Distributed Invocation of Composite Web Services

Chang-Sup Park[1] and Soyeon Park[2]

[1] Department of Internet Information Engineering,
University of Suwon, Korea
park@suwon.ac.kr
[2] Department of Computer Science,
University of Illinois at Urbana-Champaign
soyeon@uiuc.edu

Abstract. Web services provide a useful means to integrate heterogeneous applications distributed over the Internet based on XML and Web technologies. This paper presents an approach to execute hierarchically interacting web services efficiently. We provide a system architecture which can distribute invocations of web services over service providers by exploiting intensional XML messages embedding external web service calls. We also propose a greedy heuristic algorithm to generate an efficient strategy of executing web service calls, which can improve overall performance of distributed web service systems on the Internet.

1 Introduction

Web services are emerging as a major technology for integration of heterogeneous applications distributed over the Internet. They are self-contained, modular units of application logic which provide business functionality to other applications via Internet connections based on XML messages and Web protocols. Applications can be assembled from a set of appropriate web services and do not need to be developed from scratch. A new web service also can be composed of existing web services, which function as both clients and servers to the other web services.

Successive composition of web services builds a hierarchical structure of interactions among a large number of web services. By using *intensional* XML messages containing calls to external web services, a web service can delegate the invocation of a web service to the other one. XML-based SOAP message and protocol used for web service invocation involve clients, as well as servers, considerable performance overhead [5, 7]. Since the peer nodes have different capability and performance depending on their computing resource and workload, and communication cost between a pair of peer nodes are also different, it is desirable to select one which can execute the web service invocation most efficiently.

This paper proposes a method for executing invocation of composite web services efficiently which interact hierarchically and can return intensional results. The proposed method effectively distributes the workload on activating and completing web service calls by exploiting intensional XML data as input and output messages of web

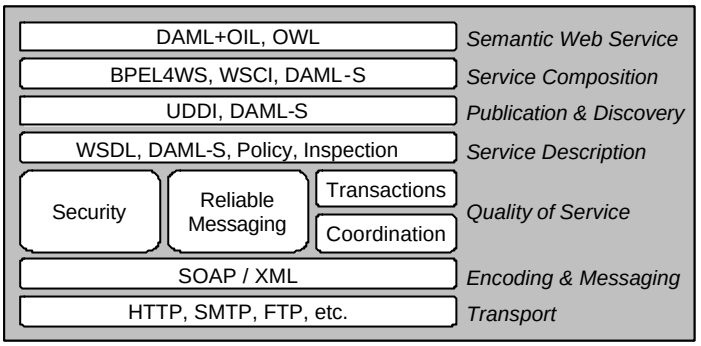


Fig. 1. Web service protocol stack

services. We provide a cost-based optimization technique to enhance overall performance of the entire web service systems. We also utilize user agents to support various kinds of web clients with different capabilities and constraints.

The paper is organized as follows. In Section 2, previous work on web service composition and application of intensional data is provided. System architecture for composite web services exchanging intensional data is presented in Section 3, and an algorithm for generating an efficient invocation strategy for web services considering intensional data is proposed in Section 4. Section 5 draws conclusion with some future work.

2 Backgrounds and Related Work

Deployment of web services is supported by various standards including Web Service Description Language (WSDL), Universal Description, Discovery, and Integration (UDDI), and Simple Object Access Protocol (SOAP). These standards respectively support definition of the functions and interfaces of web services, advertisement of web services to the community of potential user applications, and binding for remote invocation of web services [14]. Fig. 1 shows a set of suggested protocols and languages comprising web services technologies.

As a useful means to realize the service-oriented architecture paradigm in software development, extensive researches and developments have been done on the web services including web service composition, semantic description and discovery of web services, and improvement of quality of services [4]. While the business logic of a web service can be implemented in general procedural programming languages, special languages have been proposed in industries which make use of process models to describe web service composition effectively, such as Web Service Flow Language (WSFL), XLang, Business Process Execution Language for Web Services (BPEL4WS), and Web Service Choreography Interface (WSCI) [13]. Researches on the semantic web such as DAML-S ontology language support automated discovery, execution, composition, and interoperation of web service [10]. For enhancing quality

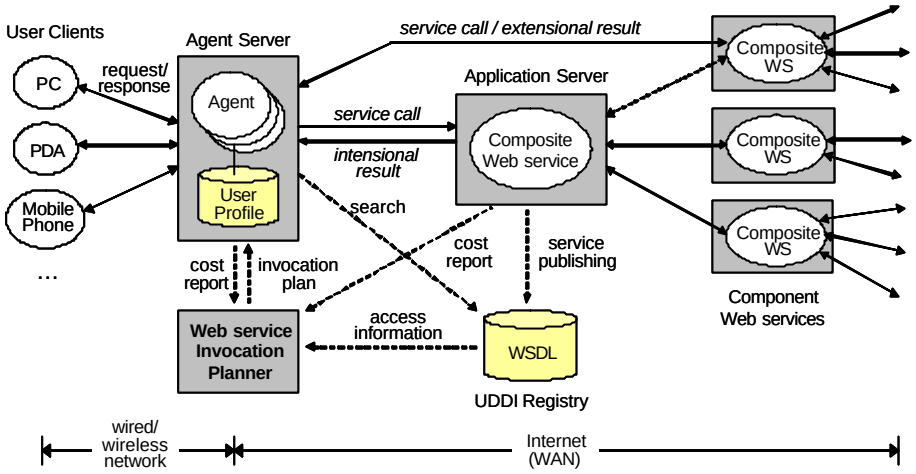


Fig. 2. System architecture

of composite web services, L. Zeng et al. [15] proposed a general QoS model for web services and an algorithm for selecting optimal web services in composition of a new service. A. Mani and A. Nagarajan [9] specified various kinds of QoS requirements for web services and techniques for satisfying them.

Recently, there have been proposed several approaches and applications exploiting intensional data which embeds calls to external web services [2, 3, 8]. Active XML [2] is a framework for data integration in a peer-to-peer environment which uses web services exchanging intensional data called Active XML documents. Each peer node provides Active XML services allowing clients to access the Active XML documents stored in the peer's repository. Active XML services can be specified as parameterized queries over Active XML documents which contain calls to external web services provided by other peers. Thus, they can be used to search, gather, and integrate data distributed over the peer-to-peer nodes effectively [1].

3 System Architecture

Fig. 2 shows the execution environment of composite web services considered in this paper. Composite web services interact with other web services available on the network. Web services can be accessed and invoked from various user client devices. They usually expect composite web services to complete execution and return *extensional* data as a final result whose type was defined by WSDL since they often have difficulties in materializing intensional data embedding service calls due to limitations such as computing power and security constraints. To support such user clients, we adopt software agents which execute independently of other web services and can handle intensional data instead of the clients [6]. They receive service requests from

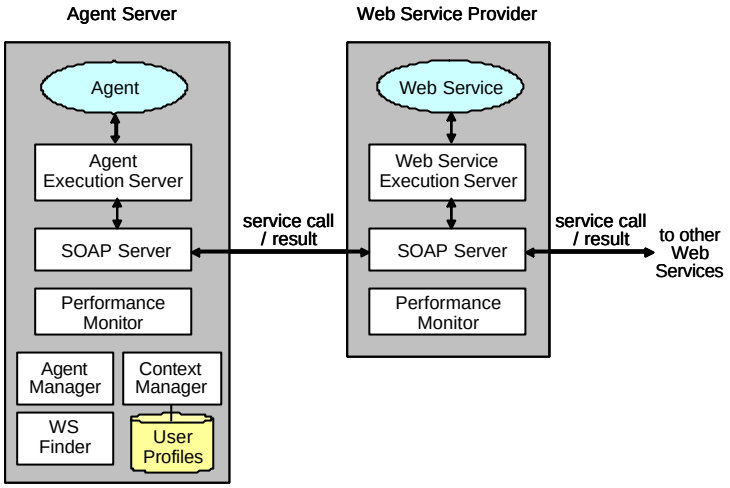


Fig. 3. Architecture of an agent server and a web service provider

user clients, find and call appropriate web services, perform deferred invocation of web services contained in intensional results, and deliver final results to the clients. They are deployed at the location near to clients, which select and access an agent through a wired or wireless communication network.

The UDDI registry manages description and access information of published web services and provides clients with an interface to search them. The web service invocation planner generates an efficient invocation strategy for web services using access and cost estimation information of web services which are provided by the UDDI registry and the performance monitors in web service providers, respectively. The detailed method is described in Section 4.

The architecture of the agent and web service providers is presented in Fig. 3. Functions of the common components are as follows:

- The SOAP server handles encoding and decoding of SOAP messages sent to and received from external web services.
- The web service/agent execution server performs creation and parsing of intensional XML data exchanged with other web services. It may activate calls to external web services contained in an intensional result returned from a web service and integrate all the partial results into the final extensional or intensional result. If a web service invocation plan is provided, it can be used to determine the web service calls to be activated by the execution server.
- The performance monitor continuously checks the performance and workload of a web service provider or agent server as well as communication costs with other web services in order to estimate execution cost of web services invocation. The estimated costs are delivered to the web service invocation planner and used to generate an optimized invocation plan for web services as shown in Section 4.

The agent server also has additional components:

- The context manager personalizes the use of web services by exploiting predefined user profiles and context information.
- The web services finder uses semantic information of available web services to select and bind appropriate services satisfying user requirements dynamically when they are not statically bound or provided by clients.
- The agent manager manages system resources and the life-cycle of agent instances in the agent server.

4 Web Service Invocation Strategy

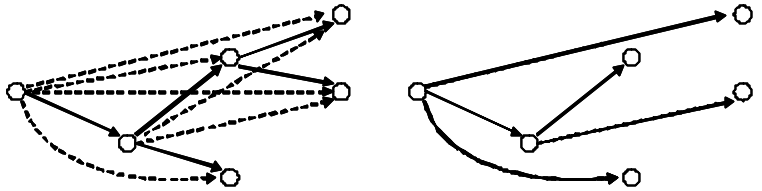
In this paper, we assume that intensional XML data can be used as input parameters and output results of the operations in web services. It means that the values of input parameters or execution results can contain calls to other web services, whose caller and invocation time are dynamically determined in run-time. To support the intensional parameters and results, the methods for type definition and type checking of the web service messages need to be extended [11].

The use of intensional data makes it possible that hierarchically interacting web services are invoked and executed in many different ways. For example, a composite web service can return to its caller an intensional result containing calls to a subset of component web services, and for each deferred invocation the caller web service can either execute it by itself or send it to its caller successively. In general, different invocation plans have different execution costs since the peer nodes that can call a web service have different performance, workload, and communication cost to the target web service. Therefore, an optimization strategy is required to generate an efficient invocation plan for involved web services.

In this section, we propose a method for generating a cost-efficient invocation plan for web services to improve the overall performance of distributed execution of web services. For global optimization of web service invocation, it is assumed that access information and invocation costs of web services are available from the UDDI registry and the performance monitor in each peer node, as described in Section 3. The optimization result is offered to all the peer nodes and used to determine whether to execute a service call or transmit intensional parameters/results to other web services.

4.1 Problem Definition

For the simplicity of description, it is assumed that there is no cycle in a sequence of service calls and there exists only one invocation path between any pair of web services. If we denote the set of finite number of vertices representing web services by V_d, the set of directed arcs connecting two web services representing a service invocation by A_d, and the weighting function for an arc in A_d which gives service invocation cost by W_d, the call relations among web services can be represented by a weighted



(a) Web services call definition tree DT and DT* (b) An invocation instance

Fig. 4. An example of call definition and invocation of web services

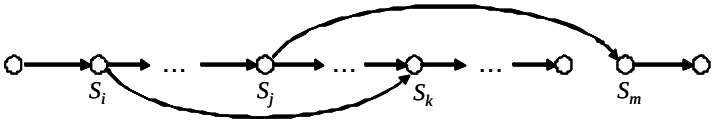


Fig. 5. An invocation plan infeasible for intensional result passing

directed tree $DT=(V_d, A_d, W_d)$ which is called *the call definition tree* for web services in this paper.

Given a call definition tree DT, a directed acyclic graph $DT^*=(V_d, A_d^*, W_d^*)$, where A_d^* is the transitive closure of A_d including weighted arcs for all the pair of vertices connected by a path in DT, represents all the possible ways of invoking web services in the intensional result passing approach. Fig. 4-(a) shows an example.

Considering intensional result passing, we have two observations related with the invocation of web services. First, if a path exists between a pair of web services v and w in DT, an execution instance is possible in which v calls w. In other words, given a path $(s_1, s_2, \ldots, s_n)$ in DT, if an intensional result containing a call to s_n is returned from s_{n-1} to s_1 directly or indirectly, web service s_1 can invoke web service s_n. Second, in a feasible execution instance for a given DT, each web service should be called from one caller web service and the result should be returned to the caller. Specifically, if we let $ET=(V_e, A_e, W_e)$ be a tree representing an invocation plan with intensional result passing, there cannot exist two different vertices s_i and s_j such that both arcs (s_i, s_k) and (s_j, s_k) are contained in A_e for all vertices s_k in V_e. These imply that an invocation plan for web services should be a spanning tree for DT^*, as shown in Fig. 4-(b). Furthermore, we have the following theorem.

Theorem 1. Assume that web services are executed with intensional results and let $P_d(v, w)= (s_1, s_2, \ldots, s_n)$ $(s_1=v, s_n=w)$ be the path from v to w $(v \neq w)$ over the call definition tree DT. In all invocation trees representing execution instances of service invocation, $ET=(V_e, A_e, W_e)$, if $(s_i, s_j) \in A_e$ then $(s_k, s_l) \notin A_e$ for some vertices k and l such that $k < i < l < j$ or $i < k < j < l$ $(1 \leq i, j, k, l \leq n)$ □

Fig. 5 shows an infeasible instance of service invocation which is described in Theorem 1. An *optimal invocation plan* for web services returning intensional results can be defined based on the above properties. Given a call definition tree $DT= (V_d, A_d, W_d)$ for a set of web services, let Gbe a set of spanning trees for DT^* which satisfy the following condition: for all vertices v and w having a path $P_d(v, w)=(s_1, s_2, \ldots, s_n)$ $(s_1=v, s_n=w)$ in DT, if $(s_i, s_j)\in A_e$ then $(s_k, s_l)\notin A_e$ such that $i < k < j < l$ or $k < i < l < j$ $(1 \le i, j, k, l \le n)$. If we define

$$W(T) = \sum_{a \in A} W(a)$$

for a weighted tree $T=(V, A, W)$, a tree T_0 such that

$$W(T_0) = \underset{T' \in \Gamma}{Min}(W(T'))$$

is called the optimal invocation plan for the given web services.

On the other hand, intensional parameters containing calls to other web services can be also utilized when data dependency exists between the input and output of a pair of web services invoked from the same web service. For example, assume that a web service A invokes other web services B and C. If the input value of C depends on the result of B, an execution instance is possible in which A invokes C with an intensional input parameter containing a call to B and C invokes B on behalf of A before it executes its own business logic. Thus, data dependencies among component web services and service calls with intensional parameters make various ways of service invocation possible. This subsequently derives the same kind of optimization problem as the one with the intensional result passing.

4.2 The Proposed Method

A naï ve solution for the optimization problem described in Section 4.1 is to search the solution space exhaustively by enumerating all the possible invocation plans and select the one with a minimum execution cost. While the method always results in the optimal solution, it requires huge amount of execution time in investigating all the possible solutions. For example, considering the vertices in the given call definition tree in the increasing order of their depth, we can generate all the invocation plans for the set of vertices whose depths are no deeper than k from the ones for the set of vertices whose depths are no deeper than k-1. This enumeration process takes exponential time in the number of web services, which means that it is not efficient enough to be applicable for a large number of web services rapidly growing over the Internet. We can use a heuristic search method such as A* algorithms [12] in order to avoid an exhaustive search by pruning an irrelevant part of the search space. However, the worst-case performance cannot be improved from the exhaustive search method.

We propose a greedy algorithm shown in Fig. 6 to produce an efficient invocation plan for the given web services in a more efficient way. To generate an invocation tree, it searches the vertices in the call definition tree in a breadth-first manner. For each vertex visited during search, it selects a vertex as its caller and inserts an arc connecting them in the current invocation tree. For a visited vertex w, only the ances-

1 **Greedy Algorithm**
2 **Input:** a call definition tree $DT = (V_d, A_d, W_d)$
3 **Output:** an invocation tree for DT
4 **begin**
5 Let Q be a queue to store nodes in DT. $Q := \emptyset$;
6 Let $ET = (V_e, A_e, W_e)$ be the result invocation tree. $ET := (\{r\}, \emptyset, \emptyset)$;
7 **while** $Q \neq \emptyset$ **do**
8 $v :=$ Dequeue(Q);
9 Let $(s_1, s_2, \ldots, s_n)$ be the sequence of nodes in $P_e(r, v)$ in ET.
10 **for** each child node w of v in DT **do**
11 Find u such that $u = \underset{u' \in P_e(r,v)}{Min} \left(W_d^*((u', w)) + \sum_{x \in Desc(w)} \underset{t \in P_e(r,u')}{Min} W_d^*((t, x)) \right)$.
12 $V_e := V_e \cup \{w\}$; $A_e := A_e \cup \{(u, w)\}$; $W_e := W_e \cup \{W_d^*((u, w))\}$;
13 Enqueue(Q, w).
14 **end for**;
15 **end while**;
16 **return** ET;
17 **end.**

Fig. 6. A greedy algorithm

tor vertices of w can be its caller, which are on the actual invocation path $P_e(r, v)$ from the root vertex r to the parent vertex v of w in the invocation tree. We select one of them by considering the costs of invoking the descendent web services of w as well as the cost of invoking w. Specifically, denoting the set of descendent vertices of w by $Desc(w)$, we select as the caller of w a web service u in $P_e(r, v)$ which makes the value of

$$W_d^*(u, w) + \sum_{x \in Desc(w)} \underset{t \in P(r,u)}{Min} W_d^*(t, x)$$

minimal, as shown in Fig. 7. This requires that for the vertices in $Desc(w)$ we should find their callers with minimum costs from a different set of ancestor vertices in $P_e(r, u)$ repetitively, which are determined by the position of the vertex u on the path $P_e(r, v)$. Therefore, the time complexity of finding the caller of w is $O(|Desc(w)| \cdot l^2)$ where l is the length of $P_e(r, v)$. The algorithm can be improved to $O(|Desc(w)| \cdot l)$ by using more memory space. Specifically, we can avoid a lot of comparisons by considering the vertex u on the path $P_e(r, v)$ in the increasing order of depth, storing the selected minimum-cost callers and their costs for the descendent vertices in $Desc(w)$ for a specific position of the vertex u, and reusing the stored information in the next step with the next position of the vertex u. As a result, if we denote the number of web services by n, the overall optimization algorithm can be executed in

$$O(\sum_{i=1}^{h} |Desc(w)| \cdot i \cdot f^i) = O(n \log^2 n).$$

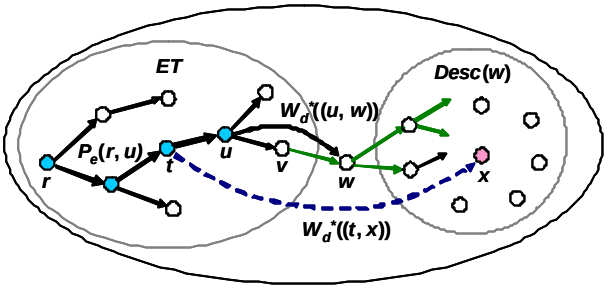


Fig. 7. Selection of the caller of *w* in the proposed greedy algorithm

5 Conclusion and Future Work

This paper addressed the problem of finding a cost-optimal invocation plan for hierarchically interacting composite web services which can exchange intensional data containing calls to other web services. Considering a large number of ways of invoking web services possible in such environment, we proposed a heuristic algorithm to generate an efficient invocation strategy for web services. It can enhance the overall performance of web services by distributing tasks involved in the activation and completion of the web services effectively over the participating peer nodes.

Future work includes performance evaluation of the proposed method and development of a distributed optimization algorithm for guaranteeing autonomy of the web services. We also plan to study on the efficient invocation strategies for web services exploiting both intensional parameters and results.

References

1. Abiteboule, S., Benjelloun, O. Milo, T., Manolescu, I., Weber, R.: Active XML: A Data-Centric Perspective on Web Services. Technical Report, No. 381. GEMO, INRIA Futurs (2004)
2. Active XML. http://activexml.net/
3. Apache Jelly: Executable XML. http://jakarta.apache.org/commons/jelly/
4. Curbera, F., et al.: The Next Step in Web Services. Communications of the ACM, 46(10) (2003) 29-34
5. Davis, D., Parashar, M.: Latency Performance of SOAP Implementations. Proc. of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2002) (2002) 407-412
6. Jennings, N., Wooldridge, M.: Software Agents, IEE Review, 42(1) (1996) 17-20
7. Kohlhoff, C., Steele, R.: Evaluating SOAP for High Performance Business Applications: Real-Time Trading Systems. Proc. of WWW'03 (2003)
8. Macromedia Coldfusion MX. http://www.macromedia.com/

9. Mani, A., Nagarajan, A.: Understanding Quality of Service for Web Services. IBM developerWorks (2002)
10. McIlraith, S. A., et al.: Semantic Web Services. IEEE Intelligent Systems, 16(2) (2001) 46-53
11. Milo, T., Abiteboul, S., Amann, B., Benjelloun, O., Dang Ngoc, F.: Exchanging Intensional XML Data. Proc. of ACM SIGMOD Conference (2003)
12. Nilsson, N. J.: Artificial Intelligence: A New Synthesis. Morgan Kaufmann Publishers, Inc., San Francisco, CA (1998)
13. Solanki, M., Abela, C.: The Landscape of Markup Languages for Web Service Composition. Technical Report, De Montfort Univ. (2003)
14. Tsalgatidou, A., Pilioura, T.: An Overview of Standards and Related Technology in Web Services. Distributed and Parallel Databases, 12(2) (2002) 135-162
15. Zeng, L., et al.: Quality Driven Web Services Composition. Proc. of Int'l Conf. on WWW (2003) 411-421