S) \geq Sim_{Filter}(S_{out}, S') \} \quad (1)$$

Consequently, the *differencing* phase consists in conducting similarity evaluation (Sim_{TED}) and *diff* generation to compare S_{out} with its most similar counterparts $S \in Filter$, identified in the filtering step.

1) Filter Similarity Measure

Three main conditions have to be satisfied for the filter step to be efficient [19, 25]: i) the filter measure has to be considerably easier to compute than the main similarity measure, ii) a substantial part of the SOAP buffer messages has to be filtered out, and iii) the completeness of the filter phase, w.r.t. the main similarity evaluation phase, has to be verified. While the first two criteria are intuitive, completeness in this context is less straightforward. It underlines that the filter step must not allow any false drop outs. In other words, all SOAP trees in the buffer ($S \in Buffer$), which are deemed similar to S_{out} w.r.t. the main similarity measure Sim_{TED}, should be included in the filter candidate set ($S \in Filter$).

Definition 2 – Upper Bound Function: Let Ω be a set of objects, a similarity function *Sim'* is an upper bound of function *Sim*, if for any two objects $p, q \in \Omega$, $Sim'(p, q) \geq Sim(p, q)$ [14] ●

Definition 3 – Filter Completeness: Given a similarity measure Sim_{TED}, and a filter characterized by similarity measure Sim_{Filter}, the filter is said to be complete w.r.t. Sim_{TED} if Sim_{Filter} is an upper bound of Sim_{TED} [19] ●

With an upper bound similarity measure, it is possible to safely filter out all buffer SOAP trees which have a filter similarity Sim_{Filter} less than the minimum acceptable similarity degree, i.e., $Thresh_{Sim}$ (cf. Formula (1)). In other words, our filter eliminates all candidate SOAP trees which are outside the maximum relevant similarity range, for the message aggregation and multicasting operation at hand.

Various TED-related filter similarity functions have been proposed in the context of structure query processing [19, 25]. These range over very coarse functions comparing the number of edges in both structures being compared [25], to more complex measures exploiting special histograms to describe the structural features of the data (distribution of the number of leaf nodes, distinct node labels…) [19]. Since existing filter methods seem either too coarse [25] or somewhat complex [19], we propose three simple filter functions to specifically

capture the main characteristics of SOAP message trees: *node edges* (*parent-child relations*) and *node order* to describe SOAP structure, and *node values* to describe SOAP message contents. Our filters are based on the vector space model widely used in information retrieval [31], which performance has been accredited in a variety of applications [38].

Definition 4 – Node Edge Vector Space: Given two SOAP trees S_i and S_j, we define corresponding parent-child vectors $\overrightarrow{V_i}$ and $\overrightarrow{V_j}$ in a space which dimensions represent, each, a single edge $e_r \in (S_i \times S_i) \cup (S_j \times S_j)$, such as $1 < r < E$ where E is the number of distinct parent-child relations in S_i and S_j. The value of a coordinate $w_{\overrightarrow{V_i}}(e_r)$ in $\overrightarrow{V_i}$ stands for the number of occurrences of edge e_r in tree S_i ●

Consequently, we exploit the Manhattan distance [24] to compute the *node edge* filter function $Sim_{n\text{-}edge}$, since it is consistent with Definitions 1 and 2, in providing a lower bound for our main TED similarity measure (the detailed mathematical proof is provided in [45]).

$$\mathrm{Sim}_{\text{n-edge}}(S_1, S_2) = 1 - \frac{\frac{1}{2}\sum_{r=1}^{E} |\, w_{\overrightarrow{V_1}}(e_r) - w_{\overrightarrow{V_2}}(e_r)|}{|S_1| + |S_2|} \quad \in [0,1] \qquad \textbf{(2)}$$

Likewise, we exploit formulas, based on the Manhattan distance, to compute both the *node order* and *node value* filter functions: $Sim_{n\text{-}order}$ and $Sim_{n\text{-}value}$, each w.r.t. its corresponding vector space defined hereunder.

Definition 5 – Node Order Vector Space: Given two SOAP trees S_i and S_j, we define the node order vectors $\overrightarrow{V_i}$ and $\overrightarrow{V_j}$ in a space whose dimensions represent, each, the Dewey index [37] (cf. Section IV.A) associated to a single node $n_r \in S_i \cup S_j$, such as $1 < r < I$ where I is the number of distinct node index values in S_i and S_j. Vector coordinates are binary, indicating whether a node of the designated Dewey index exists or not for a given dimension n_r ●

Definition 6 – Node Value Vector Space: Given two SOAP trees S_i and S_j, we define node value vectors $\overrightarrow{V_i}$ and $\overrightarrow{V_j}$ in a space whose dimensions represent, each, a distinct node value associated to a node $n_r \in S_i \cup S_j$, such as $1 < r < Vl$ where Vl is the number of distinct node values in S_i and S_j. Vector coordinates designate the occurrences of each node value ●

Consider the SOAP trees in Figure 6. The corresponding filter vector representations are depicted in Figure 7.

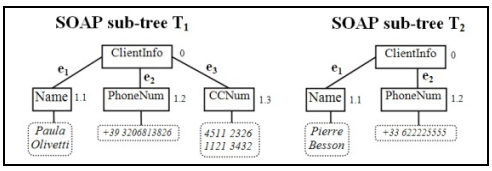


Figure 6. Sample SOAP sub-trees.

	e_1	e_2	e_3
V_1	1	1	1
V_2	1	1	0

a. Node edge vectors

	0	1.1	1.2	1.3
V_1	1	1	1	1
V_2	1	1	1	0

b. Node order vectors

	P. Olivetti	+39 320…	…
V_1	1	1	…
V_2	0	0	…

c. Node value vectors

Figure 7. SOAP tree filter vector representations.

Note that a classic solution to the problem of combining different filters is to apply them independently, and then intersect the resulting candidate sets [19]. With such an approach, separate index structures for the different filters have to be maintained and for each filtering task, a time-consuming intersection step is necessary. In addition, all filters functions would be equally weighted regardless of their relative importance. To overcome those disadvantages, we follow a different approach, combining the filter functions in one integrated Sim_{Filter} measure, weighting each function based on its discriminative power over the SOAP tree candidate set. We do so by computing the variance for each filter function over all SOAP trees within the candidate set, and normalizing each function accordingly. This brings filter similarities according to different features (parent-child relations, node order and node values) in a similar range, and assigns a larger weight to features that are a good discriminator for the specific set of candidate SOAP trees at hand. Formally:

$$\mathrm{Sim}_{\text{Filter}}(S_1, S_2) = \frac{1}{\sum_{h \in F}\left(1 - \nabla_h^{\,2}\right)} \sum_{f \in F}\left(1 - \nabla_f^{\,2}\right) Sim_f(S_1, S_2) \quad \in [0,1] \qquad \textbf{(3)}$$

where $F=\{n\text{-}edge, n\text{-}order, n\text{-}value\}$ is the set of component filters, $Sim_f(S_1, S_2)$ is the similarity function between SOAP trees S_1 and S_2 for a given filter component $f \in F$, and $\nabla_f^{\,2}$ is the variance over all SOAP trees according to the f-th filter function within the SOAP tree candidate set.

Note that the combined filter measure Sim_{Filter} is consistent with Definitions 1 and 2, since each of its component filter functions is an upper bound of our main TED measure (cf. [45] for mathematical proof).

2) Tree Edit Distance Similarity Measure

In our SOAP multicasting approach, we exploit a variation of the classic tree edit distance developed by Chawathe in [7]. Hereunder the basic definition of tree edit distance [7, 53]:

Definition 7 – Tree Edit Distance: The edit distance between two trees A and B is defined as the minimum cost of all edit scripts (*diffs*) that transform A to B, $TED(A, B)=Min\{Cost_{Diff}(A, B)\}$ ●

Definition 8 – Edit Script - Diff: It is a sequence of edit operations $Diff = \prec op_1, op_2, \ldots, op_k \succ$, transforming one tree into another. The cost of an edit script is defined as the sum of the costs of its operations: $Cost_{Diff} = \sum_{i=1}^{|Diff|} Cost_{Op_i}$ ●

We chose Chawathe's algorithm [7] since: i) it has been considered as a reference point for various XML related comparison studies [12, 34], ii) it is among the fastest and least complex TED algorithms available [44], and iii) it guarantees correct results (minimal *diffs*) in comparison with existing works, e.g., [8, 11], which utilize various heuristics to gain in performance. Chawathe's algorithm [7] exploits three basic edit operations: *node insertion*, *node deletion* and *node update*, disregarding more complex operations such as *move node*, *insert sub-tree*…, so as to increase efficiency. In our current approach, we intuitively assign identical unit costs to each operation ($Cost_{Ins}=Cost_{Del}=Cost_{Upd}=1$). Note that the

investigation of alternative cost models (e.g., considering the semantic relatedness of SOAP labels/values given a reference such as Wikipedia) will be addressed in a dedicated study.

Hence, given two SOAP trees S_1 and S_2, we compute their similarity based on the tree edit distance function:

$$\mathrm{Sim}_{\mathrm{TED}}(\mathrm{S}_1, \mathrm{S}_2) = 1 - \frac{\mathrm{TED}(\mathrm{S}_1, \mathrm{S}_2)}{|\mathrm{S}_1| + |\mathrm{S}_2|} \quad \in [0,1] \qquad \textbf{(4)}$$

It is to be noted that the classical edit distance similarity formulation, $\frac{1}{1+\mathrm{TED}}$, decreases with the difference between the trees being compared, but does not consider their common parts. Therefore, we utilize the similarity function in Formula (4) in order to capture both the commonalities and the differences between the SOAP trees being compared, so as to increase with commonality and decrease with difference [27]. This is central to identifying the most common pattern among the entities being compared, in order to enable effective SOAP tree aggregation (as discussed in Section III.A).

Let us consider a simplified example based on string edit distance, with S_1 = *'a'*, S_2 = *'axyz'*, S_3 = *'abcd'*, and S_4 = *'abcdxyz'*. Using the classic edit distance formulation, we obtain $Sim(S_1, S_2) = Sim(S_3, S_4)$ since both doublets differ in *'xyz'*. Yet, one can realize that string S_3 and S_4 are more similar than S_1 and S_2, since the latter have *4* characters in common, while the former only share one character. Such commonalities are effectively detected using Formula (4).

Consider the sample SOAP trees in Figure 6. *TED(A, B)=3*, *Diff(A, B)* consisting of three operations: i) updating the value of node *name*, ii) updating the value of *PhoneNum*, and iii) deleting node *CCNum.* Consequently, the result of SOAP tree similarity evaluation and differencing is exploited in building aggregate multicast messages to be sent over the wire.

C. SOAP Buffer Management

1) SOAP Diff Graph Representation

In order to effectively multicast buffered SOAP trees, we represent the latter as a graph-like structure, named *SOAP Diff Graph (SDG)*, connecting SOAP messages (graph nodes) via corresponding *diffs* (graph edges, Figure 8). The buffer consists of multiple *SDG* graphs corresponding to the different buffer pools, each underlining a prospective DSM multicast message.

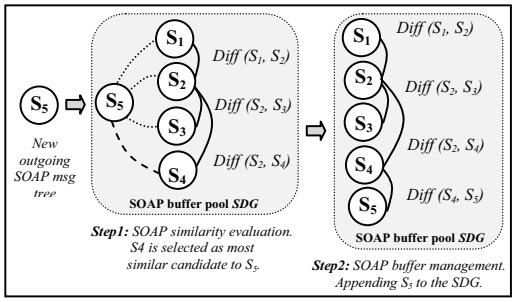


Figure 8. An Example of SOAP buffer management.

As described previously, TED computations for similarity evaluation and *diff* generation are carried out for each new outgoing SOAP message S_{out}, w.r.t. its most similar counterparts in the buffer (i.e., the SOAP tree candidates identified via the *Filter* component). Consequently, the filter candidate S_i maximizing the main similarity measure $Sim_{TED}(S_{out}, S_i)$ is selected. If $Sim_{TED}(S_{out}, S_i) \geq Thresh_{Sim}$, then S_{out} would be appended to the corresponding *SDG* graph, connected to S_i via their common *diff*. Otherwise, if $Sim(S_{out}, S_i)$ drops below $Thresh_{Sim}$, it is allocated a new buffer pool, and constitutes the first node in a new *SDG* graph. When the buffer pool time frame T_{Pool} expires, the corresponding *SDG* is encapsulated in a *DSM* multicast message and is transmitted over the network. A simple example is depicted in Figure 8 to show how an outgoing SOAP message tree S_5, is appended to a SOAP buffer pool *SDG*.

2) DSM Multicast Message

Encapsulating the *SDG* graph into a *DSM* multicast message requires identifying the multicast message *pattern* $S_{pattern}$, which is the most similar and frequent pattern in all messages, minimizing the different parts, i.e., the *diffs*. In other words, it consists in minimizing the multicast message size. Formally:

$$\mathrm{S}_{\mathrm{pattern}} = \mathrm{S}_i \in SDG \text{ verifying } \underset{\forall \mathrm{S}_i \in \mathrm{SDG}}{Min} \left\{ |\mathrm{S}_i| + \sum_j |Diff(\mathrm{S}_i, \mathrm{S}_j)| \right\} \qquad \textbf{(5)}$$

where $|S_i|$ and $|Diff(S_i, S_j)|$ denote the cardinalities (the number of nodes) of the SOAP tree S_i and the *diff* linking S_i and S_j.

This can be performed in linear time w.r.t. the number of SOAP trees in the *SDG* graph, and is achieved by pinpointing the *SDG* node (i.e., SOAP tree) with the maximum number of edges (i.e., *diffs*). The latter, which we identify as *SDG centroid*, underlines the SOAP tree requiring the least amount of transformation operations, i.e., the smallest *diffs*, in order to generate all its remaining counterparts in the *SDG*. In other words, the *SDG centroid* minimizes the differential parts in the encoded *DSM* message, and consequently the overall multicast message size. It identifies the SOAP tree with the maximum amount of commonalities w.r.t. its counterparts.

Consider the *SDG* graph in Figure 8. Here, SOAP tree S_2 is selected as *SDG centroid*, since it is connected to its counterparts with the maximum number of minimal *diffs* (*SDG* edges). Thus, the corresponding *DSM* message consists of tree S_2 as the multicast message *pattern*, and $Diff(S_1, S_2)$, $Diff(S_2, S_3)$, $Diff(S_2, S_4)$ $Diff(S_4, S_5)$ as the differential parts corresponding to each SOAP tree. Recall that our DSM messages follow the same format as SMP messages [37] w.r.t. message header, body, indexing and routing addresses.

3) DSM Message Routing

Our routing process is comparable to that of SMP [37] except that instead of aggregating and splitting common/different parts of the multicast message, the router patches the DSM *pattern*, i.e., *SDG centroid*, with the corresponding *diff* so as to regenerate the original SOAP tree. Consider the example in Figure 8, such as each SOAP tree S_i is intended for a different client C_i. The DSM replicas to be sent to each client consist of:

- The pattern S_2 and $Diff(S_1, S_2)$, to regenerate SOAP tree S_1, destined to client C_1,
- The pattern S_2, destined to client C_2,
- S_2 and $Diff(S_2, S_3)$, to regenerate S_3, destined to client C_3,
- S_2 and $Diff(S_2, S_4)$, to regenerate S_4, destined to client C_4,
- S_2 and $Diff(S_2, S_4) \oplus Diff(S_4, S_5)$, to regenerate S_5, for C_5.

The ⊕ symbol designates the *diff* composition operator, which underlines the transformation of SOAP tree S_2, via two consecutive *diffs*, so as to obtain S_5. In plain terms, it consists in transforming S_2 into S_4 (using $Diff(S_2, S_4)$) and then S_4 into S_5 (via $Diff(S_4, S_5)$) [30].

D. SOAP Message Reconstruction (MR_{DSM})

When the DSM multicast message reaches the destined end-point client/server (or end-point router), the original SOAP message is to be reconstructed, based on the DSM common *pattern* and corresponding SOAP message *diff*, in order to be processed by the destination service component (Figure 2). While *tree differencing* (i.e., tree edit distance) was used as an effective means to perform SOAP aggregation, we exploit its inverse process, *tree patching*, for message reconstruction.

Definition 9 – Tree Patching: It is defined as the problem and action of applying a *diff* to a tree structure (pattern) *T* in order to create a new version of the tree *T'*, incorporating all the changes encoded in the *diff* [22, 33]●

In short, tree patching allows regenerating the original SOAP message tree at the receiver end, by applying the *diff* corresponding to the SOAP message tree, on the common DSM message *pattern*. This comes down to executing the edit operations encoded in the output *diff* on the DSM *pattern*.

Note that in order to allow automatic SOAP tree patching, we defined machine-readable XML-based differential output format, SDL (Simple Diff Language), to be encoded in the multicast DSM message, designed to carry the minimum information necessary to regenerate original SOAP messages at multicast end-point. All details about SDL can be found in the technical report provided in [45].

V. EVALUATION

We conducted extensive simulation experiments to test the performance of our approach, and compare it to *SMP*, *traditional multicast* (aggregating identical messages only), and *unicast*. We evaluated two criteria: i) network traffic (multicast effectiveness), and ii) processing time (efficiency).

A. Network Traffic

We adopted a single sender/receiver scenario, such as the messages are multicast at the sender end-point, and reconstructed at the receiver end-point, disregarding intermediate routers. Hence, network traffic amounts to the sum of the sizes of all SOAP messages over the client/server link. As for the test data, two sets of 500 SOAP messages (each) were generated (of average 4KB per message), based on the Google web service SOAP request and response WSDLs[3], using an adaptation of IBM's XML document generator[4].

We varied three main parameters and evaluated network variation accordingly: the amount of Non-Identical Messages (*NIM %*) to be sent to the client/server, the amount of pair-wise modifications (*Modifs %*) between non-identical messages (which we tuned via the IBM generator), and the number of messages considered for multicasting (*NbMsg*).

[3] http://www.w3.org/2004/06/03-google-soap-wsdl.html
[4] http://www.alphaworks.ibm.com.

1) Network Traffic when varying NIM % and Modifs %

First, we fixed the total number of SOAP messages to be multicast, *NbMsg = 500*, and evaluated network traffic w.r.t. *NIM %* and *Modifs %*. Results in Figure 9 show that our approach (DSM) reduces traffic proportionally to the amount of differences (both *NIM %* and *Modifs %*) among messages. SMP reduces traffic w.r.t. the amount of pair-wise message modifications (*Modif %*), regardless of the amount of non-identical messages (*NIM %*), and thus produces the same 'worst case' results that are obtained via DSM (DSM's upper traffic limit) when none of the messages to be multicast are identical (*NIM %=100*). That happens because SMP only considers the intersection between messages when generating the aggregate multicast, regardless of the largest or most frequent message pattern. Traditional multicast reduces traffic w.r.t. the amount of non-identical messages (*NIM %*), but does not consider partially similar messages (*Modif %*) since it only aggregates identical messages. It produces the 'worst' results obtained using DSM, when messages are completely different (*Modif%=100*). The largest traffic is constantly produced via unicast, since the latter simply transmits messages regardless of their similarities (regardless of both *NIM%* and *Modifs%*).

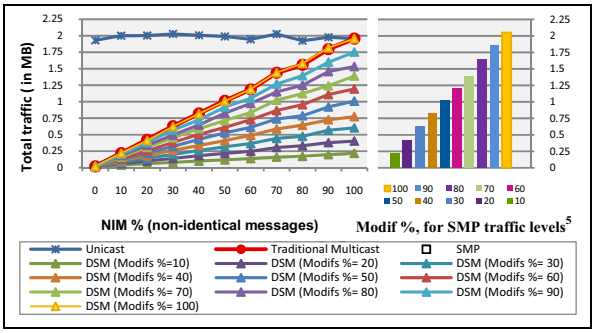
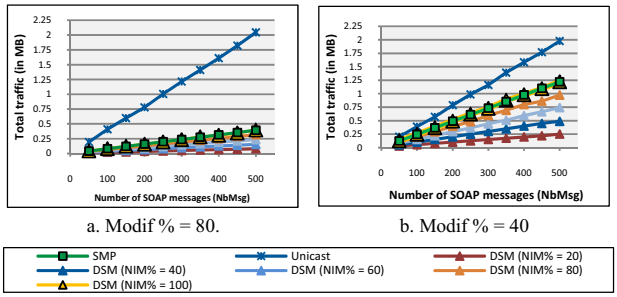


Figure 9. Variation of network traffic w.r.t. the amount of modifications between messages, for a total of *NbM=500* SOAP messages.

2) Varying the number of SOAP messages to be multicast

Figure 10 depicts network traffic when varying the n# of SOAP messages considered for multicasting (*NbMsg*), such as the n# of non-identical messages (*NIM %*) varies linearly w.r.t. the amount of pair-wise message modifications (*Modifs%*).



a. Modif % = 80. b. Modif % = 40

Figure 10. Network traffic variation, w.r.t. the total number of SOAP messages to be multicast, *NbMsg*.

Results confirm those of the previous experiment (Figure 9): i) unicast yields the highest network traffic levels, and remains

[5] SMP traffic levels are invariant w.r.t. NIM %, and are thus represented separately.

unwavering w.r.t. the number of identical and/or similar messages (*NIM %* and/or *Modif %*), ii) traditional multicast only considers identical messages and thus varies w.r.t. *NIM %* iii) traffic with SMP varies w.r.t. *Modif%*, regardless of the amount of non-identical messages *NIM %*, while ii) DSM optimizes traffic w.r.t. both *NIM%* and *Modifs%*.

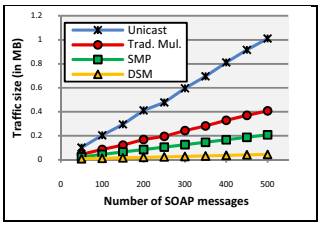


Figure 11. Network traffic, with NIM% =20 & *Modif%=10.*

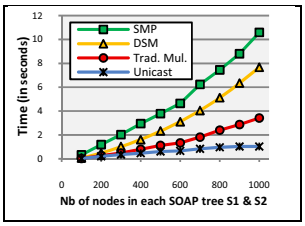


Figure 12. Timing results.

A compact representative depiction of network traffic variation in Figure 11, based on the graphs in Figure 10, for fixed average *NIM %* and *Modifs %* values, shows that the traffic gap between DSM, SMP, traditional multicast, and most evidently unicast, grows noticeably with the increasing number of messages. Results show that DSM underlines an average 20% traffic reduction in comparison with SMP.

B. Processing Time

Timing experiments were carried out on a PC with an Intel Xeon 2.66 GHz processor with 4GB RAM. Results in Figure 12 show that our method induces an average 30% reduction in processing overhead in comparison with SMP. Additional results similar to those in Figure 12 are obtained when varying the total number of SOAP messages being processed. They are omitted here due to space constraints, but are available in [45].

We are currently conducting experiments to fine-tune DSM, varying i) SOAP message aggregation similarity threshold $Thresh_{Sim}$, ii) the number and sizes of multicast buffer pools, and iii) the buffer pool time frame T_{Pool}, to identify the best input parameter values for different kinds of SOAP messages (e.g., encoding numeric data-types, type arrays, etc., [17]) and various multicasting scenarios (w.r.t. the total number of clients, message variability, routing algorithm, etc., [36]). More experimental results can be found in [45], as well as preliminary results on T_{Pool} and $Thresh_{Sim}$.

VI.CONCLUSION

In this paper, we introduced our novel framework for Differential SOAP Multicasting (DSM). Our method encompasses a novel *filter-differencing* architecture, identifying the common pattern and differences between SOAP messages, multicasting those messages which are most similar. Results show that our approach outperforms its alternative, SMP [37], and minimizes network traffic in comparison with traditional multicast and unicast. We are currently investigating the integration of our technique with optimizations of underlying protocols, e.g. sending SOAP multicasts over persistent HTTP links on high-latency networks [20]. We also plan to investigate multicasting of secure SOAP messages, for service security performance [48].

REFERENCES

[1] Abu-Ghazaleh N. and Lewis M.J., *Differential Deserialization for Optimized SOAP Performance.* Proceedings of the ACM/IEEE Conf. on Supercomputing, 2005, pp. 21-31.
[2] Abu-Ghazaleh N. *et al.*, *Differential Serialization for Optimized SOAP Performance.* Inter. Symposium on High Performance Distributed Computing (HPDC), 2004, 55-64.
[3] Azzini A. *et al.*, *Extending the Similarity-Based XML Multicast Approach with Digital Signatures.* ACM Workshop on Secure WS, 2009, 45-52, Chicago.
[4] Bille P., *A Survey on Tree Edit Distance and Related Problems.* TCS, 2005. 337(1):217-239.
[5] Bray T. *et al.*, *Extensible Markup Language (XML) 1.0 - 5th Edition. W3C recommendation, 2008*. http://www.w3.org/TR/REC-xml/ [cited Jan. 2011].
[6] Buttler D., *A Short Survey of Document Structure Similarity Algorithms.* ICOMP'04. pp. 3-9.
[7] Chawathe S., *Comparing Hierarchical Data in External Memory.* VLDB, 1999. pp. 90-101.
[8] Chawathe S. *et al.*, *Change Detection in Hierarchically Structured Information.* ACM Inter. Conf. on Management of Data (SIGMOD), 1996, 26-37. Montreal.
[9] Cheney J., *Compressing XML with Multiplexed Hierarchical PPM Models.* Proceedings of the Data Compression Conference (DCC'01), 2001. pp. 163-173.
[10] Chinnici R. *et al.*, *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, 2007, http://www.w3.org/TR/wsdl20/* [cited Jan. 2011].
[11] Cobéna G.; Abiteboul S.; and Marian A., *Detecting Changes in XML Documents.* IEEE Inter. Conf. on Data Engineering (ICDE'02), 2002. pp. 41-52.
[12] Dalamagas T. *et al.*, *A Methodology for Clustering XML Documents by Structure.* Information Systems, 2006. 31(3):187-228.
[13] Damiani E. and Marrara S., *Efficient SOAP Message Exchange and Evaluation Through XML Similarity.* ACM Workshop on Secure WS, 2008. pp.29-36.
[14] Davey B. A. and Priestley H. A., *Introduction to Lattices and Order (2nd Edition).* Cambridge University Press, pp. 310., 2002.
[15] Devaram K. and Andersen D., *SOAP Optimization via Parameterized Client-Side Caching.* IEEE/ACM CCGRID'02 Symposium, 2002. pp.439-312.
[16] Gannon D. *et al.*, *On Building Parallel and Grid Applications: Component Technology and Distributed Services.* IEEE CLADE '04, p. 44, Washington DC.
[17] Head M.R. et al., *A Benchmark Suite for SOAP-based Communication in Grid Web Services.* Proceedings of the ACM/IEEE Conference on Supercomputing, 2005. pp. 19, Seattle.
[18] Horstmann M. and Kirtland M. *DCOM Architecture*. Microsoft MSDN, http://msdn.microsoft.com/en-us/library/ms809311.aspx, 1997 [cited Jan. 2011].
[19] Kailing K.; Kriegel H.P.; Schonauer S.; and Seidl T., *Efficient Similarity Search for Hierarchical Data in Large Databases.* EDBT'04, 2004. pp. 676-693.
[20] Kangasharju J. *et al.*, *Comparing SOAP Performance for Various Encodings, Protocols, and Connections.* IFIP 8th Inter. Conf., PWC'03, 2003, pp. 397-406.
[21] Kohlhoff C. and Steele R., *Evaluating SOAP for High Performance Business Applications: Real-Time Trading Systems.* Proc. of the WWW Conf., 2003. Budapest.
[22] Komvoteas K., *XML Diff and Patch Tool.* MS in Distributed Multimedia and Information Systems Dissertation, Edinburgh, Heriot-Watt University, 2003.
[23] Korn F.; Sidiropoulos N.; Faloutsos C.; Siegel E. and Protopapas Z., *Fast and Effective Retrieval of Medical Tumor Shapes. I.* IEEE TKDE 1998, 10(889:904).
[24] Krause E.F., *Taxicab Geometry - An Adventure in Non-Euclidean Geometry.* Dover Publications - NY, 1987. pp. 88.
[25] Kriegel H.P. and Schönauer S., *Similarity Search in Structured Data.* DaWaK 2003, 309-319.
[26] Liefke H. and Suciu D., *XMill: An Efficient Compressor for XML Data.* University of Pennsylvania Technical Report MSCIS-99-26., 2000.
[27] Lin D., *An Information-Theoretic Definition of Similarity.* Inter. ICML Conf., 1998, 296-304.
[28] Ma Y. and Chbeir R., *Content and Structure Based Approach for XML Similarity.* Inter. Conf. on Computer and Info. Tech. (ICCIT), 2005, 136-140.
[29] Makino S. *et al.*., *Improving WS-Security Performance with a Template-Based Approach.* IEEE International Conference on Web Services (ICWS'05), 2005. pp. 581-588.
[30] Marian A. *et al.*., *Change-Centric Management of Versions in an XML Warehouse.* Proc. of the VLDB Conf., 2001. pp. 581-590.
[31] McGill M., *Introduction to Modern Information Retrieval.* 1983. McGraw-Hill.
[32] Megginson *et al.*, *The Simple API for XML*, http://www.megginson.com/SAX/.
[33] Mouat A., *XML Diff and Patch Utilities.* MS Dissertation, 2002, Heriot-Watt Univ, Sc.
[34] Nierman & Jagadish, *Evaluating structural similarity in XML documents.* WebDB '02, 61-66.
[35] Object Management Group. *The Common Object Request Broker: Architecture and Specification.* Version 3.0.3, http://www.omg.org/technology/, 2004.
[36] Phan K.A. *et al.*, *Minimal Traffic-Constrained Similarity-Based SOAP Multicast Routing Protocol*. OTM Confederated Inter. Conf., 2009. LNCS, pp. 558-576.
[37] Phan K.A. *et al.*, *Similarity-Based SOAP Multicast Protocol to Reduce Bandwidth and Latency in Web Services.* IEEE TSC, 2008, 1:2(88-103).
[38] Salton G., *Automatic text processing: the transformation, analysis, and retrieval of information by computer.* Addison-Wesley Longman, Boston, 1989. pp. 530.
[39] Singh G. *et al.*, *A Metadata Catalog Service for Data Intensive Applications*. Proceedings of the ACM/IEEE Conference on Supercomputing, 2003, 33, Washington DC
[40] Slominski A. *XSOAP*. 2004, http://www.extreme.indiana.edu/xgws/xsoap/ [cited Jan. 2011].
[41] Sun. *Java Remote Message Invocation (RMI)*. http://java.sun.com/j2se/1.5.0/rmi/, 2005
[42] Suzumura T. *et al.*, *Optimizing Web Services Performance by Differential Deserialization* IEEE Conference on Web Services (ICWS'05), 2005. (1):185- 192.
[43] Takeuchi Y. *et al.*, *A Differential-Analysis Approach for Improving SOAP Processing Performance.* IEEE Conf. on e-Tech., e-Com. and e-Service (EEE), 2005, 472-479.
[44] Tekli J.; Chbeir R. and Yétongnon K., *An Overview of XML Similarity: Background, Current Trends and Future Directions.* Elsevier Computer Science Review, 2009. 3(3):151-173.
[45] Tekli J. *et al.*, Tech. Rep. DSM-TR-10, 2010, http://dbconf.u-bourgogne.fr/DSM-TR-10.pdf.
[46] Tekli J.; Damiani E.; Chbeir R. and Gianini G., *SOAP Processing Performance and Enhacement.* To appear in IEEE Transactions on Services Computing (IEEE TSC), 2011.
[47] Teraguchi M. *et al.*, *Optimized Web Services Security Performance with Differential Parsing.* International Conference on Service-Oriented Computing (ICSOC'06), 2006. pp. 277-288.
[48] Turkmen F. and Crispo C., *Performance evaluation of XACML PDP implementations.* ACM workshop on Secure Web Services (SWS), Virginia, USA., 2008. pp. 37-44.
[49] Werner C. *et al.*, *WSDL-Driven SOAP Compression.* J. of WS Research, 2005. 2(1):18-35.
[50] WWW Consortium. *SOAP Version 1.2*, 2007, http://www.w3.org/TR/soap/, [cited Jan. 2011]
[51] WWW Consortium. *The Document Object Model*. http://www.w3.org/DOM [cited Jan. 2011]
[52] Zhang B.; Jamin S. and Zhang L., *Host Multicast: A Framework for Delivering Multicast to End Users.* IEEE Conf. on Computer Communications (INFOCOM'02), 2002, 1366-1375.
[53] Zhang K. and Shasha D., *Simple Fast Algorithms for the Editing Distance between Trees and Related Problems.* SIAM Journal of Computing, 1989. 18(6):1245-1262.