An Optimization Rule for ActiveXML Workflows

Sattanathan Subramanian and Guttorm Sindre

Department of Computer and Information Science,
Norwegian University of Science and Technology (NTNU),
NO-7491 Trondheim, Norway
{sat,guttors}@idi.ntnu.no

Abstract. Web services are used as the de facto standard to develop the modern business applications that require collaboration and coordination among the business partners. ActiveXML (AXML) is viewed as a data-oriented workflow language for specifying the Web service calls and their interactions. The present workflow engines execute the workflow specifications *strictly* without attempting to optimize the Web service calls. This paper proposes an optimization approach as a rule called $\mathcal{SCG}$, to improve the performance of workflows in the context of AXML. The $\mathcal{SCG}$ groups the Web service calls of AXML documents, and reuses the existing results of other equivalent Web service calls.

Keywords: ActiveXML, Optimization, Web service, Workflow.

1 Introduction and Related Work

Workflow systems have been in research [11, 13] as well as industry [1] for many years to support the collaboration among the business processes. The workflows are often implemented as Web services to cross the boundaries of networks, architectures, platforms, and organizations [7, 8]. A Web service (*service*, hereafter) based workflow typically contains a number of service calls to perform its tasks. Due to the control flow nature of workflow systems [14], the service calls are *strictly* invoked in an order given in the workflow specification without attempting any optimizations, even when less expensive workflows could be devised that obtain the same results. Hence, the execution of many such workflows consumes more time and resources (e.g., bandwidth) than necessary, a clear weakness of the state-of-the-art. Emphasizing dataflow instead of control flow could help to relax the execution of workflows. The Active XML (*AXML*) [3] platform facilitates the *data-oriented* workflows, where the dataflow is considered as the processing key. An AXML document is an XML document with the embedded service calls to the Web services, executed in a peer-to-peer architecture (`www.activexml.net`). The present optimization framework of AXML called OptimAX [4, 5, 6] has the rules to delegate and instantiate the service calls among the peers, to compose and decompose the simple queries that are available directly, and to eliminate the redundant computation. This paper extends the OptimAX framework by proposing a new rule called $\mathcal{SCG}$ to optimize the number of service calls in the

M. Gaedke, M. Grossniklaus, and O. Díaz (Eds.): ICWE 2009, LNCS 5648, pp. 410–418, 2009.

AXML documents. Our optimization approach is leveraged from the continuous query optimization [10] of database systems to the AXML by correlating the continuous queries as service calls, which groups the service calls of one with another based on their data source (i.e., Web service) and reuses the existing results received from other equivalent service calls. This improves the execution complexity of the *peer* that executes the service calls of an AXML document, of the *Web service* that provides the actual service, and of the *database* connected behind the Web service. The framework proposed in [13] optimizes a BPEL based workflow into a single SQL activity. Whereas, we specifically intend to optimize the service calls of workflow in the same form. This widens the scope of optimization across the workflows and minimizes the changes required in the execution environment. Lazy query evaluation [2] evaluates the user queries and detects the relevant service calls in an AXML document that can bring data. The $\mathcal{SCG}$ is intended to be applied after detecting such relevant service calls from the AXML document.

We consider a four-tier architecture for the purpose of this paper, i.e., the *clients* to request workflow applications, *AXML peer* to execute the respective AXML document, and *Web services* to process the service requests of AXML peer and deliver back the results after accessing its *databases*. Finally, the AXML peer delivers the results to the respective clients. This paper considers the service calls that carry queries (e.g., SQL) of selection and projection, and one-stage execution of AXML documents, i.e., the new service call sc_i which is returned from the activation of another service call sc_{i-1} is omitted.

The paper is organized as follows: Section 2 shows a motivating example. Section 3 illustrates the idea of $\mathcal{SCG}$ through some examples of relational algebra and shows its algebraic execution. Section 4 concludes the paper.

2 Motivating Example

Consider that there is a *FlightEnquiry* AXML document (shown in the Fig. 1), which is meant for comparing the prices and availability of flights as per the travel requirements of the user. The input of *FlightEnquiry* is `travelDetails` like starting point, destination, date, etc, and output is `flightDetails` like

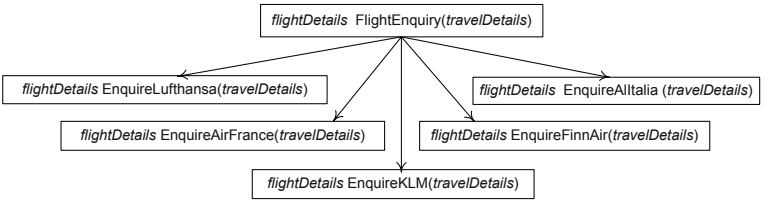

Fig. 1. FlightEnquiry Workflow

price, availability, etc. *FlightEnquiry* has five independent service calls targeting five different flight Web services. Those are, *EnquireLufthansa*, *EnquireAirFrance*, *EnquireKLM*, *EnquireFinnAir*, and *EnquireAlItalia*. All these service calls carry the same input and output of *FlightEnquiry*, i.e., `travelDetails` as input and `flightDetails` as output. *FlightEnquiry* is potentially accessible for many users in the Internet through some graphical interfaces. So, many (thousands of) *FlightEnquiry* requests can reach the AXML peer simultaneously. In such situations, the AXML peer and the respective Web services are heavily loaded since all the service calls of the AXML document are processed separately. We understand that the load balancing [9] techniques provide a single domain name from multiple servers by consolidating many HTTP requests of various clients as a single TCP socket to the back-end servers. But, this cannot consolidate the service calls and their associated queries to optimize the execution in the AXML peer, Web services, and databases. Similarly, web caching techniques are useful if the queries associated with the service calls are unique, however, the queries tend to be different in real due to the personalization of users over the web content [12]. For the purpose of discussion, we consider that an AXML peer receives one thousand *FlightEnquiry* requests simultaneously. These one thousand requests internally require five thousand service calls, i.e., one thousand to *EnquireLufthansa*, one thousand to *EnquireAirFrance*, etc. The AXML peer and the servers of Web services like *EnquireLufthansa* are heavily loaded in this case. We also consider that the *EnquireLufthansa* service is implemented on top of a `LufthansaDB` database which has the details of flights, availability, price, etc. So, one thousand *EnquireLufthansa* requests additionally produce one thousand data requests to the `LufthansaDB`. Assuming that the workflow requests arrive at sufficiently close moments in time and their answers can be forcedly synchronized with no perceived disadvantage to the user, the execution complexity of one thousand instances of *FlightEnquiry* can be reduced if the service calls targeting the same service are grouped as "one", i.e., only *one* service call to *EnquireLufthansa* to obtain the required results of one-thousand service calls. Similarly for *EnquireAirFrance*, *EnquireKLM*, *EnquireFinnAir*, and *EnquireAlItalia*. This implicitly optimizes the data requests to the respective databases from the Web services in the same way.

3 Service Calls Grouping

3.1 Idea Illustration

This paper groups many independent service calls of AXML document that target the same Web service into *one* service call. The state-of-the-art way of executing service calls of AXML documents is shown in Fig. 2(a), i.e., independent service calls ($sc_1, sc_2, ..., sc_n$; hereafter we refer this list as $sc_{1..n}$) are activated and its results are received individually although those calls target the same Web service. Also, it can be noticed that each Web service request opens an additional database request as shown in Fig. 2(a). The proposed optimization is shown in Fig. 2(b), where all the independent service calls are grouped as one

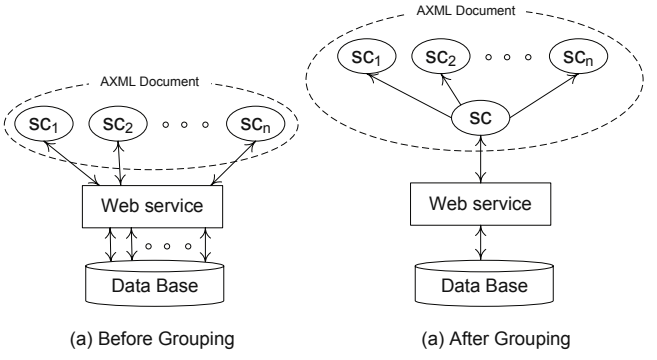

Fig. 2. Idea of Grouping Service calls

service call (i.e., *sc*), and similarly to the database access from the Web service. The *sc* carries all the data constraints of $sc_{1..n}$. The activation of *sc* receives the results required for $sc_{1..n}$, which are accessed and filtered by $sc_{1..n}$ according to their individual data constraint.

The service calls are grouped by encoding their input parameters as XML trees. The input parameter has a *condition* and a finite set of *output attribute names* for which the values are expected from the Web service. Fig. 3(a) shows the generalized encoded tree format of the service call's input parameter. In that, the *conditional attribute names and its values* are located in the leaf position, the *relational operator* is located as the parent of conditional attribute name and its value, the *logical operator* is located as the parent of relational operator, and the *output attribute names* are located as the parent of logical operator (or relational operator). The output attribute names are always located in the top level of the encoded tree, and the relational operator can be the immediate child of output attribute name when there is no logical operator in the condition. Fig. 3(b) shows the encoded tree form of the following service call's input parameter:

$$\Pi_{ws.weather}(\sigma_{ws.country=\text{“}France\text{”}\ and\ ws.city=\text{“}Paris\text{”}}(WeatherService\ ws)).$$

For the sake of simplicity, we use the relational algebra to represent the input parameters of service call including the service name. Also, we use generic

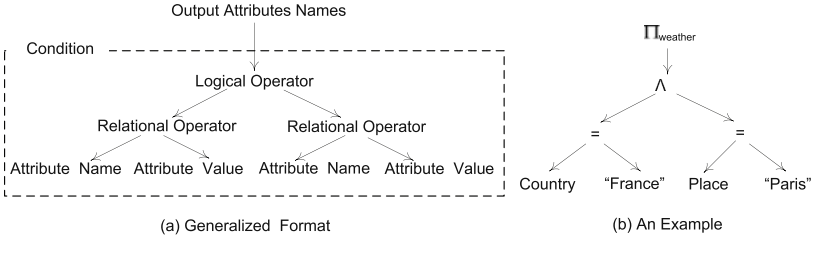

Fig. 3. Encoded Input Parameter of a Service Call

examples in this section to illustrate all possible cases of grouping service calls. In the above example, $ws.country$ = "$France$" *and* $ws.city$ = "$Paris$" is the condition and $ws.weather$ is the output attribute name.

Two (or more) independent service calls are grouped using any of the following two cases,

Gr_1 : `same input parameters and target.` The input parameters of service calls are *exactly same* including the targeted Web service. For example,

$$sc_a = \Pi_{ss.stockRate}(\sigma_{ss.stockName=\text{"}Dell\text{"}}(StockService\ ss)).$$
$$sc_b = \Pi_{ss.stockRate}(\sigma_{ss.stockName=\text{"}Dell\text{"}}(StockService\ ss)).$$

The sc_a and sc_b have the same condition (i.e., $ss.stockName$ = "$Dell$"), output attribute name (i.e., $ss.stockRate$), and target (i.e., $StockService$). So, here sc_a or sc_b can be considered as the input parameter of grouped service call.

Gr_2 : `related input parameters but same target.` The input parameters of service calls are different from each other, but the targeted service is same for both. The difference can be in the condition or output attribute name or both. Between the conditions, the difference can be in relational or logical operator or both, or conditional attribute name or its value or both, or both. For example,

$$sc_c = \Pi_{ss.stockRate}(\sigma_{ss.stockName=\text{"}Yahoo\text{"}}(StockService\ ss)).$$

The conditional attribute value differs between sc_a (or sc_b) and sc_c, i.e., sc_a has "$Dell$" as the value for $stockName$, whereas, sc_c has "$Yahoo$" as the value for $stockName$. So, the sc_a and sc_c are grouped as,

$$sc = \Pi_{ss.stockRate}(\sigma_{ss.stockName=\text{"}Dell\text{"}\ or\ \text{"}Yahoo\text{"}}(StockService\ ss)).$$

The sc joins the conditions of sc_a and sc_c as $ss.stockName$ = "$Dell$" *or* "$Yahoo$", and the output parameter name, i.e., $stockRate$. Now, we consider another input parameter of a new service call,

$$sc_d = \Pi_{ss.stockRate,ss.noOfAvailableStocks}(\sigma_{ss.stockName=\text{"}Yahoo\text{"}}(StockService\ ss)).$$

Here, the output attribute name differs between sc and sc_d, i.e., sc has $ss.stockRate$, whereas, sc_d has $ss.stockRate$ and $ss.noOfAvailableStocks$. It can be observed that the condition of sc is the superset of sc_d's condition. So, sc and sc_d are grouped as,

$$sc = \Pi_{ss.stockRate,ss.noOfAvailableStocks}(\sigma_{ss.stockName=\text{'}Dell\text{'}\ or\ \text{'}Yahoo\text{'}}(StockService\ ss)).$$

Now sc has two output attribute names (i.e., $ss.stockRate$ and $ss.noOfAvailableStocks$) to get the required results of sc_a, sc_b, sc_c and sc_d.

The conditional attribute names of respective input parameters (i.e., sc_a, sc_b, sc_c and sc_d) are added as the output attribute names of sc if it is not available

already, i.e., $ss.stockName$ is added as a new output parameter name. The updated sc is,

$$sc = \Pi_{ss.stockRate,ss.noOfAvailableStocks,ss.stockName} (\sigma_{ss.stockName=\text{“}Dell\text{”} \ or \ \text{“}Yahoo\text{”}}(StockService\ ss)).$$

This is done for filtering the service results after the activation of a grouped service call. The intended results of sc_a, sc_b, sc_c and sc_d are filtered from the service results (say, R) of sc after its activation. This filtering is done based on the input parameter of the respective service call. In our example,

$$sc_a,\ sc_b = \Pi_{stockRate}(\sigma_{stockName=`Dell'}(R))$$
$$sc_c = \Pi_{stockRate}(\sigma_{stockName=`Yahoo'}(R))$$
$$sc_d = \Pi_{stockRate,noOfAvailableStocks}(\sigma_{stockName=`Yahoo'}(R))$$

3.2 Rule for AXML

Consider that there are n independent service calls $sc_{1..n}$ in the AXML document d located at the peer p, targeting the Web service ws located at the peer p_1, with the input parameters $t_1, t_2, ..., t_n$ (hereafter, we refer this list as $t_{1..n}$) respectively. For this, the $\mathcal{SCG}$ rule is,

$$eval@p(\sharp x_0@p\langle sc_1@p_1(t_1)\rangle, \sharp x_1@p\langle sc_2@p_1(t_2)\rangle, ..., \sharp x_n@p\langle sc_n@p_1(t_n)\rangle) \rightarrow \sharp x_0@p\langle filter@p(\sharp y_0@p, t_1)\rangle, \sharp x_1@p\langle filter@p(\sharp y_0@p, t_2)\rangle, ..., \sharp x_n@p\langle filter@p(\sharp y_0@p, t_n)\rangle, \sharp y_0@p\langle eval@p(sc@p_1(t))\rangle$$

where, $eval$ is an operation that evaluates the service calls of d, sc is the grouped service call which is grouped from $sc_{1..n}$, t is the input parameter which is grouped from $t_{1..n}$, $filter$ is a newly proposed operation available in the peer p to filter the service results of sc. The optimization is the the first stage of $eval$, which creates sc and replaces $sc_{1..n}$ with $filter$s. The inputs of the $filter$ operation are the service results of sc (in our case, it is available at $\sharp y_0@p$) and the input parameter of respective service call which needs the partial results from sc. The pictorial view of this optimization rule is shown in Fig. 4, where the bidirectional arrow line indicates the Web service request and response.

The detailed step-by-step algebraic execution of the $\mathcal{SCG}$ is follows,

$$eval@p(\sharp x_0@p\langle sc_1@p_1(t_1)\rangle, \sharp x_1@p\langle sc_2@p_1(t_2)\rangle, ..., \sharp x_n@p\langle sc_n@p_1(t_n)\rangle)$$

Next, the grouped service call $sc@p_1$ is created at $\sharp y_o@p$ with the input parameter t which is created by grouping the input parameters $t_{1..n}$ of $sc_{1..n}$. The service calls $sc_{1..n}$ are replaced with $filter$ operations. Every filter operation takes the service results that are produced by sc and the respective input parameter of service call sc_1 or sc_2 or ... or sc_n.

$$\rightarrow \sharp x_0@p\langle filter@p(\sharp y_0@p, t_1)\rangle, \sharp x_1@p\langle filter@p(\sharp y_0@p, t_2)\rangle, ..., \sharp x_n@p\langle filter@p(\sharp y_0@p, t_n)\rangle, \sharp y_0@p\langle eval@p(sc@p_1(t))\rangle$$

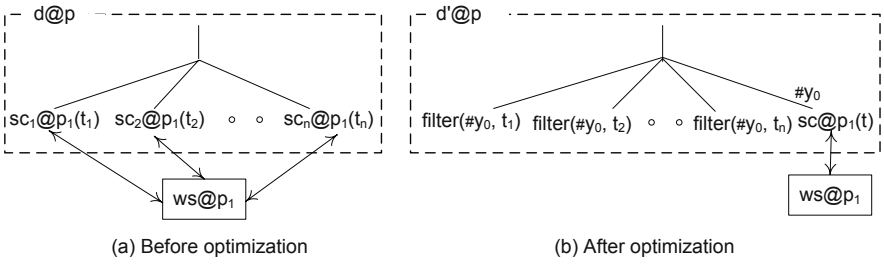

Fig. 4. The $\mathcal{SCG}$ Rule

Next, the $sc@p_1$ is evaluated at p. This results in *send* and *receive* operations from $(new)@p_1$ to $\sharp y_0@p$. Note that the *send* and *receive* are the predefined services locally available in the peer, used to move the streams of XML data from one peer to another [6]. The *send* service has two parameters: `what` (represents the data to be sent from one site to another), and `where` (represents a node id). There are no parameters for *receive*. The symbol ∘ indicates the start of an operation, and • indicates the end of an operation.

$$\rightarrow \sharp x_0@p\langle filter@p(\sharp y_0@p, t_1)\rangle, \sharp x_1@p\langle filter@p(\sharp y_0@p, t_2)\rangle, \\ ..., \sharp x_n@p\langle filter@p(\sharp y_0@p, t_n)\rangle, \sharp y_0@p\langle \circ receive@p()\rangle \ \& \\ (new)@p_1 : eval@p_1(send@p_1(\sharp y_0@p, sc@p_1(t)))$$

Next, the *receive* operation ends its operation.

$$\rightarrow \sharp x_0@p\langle filter@p(\sharp y_0@p, t_1)\rangle, \sharp x_1@p\langle filter@p(\sharp y_0@p, t_2)\rangle, \\ ..., \sharp x_n@p\langle filter@p(\sharp y_0@p, t_n)\rangle, \sharp y_0@p\langle \bullet receive@p()\rangle$$

Next, the $filter$ operations start filtering the result available at $\sharp y_0@p$ based on the respective input parameter t_1 or t_2 or ... or t_n.

$$\rightarrow \sharp x_0@p\langle \circ filter@p(\sharp y_0@p, t_1)\rangle, \sharp x_1@p\langle \circ filter@p(\sharp y_0@p, t_2)\rangle, \\ ..., \sharp x_n@p\langle \circ filter@p(\sharp y_0@p, t_n)\rangle, \sharp y_0@p$$

Finally, the $filter$ ends its operation.

$$\rightarrow \sharp x_0@p\langle \bullet filter@p(\sharp y_0@p, t_1)\rangle, \sharp x_1@p\langle \bullet filter@p(\sharp y_0@p, t_2)\rangle, \\ ..., \sharp x_n@p\langle \bullet filter@p(\sharp y_0@p, t_n)\rangle$$

4 Conclusion and Future Work

This paper has provided an optimization approach for the execution of AXML documents. The proposed $\mathcal{SCG}$ rule groups Web service calls that carry selection and/or projection queries. Our optimization approach relates one service call with another based on its target i.e., Web service, and reuses the existing results that are received from other equivalent service calls. This improves the performance of the *workflow server* and the *Web service* including the *database*

server that provides the data to Web service. The algebraic executions are shown to illustrate the service calls grouping in the framework of AXML.

Presently, the $\mathcal{SCG}$ is limited to the independent service calls. So, this has to be extended to accommodate the update queries (e.g., insert, delete, modify) that are dependent of one with another. The nested transactions of database systems can be correlated to the service calls in order to maintain the ACID properties while grouping. The open questions related to the performance of $\mathcal{SCG}$ are: (i) what is the execution improvement of an AXML document in a real application?, (ii) what extent the workload of Web service and its associated database server is reduced?, (iii) what is the efficient threshold time of the workflow server to wait and accumulate the workflow requests for applying $\mathcal{SCG}$ in a wider range?, (v) how many service calls can be grouped together in order to match the capacity of a server?, and (vi) how far the $\mathcal{SCG}$ is efficient than the classical web caching techniques especially when the queries are identical?

Acknowledgements

This work was carried out during the tenure of an ERCIM "Alain Bensoussan" Fellowship Programme. First author would like to thank Serge Abiteboul and Ioana Manolescu of INRIA for their initial ideas and suggestions.

References

[1] Aalst, W.M.P.: Trends in business process analysis - from verification to process mining. In: The proceedings of International Conference on Enterprise Information Systems (ICEIS), Madeira, Portugal (2007)

[2] Abiteboul, S., Benjelloun, O., Cautis, B., Manolescu, I., Milo, T., Preda, N.: Lazy query evaluation for Active XML. In: The proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD), Paris, France (2004)

[3] Abiteboul, S., Benjelloun, O., Milo, T.: The Active XML project: an overview. The VLDB Journal 17(5) (2008)

[4] Abiteboul, S., Manolescu, I., Taropa, E.: A Framework for Distributed XML Data Management. In: The proceedings of International Conference on Extending Database Technology (EDBT), Munich, Germany (2006)

[5] Abiteboul, S., Manolescu, I., Zoupanos, S.: OptimAX: optimizing distributed continuous queries. In: The Proceedings of International Conference on Bases de Donnees Avancees (BDA), Marseille, France (2007) (A French Conference)

[6] Abiteboul, S., Manolescu, I., Zoupanos, S.: OptimAX: Optimizing Distributed ActiveXML Applications. In: the Proceedings of Eighth International Conference on Web Engineering (ICWE), New York, USA (2008)

[7] Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services, Version 1.1. Standard proposed by BEA Systems. IBM Corporation, and Microsoft Corporation (2003)

[8] Benatallah, B., Sheng, Q.Z., Dumas, M.: The Self-Serv Environment for Web Services Composition. IEEE Internet Computing 7(1) (January/February 2003)

[9] Bourke, T.: Server load balancing. O'Reilly & Associates, Inc., Sebastopol (2001)
[10] Chen, J., DeWitt, D.J., Tian, F., Wang, Y.: NiagaraCQ: a scalable continuous query system for Internet databases. In: The Proceedings of ACM International Conference on Management of data (SIGMOD), Dallas, Texas, United States (2000)
[11] Hull, R., Su, J.: The Vortex Approach to Integration and Coordination of Workflows (A position paper). In: The Proceedings of International Joint Conference on Work Activities Coordination and Collaboration (WACC), San Francisco (1999)
[12] Sivasubramanian, S., Pierre, G., Steen, M., Gustavo, G.: Analysis of Caching and Replication Strategies for Web Applications. IEEE Internet Computing 11(1) (January 2007)
[13] Vrhovnik, M., Schwarz, H., Suhre, O., Mitschang, B., Markl, V., Maier, A., Kraft, T.: An Approach to Optimize Data Processing in Business Processes. In: Proceedings of International Conference on Very Large Data Bases (VLDB), Vienna, Austria (2007)
[14] Wang, J., Kumar, A.: A Framework for Document-Driven Workflow Systems. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 285–301. Springer, Heidelberg (2005)