In the First International Conference on Web Services (ICWS), Las Vegas, Nevada, June 23-26, 2003.

Proteus: A System for Dynamically Composing and Intelligently Executing Web Services[1]

Shahram Ghandeharizadeh, Craig A. Knoblock, Christos Papadopoulos, Cyrus Shahabi,
Esam Alwagait, José Luis Ambite, Min Cai, Ching-Chien Chen, Parikshit Pol, Rolfe Schmidt,
Saihong Song, Snehal Thakkar, and Runfang Zhou
University of Southern California
Los Angeles, California 90089-0781

Abstract

Many organizations envision web services as an enabling component of Internet-scale computing. A final vision of web services is to realize a dynamic environment that identifies, composes and executes web services in response to a query. This vision shapes the design and implementation of Proteus. In addition to describing Proteus' novel components, this paper outlines its initial system design.

A. Introduction

A web service is either a computation or an information service with a published interface. Its essence is a remote procedure call (RPC) that consumes and processes some input data in order to produce output data. It is a concept that renders web applications extensible: By identifying each component of a web application as a web service, an organization may combine these web services with others to rapidly develop a new web application. The new web application may consist of web services that span the boundaries of several (if not many) organizations. Proteus[2] is a system to: a) dynamically compose plans that integrate web services, b) execute a plan as efficiently as possible in the presence of failures and web service migrations, and c) monitor and show the status of different components at runtime.

As a motivating example, consider the problem of identifying a building in an image. This can be done by combining web services for imagery (i.e., TerraService) with services for the property tax sites and online phone books (see the description of our previous work, Section B). One could write a program to integrate information from the appropriate web services to solve this query for a given area, but the challenge is that there are approximately a thousand property tax sites and hundreds of telephone books for the US and each of them has different levels of coverage. For example, in New York State there is one tax service, but in California there are dozens. A better alternative is for a system such as Proteus to dynamically identify, compose, and execute the appropriate web services to process a query. First, Proteus would identify the relevant web services. In our example, relevant web services would include Microsoft's TerraService for imagery, the property tax and telephone book services for the given area, and a geocoding service to convert street addresses into lat/long coordinates. Second, it identifies the most efficient plan and executes it to produce a timely response. Third, it monitors and controls the execution of a plan in support of physical-location-independence, which means the plan will execute as long as a copy of the referenced web services is available. This criterion is important because it frees the end user (and programmers) by requiring the system to

[1] This research is supported in part by an unrestricted cash gift from Microsoft Research.

[2] Proteus is a Greek sea god with the ability to change shape at will and predict the future.

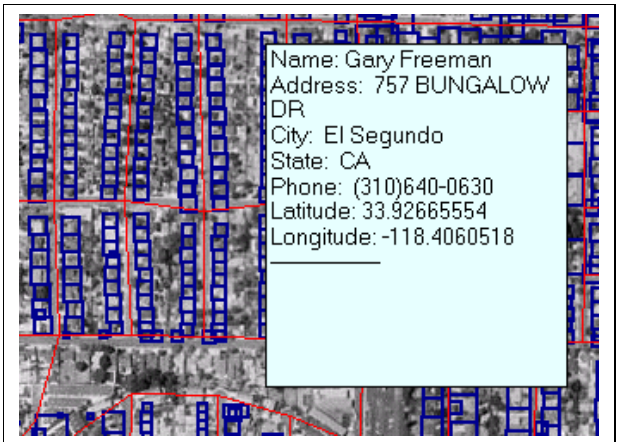
resolve the location of a web service in the presence of both a) web service migrations to balance load, and b) node failures that render a copy of a web service unavailable. In order to monitor the execution of a plan, Proteus will provide visualization tools that query the run-time components for their status.

We have investigated several key components of Proteus including converting semi-structured sources into web services, combining web services [Thakkar et al., 2002], and techniques to efficiently execute those services using compression [Cai et al. 2002, Ghandeharizadeh et al. 2002b], streaming, and a Dependable Web service (DeW) framework [Alwagait et al. 2003]. This paper provides an overview of these techniques and outlines our immediate future research directions.

B. Dynamic Composition of Web Services

This section describes: 1) techniques to convert online sources into web services using our previous work on wrapper building tools, 2) automatic techniques for composing XML web services from existing web sources. Wrapper building tools facilitate building a wrapper around a web source with minimal user assistance. A wrapper provides a uniform access method to extract and query information from a web source [Knoblock et al. 2000]. We have built a tool using the Microsoft.Net framework to convert existing wrappers to XML web services. We utilized the above-mentioned tools to compose several web services from existing web sources such as property tax web sites, white pages, Yahoo weather, etc.

We also investigated a mediator-based approach to dynamically compose new web services from existing web services. Initial work on our approach concentrated on manually modeling web services as data sources in the mediator's domain model and on generating integration plans to answer user queries by integrating information from different web services [Thakkar et al. 2002]. We demonstrated the usefulness of these techniques by building a geocoder web service to find very accurate geographic coordinates for a given address (http://apollo.isi.edu/Geocoder/Service1.asmx). Our geocoder utilizes information from the Tigerline files and various property tax web services to find more accurate geographic coordinates compared to the other geocoders available on the web.



We utilized our tools to develop the BuildingFinder application, see - http://apollo.isi.edu/BuildingFinder/WebForm1.aspx, which identifies buildings in a satellite image and finds information about the buildings such as, name of the person living in the building, the address of the building, etc. The Building Finder application superimposes the buildings and road network on the satellite imagery obtained from Microsoft TerraService web service. As shown in Figure 1, the Building Finder application identifies buildings in the satellite image with a high accuracy by combining data from various property tax web services, white page web services, and a geocoder.

C. Efficient Execution of Plans

One may improve the execution of a plan in a variety of ways ranging from minimizing the number of web services invoked by an integrated plan to identifying the fastest replica of a web service to process the plan. With XML emerging as the standard of choice for data interoperability among web services, we started our investigation by focusing on efficient transmission of XML messages. Compression techniques such as Zip and XMill reduce message size. In [Cai et al. 2002], we showed that with a message M1, XMill compressed XML-formatted M1 is at times smaller than the Zip compressed binary-formatted M1. Next, we showed the tradeoff between the overhead and benefit of using compression [Ghandeharizadeh et al. 2002b]. The overhead is the time required to a) compress a message prior to its transmission, and b) uncompress the message upon its arrival at the client. Its granularity is units of time and dependent on the processor speed of the participating machines, i.e., the machines hosting a producer and a consumer, respectively. Compression's benefit is the reduced message size that results in a shorter transmission time. This benefit can also be measured in units of time and its value depends on network characteristics such as latency, loss-rate, and bandwidth. We have developed a middleware, termed Network Adaptable Middleware (NAM) [Ghandeharizadeh et al. 2003], that decides when to compress a message with the objective to minimize the time required to deliver a message from a producer to a consumer. In other words, NAM strives to ensure the overhead of compression does not outweigh its benefits. It is designed to be general purpose and adaptable to any application. For example, it employs a Cubic regression technique that consumes the previously observed

compressed message sizes and compression times to estimate future elapsed times.

In [Ghandeharizadeh et al. 2003], we reported on the performance of NAM with a variety of message sizes and processor speeds when compared with environments that employ either 1) uncompress transmission always, representative of today's common practice, 2) Zip compressed transmission always, 3) XMill compressed transmission always. The results demonstrate the superiority of NAM. As a sample of these results, Figure 2 shows the percentage improvement obtained by NAM when compared with these alternatives for different network configurations. These configurations are representative of Internet connections that provide either a) a low, moderate, or high latency, denoted $\downarrow$L, $\leftrightarrow$L, $\uparrow$L, b) a low, moderate, or high bandwidth, denoted $\downarrow$B, $\leftrightarrow$B, $\uparrow$B, or c) a combination of these. For a low latency (7 msec), moderate bandwidth connection (90 Mbps), we used an environment consisting of a client at the University of San Diego invoking a Web Service at USC. This is termed the Southern California configuration, $SC_{\downarrow L,\leftrightarrow B}$. For a moderate latency connection (90 msec), our

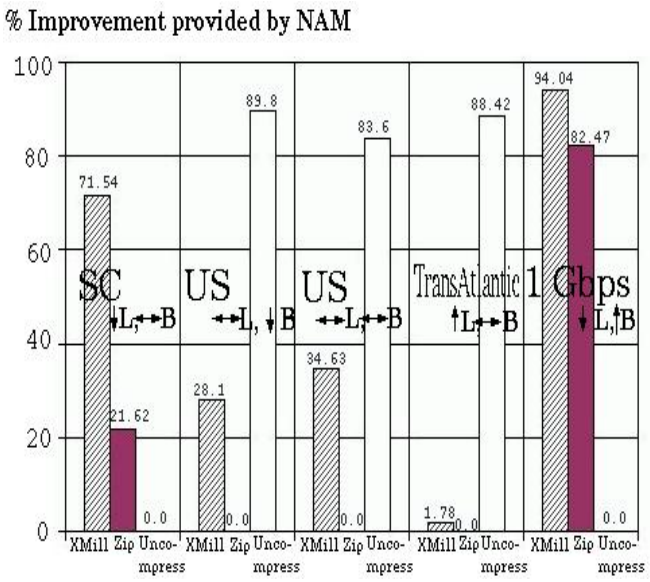


Figure 2: NAM with 1 GHz processor

experimental environment consisted of clients located on the east-coast invoking the web service at USC. We analyzed both a moderate bandwidth connection with the client located at the University of Massachussets-Amherst, and a low bandwidth connection (1 to 1.2 Mbps) with the client located at ISI-East, denoted as $US_{\leftrightarrow L,\leftrightarrow B}$ and $US_{\leftrightarrow L,\downarrow B}$, respectively. For a high latency (178 msec), moderate bandwidth connection we employed a client located at the University of Saarlandes in Germany. This configuration is termed $Trans\text{-}Atlantic_{\downarrow L,\uparrow B}$. We were pleasantly surprised to observe comparable bandwidths (~ 90 Mbps) for both Trans-Atlantic and intra-continent US connections. Finally, we analyzed an intranet configuration using the gigabit Ethernet switch. This is a low latency (0.3 msec), high bandwidth (~300 to 500 Mbps) connection. The results demonstrate that NAM employs uncompressed transmission when latency is negligible, i.e., 1Gbps and SC, to provide substantial savings when compared with a compress-always environment. With a high latency connection, NAM is flexible enough to switch to a compressed transmission to provide substantial savings when compared with an uncompressed transmission always which is representative of today's common practice.

D. Physical location independence

With a collection of nodes hosting many different web services, the overall performance is maximized when the system load is evenly distributed across the nodes. In a dynamic environment, this is accomplished by migrating web services from a busy server to an idle one [Sommers et al. 2001]. A challenge is how to ensure remote applications are not impacted by this change in the physical location of migrating web services [Ghandeharizadeh et al. 2002a]. (This issue also arises with node failures.) To address this challenge, we have developed a Dependable Web (DeW) framework that advocates the following straightforward concept: separate the functionality provided by a web service from the infrastructure that provides it with additional information to recover from an anticipated change such as web service migrations and node failures [Alwagait et al. 2003]. The infrastructure is realized using a cloud of DeW registries that listen on a known address. We represent a dynamic change as an exception. The handlers for these exceptions are stored in the DeW registries. Upon encountering an anticipated change, a client (or a participating web service) component raises a DeW exception, contacts a DeW registry for its handler, downloads the handler and executes it in order to recover. At the time of this writing, we are focused on an implementation of the DeW registries using a peer-to-peer network such as Pastry developed at Microsoft Research.

E. Future research

We are investigating the design and implementation of Proteus using a variety of frameworks: Global XML Web Service Architecture (GXA) from Microsoft [Box 2002], XL [Florescu et al. 2003], the Business Process Execution Language (BPEL4WS) [Curbera et al., 2002], etc. It will automatically integrate existing web services to construct

new services, such as the Building Finder service. Here, we describe Proteus on various specifications of GXA and our previous research on both composing web services from web sources and on efficiently and reliably executing web services.

In order to support the dynamic composition and execution of web services, we plan to investigate the following key research challenges using Proteus: (1) utilize the WS-Inspection specification to provide information that models the web services as data sources, (2) find existing web services using the WS-Inspection specification of the GXA and UDDI at run-time, (3) generate an integration plan for a new web service using a mediator and encode the integration plan using the WS-Routing specification of the GXA, and (4) execute the plan efficiently using NAM (and streaming) in a physical-location-independent manner by using DeW in combination with WS-Referral and other extensions to WSDL.

Figure 3 shows a possible architecture for implementation of Proteus. In this illustration, wrappers are used to convert various web sources into web services. WS-Inspection will be used to discover existing services and model them as data sources. When a client submits a query to Proteus, the system invokes different web services to generate a response for the query. Proteus employs the Web Service Composition to identify the relevant web services. Next, it generates a plan to query relevant data from various web services. The "Web Service Execution" engine executes this plan in a physical-location-independent manner. If it is required to establish intermediate web services that integrate data from multiple sources, it will employ NAM to facilitate efficient transmission of data.

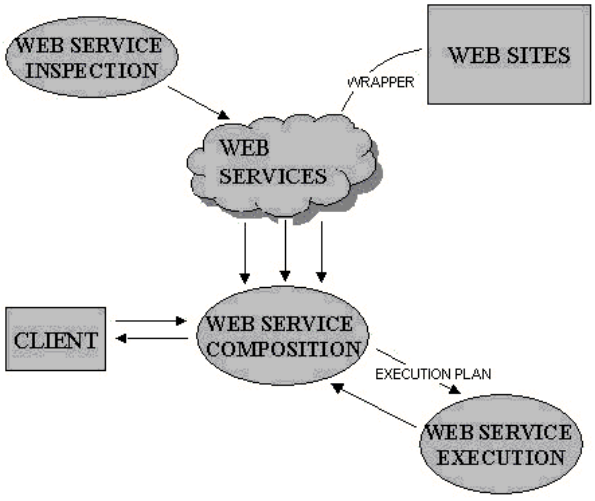


Figure 3: Overview of Proteus

We plan to develop a representation scheme to model the information and/or capabilities provided by a web service. For information services, the representation scheme will cover information such as the type of information, the attributes provided, the binding patterns required, and the coverage of the information by the web service. For services that support transactions, such as a book buying service, the representation scheme must also describe the capabilities of the service. We plan to use ontology of services for this purpose. For both information and transaction services, the representation will be stored within the WS-Inspection specification for the web service.

Using this representation scheme, we will then develop a framework that can automatically locate different web services using UDDI's Inquiry method and WS-Inspection document for the web service. This will allow a user to publish a new service that has been appropriately described and have this service immediately available to Proteus to integrate into composed services.

Finally, we will develop a mediator framework for Proteus that can utilize the WS-Inspection document of the web service to dynamically build a plan that composes the available web services to handle requests that could not be answered by an individual service. The mediator framework will utilize the data source models to generate an integration plan to answer the user query. We plan to investigate how the WS-Routing and WS-Referral specifications can be used to implement such integration plans. For example, if a user queries Proteus to retrieve geographic coordinates, address, city, and state where 'John Smith' lives, its mediator will convert that query into a WS-Routing header that visits two different web services. First, a white page web service to retrieve locations for 'John Smith'. Next, the WS-Routing header specifies that the output of this web service (locations for 'John Smith') should be forwarded to a geocoder web service that consumes the locations to produce a coordinate for each location. The ultimate receiver in this case is the client, which consumes the coordinates and retrieves the appropriate aerial maps from TerraService for display.

During the execution of a plan, one or more data sources may not comply with WS-Routing specification. One possible solution is for Proteus to setup WS-Routing intermediaries, SOAP routers, that consume the output of one web service (say the white page) and covert the resulting SOAP messages into a WS-Routing compliant one, ready for routing to the next web service. This execution employs both our compression middleware and the DeW registries. Both are a natural fit with GXA's WS-Routing and WS-Referral available from Microsoft, i.e., Web Services Development Kit (WSDK).

To evaluate the functionality of Proteus, we will start with hand-generated query plans and then later use the automatically generated plans. We will simulate run-time errors, e.g., 811 Service Unavailable, 811 Service Too Busy, etc., to analyze interactions with DeW. In particular, we will investigate how DeW may generate WS-Referral statements to populate a SOAP router once it is informed of a dynamic change, e.g., migration, failure, etc. (Note that the environment may either use a global DeW registry or DeW registries provided by a service provider; currently, we envision the use of WS-Routing to support this flexibility.). We will analyze streaming to further enhance response time.

In order to facilitate debugging of Proteus, we will develop an interface that visualizes the run-time system and how a plan is processed. This user interface complements the GXA framework to empower a programmer to: a) monitor the behavior of Proteus, and b) examine the correctness of the run-time environment when processing a plan. Obviously, a run-time component must provide a mechanism (either a pull, i.e., query, or push) that reports its status to the interface. While it is not realistic to assume the existence of this interface for all participating components, we can provide them for our components, e.g., DeW registries, permanent and dynamic SOAP routers established by a plan, the services we create using wrappers, etc. To maintain a modular design, these components generate XML formatted messages that describe their input, output, and internal status, e.g., with SOAP a router, it might output its WS-Referral statements.

An exciting, and significant contribution of Proteus is its demonstration that the dynamic and efficient composition of web services is both feasible and useful. In our vision, end users would describe and publish their web services. Proteus discovers and uses those descriptions to respond to user requests by automatically generating integration plans and then efficiently and reliably executing them. The system is fully dynamic because as soon as a new service is published, it can be used to process user requests (in the context of GXA, this is accomplished using WS-Referral and WS-Routing).

Bibliography

[Alwgait et al. 2003] E. Alwagait and S. Ghandeharizadeh 2003. DeW: A Dependable Web Services Framework. Submitted for publication.

[Arens et al. 1996] Y. Arens, C. A. Knoblock and W. Shen 1996. Query Reformulation for Dynamic Information Integration. Intelligent Integration of Information.

[Box 2002] D. Box. Understanding GXA. Microsoft Corporation. URL: http://msdn.microsoft.com/webservices /understanding/gxa/default.aspx. July 2002.

[Cai et al. 2002] M. Cai, S. Ghandeharizadeh, R. Schmidt, and S. Song. A Comparison of Alternative Encoding Mechanisms for Web Services. In 13th International Conference on Database and Expert Systems Applications (DEXA), Aix en Provence, France, September 2002.

[Curbera et al., 2002]. F. Curbera, Y. Goland, J. Klein, F. Leymann, S. Thatte, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.0, 31 July 2002.

[Ghandeharizadeh et al. 2002a] S. Ghandeharizadeh, F. Sommers, J. Kuntal, E. Alwagait. A Document as a Web Service: Two Complementary Frameworks. In the Second International Workshop on Multimedia Data Document Engineering (held in conjunction with EDBT), Prague, Czech Republic, March 2002.

[Ghandeharizadeh et al. 2002b] S. Ghandeharizadeh, C. Papadopoulos, M. Cai, and K. Chintalapudi. Performance of Networked XML-Driven Cooperative Applications. In Proceedings of the Second International Workshop on Cooperative Internet Computing (CIC, held in conjunction with VLDB), Hong Kong, China, August 2002.

[Ghandeharizadeh et al. 2003] S. Ghandeharizadeh, C. Papadopoulos, P. Pol, and R. Zhou. NAM: A Network Adaptable Middleware to Enhance Response Time of Web Services. Submitted for publication.

[Knoblock et al. 2000] C.A. Knoblock, K. Lerman, S. Minton, and I. Muslea. Accurately and Reliably Extracting Information from the Web: A Machine Learning Approach. IEEE Data Engineering Bulletin, 23 (4), December 2000.

[Knoblock et al. 2001] C.A. Knoblock, S. Minton, J.L. Ambite, N. Ashish, I. Muslea, A. Philpot and S. Tejada 2001. The Ariadne Approach to Web-Based Information Integration. International Journal on Intelligent Cooperative Information Systems (IJCIS) 10(1-2): 145-169.

[Sommers et al. 2001] F. Sommers, S. Ghandeharizadeh, and S. Gao. Cluster-Based Computing with Active, Persistent Objects on the Web. In IEEE International Conference on Cluster Computing, October 2001.

[Thakkar et al. 2002] S. Thakkar, C. A. Knoblock, J. Ambite and C. Shahabi. Dynamically Composing Web Services from on-Line Sources. In Proceeding of 2002 AAAI Workshop on Intelligent Service Integration, Edmonton, Alberta, Canada.

[Florescu et al. 2003] D. Florescu, A. Gruenhagen, and D. Kossmann. XL: A Platform for Web Services. *First Biennial Conference on Innovative Data Systems Research*, Asilomar, CA, January 2003.