ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology
Volume 1, Issue 4, June 2012

Promulgate: an approach to Optimize the data transfer in Service Oriented Architecture

B. Chandra Mouli
M. Tech (Software Engineering),
Audisankara College of Engineering & Technology,
Gudur, Andhrapradesh, India,
bathalamouli@gmail.com

Prof. C. Rajendra,
Head, Department of CSE,
Audisankara College of Engineering & Technology,
Gudur, Andhrapradesh, India.

***Abstract*— The main potential benefit of Service-oriented architecture (SOA) is applying across multiple solution environments based on the request and reply paradigm. Service-oriented architecture integrates both enterprise and application architectures. When the services and size of the workflow increases, the SOA orchestration reaches the scalability limits and same effected on the data transmission between services, a standard orchestration desires to transport all the data through a workflow engine when communicate on the web service without any third party data transfer, which results redundant data transfer and engine to become a bottleneck to execution of a workflow.**

As a solution, we present Promulgate, an alternative service-oriented architecture. Promulgate proposed an orchestration model as central control in grouping with a choreography model of optimized distributed data transport. We are using Data caching techniques by using dynamic proxy deployment and SOAP compression techniques by using zipped SOAP for this hybrid architecture to optimize the data transfer in SOA.

Keywords-SOA; Promulgate; data transfer; data caching; SOAP- compression; ZippedSOAP.

I. INTRODUCTION

Service-oriented architecture (SOA) is the advancement of distributed computing based on the request and reply paradigm. In the early days, distributed applications communicated using proprietary protocols, and system administrators used ad hoc methods to manage systems that might be across town, on another continent, or anywhere in between. Many standards have been developed over the years to reduce the costs of deployment and maintenance, with varying degrees of success. Today, the key technologies in distributed systems are service-oriented architecture (SOA). [1]

In considering the term *service-oriented architecture*, it is useful to review the key terms. *Architecture* is the structure of a system, defining its functions, externally visible properties, and interfaces and internal components and their relationships, along with the principles governing its design, operation, and evolution [2]. A *service* is a software component that can be access via a network to provide functionality to a service client and used in a technology neutral standard form [3]. The term *service-oriented architecture* is a pattern of constructing distributed systems that transport functionality as *services*, with the loosely coupling and interacting services. In older environments, the building of the solution was so straight forward that the task of abstracting and defining its architecture was hardly ever performed. With the rise of multi-tier applications, the variations with which applications could be delivered began to dramatically increase. IT departments started to recognize the need for a standardized definition of a baseline application that could act as a template for all others. This definition was abstract in nature, but specifically explained the technology, boundaries, rules, limitations, and design characteristics that apply to all solutions based on this template. This was the birth of the application architecture.

Application architecture is a blueprint for the application development team; different organizations may construct different applications architectures. Some may document it as high level providing abstract physical and logical representations of the technical blueprint, and others include data models, communication flow diagrams, application-wide security requirements, and aspects of infrastructure. It is not unusual for an organization to have numerous application architectures. A single architecture document typically represents a distinct solution environment. For example, an organization that houses both .NET and J2EE solutions would very likely have separate application architecture specifications for each. It is for this reason that when multiple application architectures exist within an organization, they are almost always accompanied by and kept in alignment with governing enterprise architecture.

In larger IT organizations, the need to control and direct IT infrastructure is critical, when various, disparate application architectures co-exist and sometimes even integrate, the demands on the underlying hosting platforms can be complex and onerous. Therefore, it is necessary for create a master specification, providing a high-level overview of all forms of heterogeneity that exist within an enterprise, as well as a definition of the supporting infrastructure. An *enterprise architecture* specification is to an organization what an urban plan is to a city. Typically, changes to enterprise architectures directly affect application architectures, which is why architecture specifications often are maintained by the same group of individuals. Further, enterprise architectures often contain a long-term vision of how the organization plans to evolve its technology and environments.

Service-oriented architecture integrate both enterprise and application architectures. The main potential benefit of SOA is applying across multiple solution environments. This is where the investment in building reusable and interoperable services based on a vendor-neutral communications platform can fully be leveraged. Both the architectures are integrated by placing the service layer in between them. The service layer acts as an interface between the both the architectures.

Service is a functionality offered to satisfy the consumer's need according to a negotiated contract which includes Service Agreement. *Web Service* is a piece of code that can be retrieving via a network to provide services as functionality to a service client without knowing the code behind it. Web service is also defied as a software system designed to support interoperable machine to machine interaction over a network. A Web service provides as "a standardized way of integrating Web-based applications using the XML, SOAP, WSDL, and UDDI open standards over an Internet protocol backbone. The XML, SOAP, UDDI are the available open standards, each have their own uses, XML is for to tagging the data, SOAP is for transferring the data, WSDL is for describing the services available in the service describing, and UDDI is for listing what services are available in the universal description.

Work flow is the automation of business processes, in whole or in part, during which documents, information or tasks are passed from one participant to another for action according to a set of procedural rules.

Orchestration is a centrally controlled set of workflow logic facilitates interoperability between two or more different applications. With orchestration, different processes can be connected without having to redevelop the solutions that originally automated the processes individually. Orchestration bridges this gap by introducing new workflow logic. Further, the use of orchestration can significantly reduce the complexity of solution environments. Workflow logic is abstracted and more easily maintained than when embedded within individual solution components.

The requirement for organizations to interoperate via services is becoming increasingly real and increasingly complex. This is especially true when interoperation requirements extend into the realm of collaboration, where multiple services from different organizations need to work together to achieve a common goal. The *Web Services Choreography Description Language* (WS-CDL) is one of several specifications that attempts to organize information exchange between multiple organizations (or even multiple applications within organizations), with an emphasis on public collaboration.

By adopting a choreography model the output of a service incantation can be conceded straight to where it is required, as input to the next service in the workflow, not through a centralized workflow engine as in the case with orchestration. Decentralized control brings a new set of problems, which are the result of message passing between asynchronous distributed and concurrent processes. In addition, current choreography models are persistent, in that each individual service needs to be re-engineered in order to take part in choreography.

When the services and size of the workflow increases, the SOA orchestration reaches the scalability limits and same effected on the data transmission between services, a standard orchestration desires to transport all the data through a workflow engine when communicate on the web service without any third party data transfer, which results redundant data transfer and engine to become a bottleneck to execution of a workflow.

This paper presents and evaluates the Promulgate architecture, which sits in between pure orchestration (completely centralized) and pure choreography (completely decentralized). This centralized control flow; distributed data flow model maintains the sturdiness and plainness of centralized orchestration but services choreography by allowing services to transfer data between themselves, without the complications associated with modeling and deploying service choreographies. The Promulgate architecture reduces data transfer between services (which don't contain functionality for third-party data transfer), which in turn speeds up the execution time of workflows and removes the bottlenecks associated with centralized orchestration.

The promulgate approach concentrates on the two main issues, one is the data caching using dynamic proxy deployment to avoid the load balancing depending on the network traffic, and another one is zippedSOAP to reduce the quantity of the data transfer between the web services.

II. The Promulgate Architecture

This Section describes the Promulgate architecture's hybrid model, Web services implementation,

A. *Promulgate Actors*

The Promulgate architecture is a hybrid between orchestration and choreography techniques. In order to provide Web services with the required functionality proxies are introduced. Proxies are less disturbing than previous choreography models as individual services do not have to be reconfigured in order to take part in a workflow.

Proxies are proscribed during a centralized workflow engine, running a chance workflow language; allowing standards-based approaches and tooling to be utilized. Proxies switch references to data with the workflow engine and overtake actual data directly to where they are required for the next service incantation. This allows the workflow engine to monitor the progress and make changes to the execution of a workflow. To utilize a proxy, a Web service must first be registered. The Promulgate actors and interactions are illustrated by figure 1 and described below:

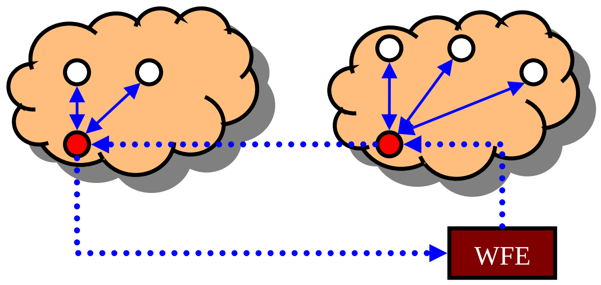

Figure 1 Promulgate architecture, web service are represented by hollow circles, proxies by red solid circles, dashed lines WAN hops, solid lines are LAN hops.

- *Engine to proxy (WFE→P) interaction*: In the standard orchestration model, the workflow engine interacts directly with all Web services, which for

543

the remainder of the paper the author denote the engine to Web service interaction (WFE→S). Using the *Promulgate* architecture, the workflow engine remains the centralized orchestrator for the workflow, however the task of service invocation is delegated to a proxy (WFE→P).

- *Proxy-Web service (P→S) interaction*: Proxies neither create Web service requests, nor do they utilize their responses. Proxies raise services on behalf of a workflow engine, store the result (if required) and return a reference to the workflow engine.
- *Proxy to proxy (P→P) interaction*: Proxies elevate Web services on behalf of a workflow engine, instead of transfer the results of a service incantation back to the workflow engine; they can be stored (if required) within the proxy. In order for a workflow to improvement, i.e., the output of a service incantation is required as input to another service incantation, proxies pass data between themselves, moving it closer to the source of the next Web service incantation.

Proxies are preferably deployed as near as possible to registered Web services; by near we mean in terms of network distance, so that the communication slide between a proxy and a Web service is minimized.

B. *Web Services Implementation*

WS-Promulgate are implemented using a combination of .Net and the IIS and Apache Axis Web services toolkit [4]. Proxies are simple to deploy and can be configured remotely, no specialized programming needs to take place in order to develop their functionality. The only changes essential are to the workflow condition itself which invokes methods on the proxy rather than the services straight. WS-Promulgate are multithreaded and allow several applications to invoke methods concurrently. Outcome from Web service invocations are stored at a proxy by labeling them with a Universally Unique Identifier and writing them to disk. There is an assumption that there is sufficient disk space and that storage is temporary, which the proxy can clean up/delete afterwards.

III. SOAP COMPRESSION

SOAP is a simple XML-based protocol to let applications exchange information over HTTP, Or more simply: SOAP is a protocol for accessing a Web Service. It is important for application development to allow Internet communication between programs. A better way to communicate between applications is over HTTP, because HTTP is supported by all Internet browsers and servers. SOAP was created to accomplish this. SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages.

A. *ZippedSOAP*

The entire SOAP transmissions are depends on the serialization to de-serialization, and de-serialization to serialization. Where the *ZippedSOAP* squash the SOAP message before the SOAP request transmits via internet and SOAP response transmits via internet. The zippedSOAP structure is described in figure 4. The diagram shows when the compression will take place [5].

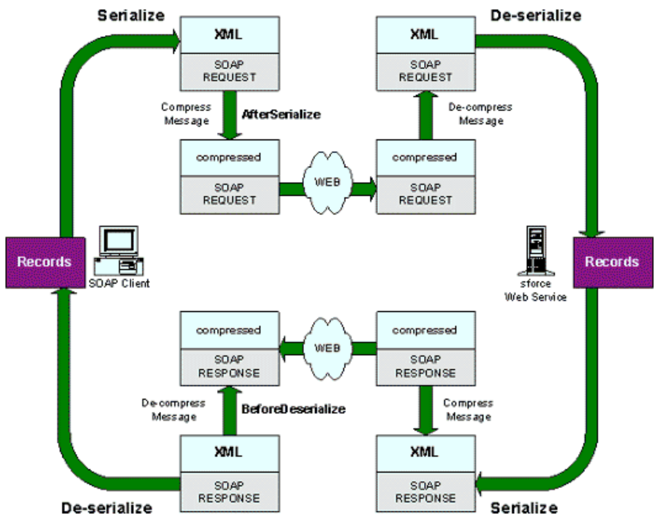

Figure 2 Zipped Soap process

IV. EXPERIMENTAL SETUP

A number of performance analysis experiments have been devised in order to observe and analyze the behavior of various workflow patterns and variables to compare the Promulgate architecture with that of a standard centralized orchestration model. The following sections describe the structure of the resulting performance analysis experiments.

A. *Node Size and Network Configurations*

In our experimental set up, data flows with no data transformations (i.e., the output of one method invocation is the input to another). We use nonblocking asynchronous communication between the Web services, even though data forwarding between proxies occurs after all Web services have refined execution. For each of the 3 workflow patterns: sequence, fan-in and fan-out, the time taken for the pattern to complete using centralized orchestration and using the Promulgate architecture is recorded (in milliseconds) as the size of the input data (in Megabytes) is increased. This basic set up is then run incrementally over a number of workflow node sizes: 4, 8 and 16 nodes in order to discover the affects of scalability. Every node size (i.e., 4, 8 or 16) is then sprint over unusual network configurations to explore how network configuration affects the performance of a workflow.

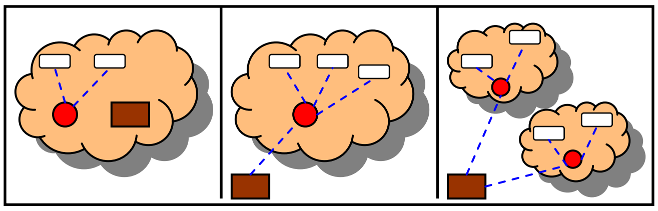

Figure 3 ***Configurations*** **from left to right: local LAN, remote LAN, internet scale.**

We have selected the following network configuration (shown in Figure 5) which imitate common scenarios when composing sets of physically distributed services:

LAN experiments: Stirring the Web services to a comparatively uniform network topology in terms of speed, e.g., a LAN allows for a basic analysis of the two models. In a LAN, it is expected that the cost of the communication links WFE→S, P→S and P→P have little variance in terms of bandwidth. In this case the performance assistance with respect to the different workflow patterns may be showing more readily. We explore this with a LAN configuration with a local workflow engine i.e., same LAN and a LAN with a remote workflow engine i.e., connected through a WAN.

V. LAN Configuration

A pool of computers from the Audisankara College of Engineering and Technology (ACET) LAN were selected as workflow nodes for these configurations. These machines are all located in the same building and are connected to the network via a 100 Mbit network connection. All these machines share the same hardware/operating environment: Intel Core 2 Duo with 2GB RAM running Windows XP. In addition to these coordinated machines, related servers (detach machines) were selected to act as the workflow engine.

A. LAN - Remote Orchestration

By moving the workflow engine exterior of the LAN configuration, we can discover workflow engine to Web service (WFE→S) communication. This reflects a very common real world scenario where a specific organization provides the whole Web services used to create a workflow, but the services are being orchestrated remotely. In particular this is applicable to Cloud Computing scenarios where services are typically hosted on groups of co-located machines and where the end user requiring these services is remote.

B. LAN - Local Orchestration

In order to discover the limits of our Promulgate approach, the workflow engine is deployed on a computer that is also connected to the proxies and Web services via a network switch. This is a proper tentative setup to test the supposition of communication tie equality but does not essentially imitate frequent deployed patterns of Web services; in general the workflow engine is remote.

In a LAN environment we make the assumption that the cost of communication is relatively uniform, therefore the cost of WFE→S, WFE→P, P→S and P→P can be considered approximately equal. The Promulgate architecture introduces extra communication links (between P→S) across this uniform network topology, which in turn degrades the execution time of a workflow when compared to standard orchestration.

VI. Related work

This Section discusses all related work from the literature, spanning pure choreography languages, enhancements to widely used modeling techniques, i.e., BPMN, decentralized orchestration, data flow optimization architectures and Grid toolkits.

A. Techniques in Data Transfer Optimisation

There are a limited number of research papers that have identified the problem of a centralized approach to service orchestration.

The *Flow-based Infrastructure for Composing autonomous Services* or FICAS [6] is distributed data-flow architecture for composing software services. Composition of the services in the FICAS architecture is specified using the Compositional Language for Autonomous Services (CLAS), FICAS is intrusive to the application code as each application that is to be deployed needs to be wrapped with a FICAS interface. In contrast, our proxy model is stretchier as the services themselves entail no variation and do not even need to know that they are interacting with a proxy. Additionally our Promulgate proxy approach introduces the concept of transient references to data around and deals with modern workflow standards.

In [8] a similar untainted choreography approach is also proposed. Authors introduce a style for transforming the orchestration logic in BPEL into a set of individual actions that co-ordinate themselves by transecting tokens over shared, distributed tuple spaces. The model suitable for execution is called Executable Workflow Networks (EWFN), a Petri nets dialect. This approach utilizes a untainted choreography model which has many additional modeling and performing problems associated with it, due primarily to the complexity of message passing between distributed, concurrent processes.

In [9] *Data caching techniques* in Grid workflows are proposed. This design caches virtual data of previous queries, so any overlapping queries and processing do not have to be repeated. The Promulgate architecture stores data at a proxy so that it can be transferred directly to the next stage of the workflow, avoiding costly network hops back to the workflow engine. It is then up to the user to clean up stored data afterwards, the automated data caching techniques proposed could be applied to the Promulgate proxies to further enhance performance.

B. Third-party Data Transfers

This paper focuses primarily on optimizing service oriented data transmissions and workflows, where services are: not equipped to handle third-party transfers, owned and maintained by different organizations, and cannot be altered in anyway prior to enactment.

Directed Acyclic Graph Manager (DAGMan) [10] submits jobs represented as a DAG to a Condor pool of resources. DAGMan removes the workflow bottleneck as data are transferred straight among vertices in a DAG however focuses solely on Condor bases Grid jobs and does not tackle the bottleneck problems related with orchestrating service-oriented workflows.

Triana[11] is an open-source problem solving environment. It is designed to define process, analyze, manage, execute and monitor workflows. Triana can distribute sections of a workflow to remote machines through a connected peer-to-peer network.

OGSA-DAI [12] is a middleware product that supports the exposure of data resources on to Grids. This middleware facilitates data streaming between home OGSA-DAI instances. Our Promulgate model could be implemented on this stand to take advantages of its streaming model.

Grid Services Flow Language (GSFL) [13] addresses some of the issues discussed in this paper in the context of Grid services; in particular services adopt a peer-to peer data flow model. However, individual services have to be altered prior to enactment, which is an invasive and custom solution, something that is avoided in the Promulgate architecture.

VII. CONCLUSSIONS

When the services and size of the workflow increases, the SOA orchestration reaches the scalability limits and same effected on the data transmission between services, a standard orchestration desires to transport all the data through a workflow engine when communicate on the web service without any third party data transfer, which results redundant data transfer and engine to become a bottleneck to execution of a workflow. Decentralized choreography models, even though optimal in requisites of data transfer are far more composite to build due to message passing between distributed, synchronized process and in practice and rarely deployed.

This paper has presented the Promulgate architecture; a centralized orchestration model of control with a peer to-peer choreography model of data transfer. This additional functionality is achieved through the deployment of a dynamic proxy and the zipped SOAP techniques that provides a gateway and standard API to Web service invocation. Prominently, proxies can be installed without unsettling current services and with least changes in the workflows that make use of them. This litheness allows a regular change of infrastructures, where one could concentrate first on improving data transfers between services that handle large amounts data.

Future work includes the following challenges:

Data transformation: Shimming is the process of changing the frequent workflow task to the output data from one service into a slightly diverse layout to use as input into another service. This will affect the performance of a workflow further as it introduces an extra service into the workflow chain. Our future work includes developing a shim loader component, where custom shims can be uploaded to the proxy, allowing shims to be performed locally at a proxy, avoiding the introduction of further network hops.

REFERENCES

[1] An Overview of Service-Oriented Architecture, Web Service and Grid Computing by Latha Srinivasan and Jem Treadwll HP Software Global Business Unit (Nov, 2005).

[2] Software Architecture in Practice, Second Edition by Len Bass, Paul Clemgents, Rick Kazman.

[3] Service Oriented Architecture: Concepts,, Technology, and Design by Thomas Erl

[4] Apache Axis. http://ws.apache.org/axis [24/02/11].

[5] SOAPCompression, http://wiki.developerforce.com/page/SOAP_Compression

[6] D. Liu, K. H. Law, and G. Wiederhold. Data-flow Distribution in FICAS Service Composition Infrastructure. In Proceedings of the 15th International Conference on Parallel and Distributed Computing Systems, 2002.

[7] W. Binder, I. Constantinescu, and B. Faltings. Decentralized Ochestration of Composite Web Services. In Proccedings of the International Conference on Web Services, ICWS'06, pages 869–876. IEEE Computer Society, 2006.

[8] D. Martin, D. Wutke, and F. Leymann. A Novel Approach to Decentralized Workflow Enactment. EDOC '08. 12th International IEEE Conference on Enterprise Distributed Object Computing, Pages 127–136, 2008.

[9] D. Chiu and G. Agrawal. Hierarchical caches for grid workflows. In CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pages 228–235, Washington, DC, USA, 2009. IEEE Computer Society.

[10] Condor Team. www.cs.wisc.edu/condor/dagman [07/06/10].

[11] I. Taylor, M. Shields, I. Wang, and R. Philp. Distributed P2P Computing within Triana: A Galaxy Visualization Test Case. In 17th International Parallel and Distributed Processing Symposium (IPDPS 2003), pages 16–27. IEEE Computer Society, 2003.

[12] K. Karasavvas, M. Antonioletti, M. Atkinson, N. C. Hong, T. Sugden, A. Hume, M. Jackson, A. Krause, and C. Palansuriya. Introduction to OGSA-DAI Services. In LNCS, volume 3458, pages 1–12, 2005

[13] S. Krishnan, P. Wagstrom, and G. von Laszewski. GSFL: A Workflow Framework for Grid Services. Technical report, Argonne National Argonne National Laboratory, 2002.