ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 2, February 2014

Pattern of Algorithm for Web Services using Distributed Application

Prof.Prashant G. Nandanwar[1], Prof.Sushma G. Patle[2]

Department of MCA, Rajiv Gandhi College of Engineering &Research, Nagpur. Maharashtra, India[1,2]

Abstract: Web services technology is all about distributed computing. There is no fundamentally new basic concept behind this and related technologies. What is really new is the reach of Web services and its ubiquitous support by literally all major vendors. Most likely, heterogeneity will at the end no longer be an obstruction for distributed applications. This will have impact on application architectures, middleware, as well as the way in which people will think about computing and businesses use computing resources. We sketch these impacts as well as some exemplary research work to be done to actually build the outline environment.

Keyword: web service, Technology,

I. INTRODUCTION

Web service is a virtual component that can be accessed via multiple formats and protocols. Such a component can be located anywhere in the network, e.g. on a machine on a different continent or within a thread in the same operating system process. Consequently, the environment for Web services is heterogeneous and distributed from the outset. Furthermore, Web services support a service-oriented architecture in which requestors can discover Web services and dynamically bind to them. But the primary focus of Web service technology is communication between Web services themselves, i.e. requestors are again Web services. Thus, to make the corresponding heterogeneous, distributed, and dynamic discovery-based environment work in practice, interoperability is key and standards are a must. A whole stack of standards has already been proposed (e.g. WSDL, SOAP, UDDI and WS-Security) and others will follow (see for example the roadmaps). Based on these standards a set of interoperability profiles will be published that describe artifacts from collections of Web services standards and its recommended collective usage to ensure interoperability across platforms and languages. We describe the overall Web service environment and underlying basic concepts.

Grid technology is about to evolve towards a "virtualization layer" for hosting Web services Corresponding environments are under implementation, for example for Java . This will enable what has been called recently "utility computing" or "on demand computing". Section 3 sketches this development. Applications in this environment will consist of two parts, namely collections of individual and autonomic Web services (i.e. components) and aggregation specifications defined as business processes. This will make the two-level programming model pervasive and will even allow involving human beings in applications. The corresponding application structure is outlined in section finally; Web services also need to be aggregated in a less structured manner: Corresponding aggregation models for Web services appear that allow building unstructured collections of Web services. Section 5 sketches the basics. We conclude in chapter 6 and present the draft of a high-level middleware stack that supports the execution of this kind of applications.

II. VIRTUAL COMPONENT

Web service technology makes functions available independent of many aspects of the proper implementation of the Web Service: A requestor has no need to know the programming model chosen to implement a Web service, i.e. whether the Web service is implemented in procedural or object oriented manner, for example. The programming language used to implement a Web service is completely irrelevant for a requestor. It doesn't matter whether the Web service is based on functions of a monolithic application system or whether it is build as a component, and if it is a component what the underlying component model is (e.g. J2EE, .NET). Any specific formats and protocols assumed by the Web service for direct communication is irrelevant for a requestor, i.e. it is hidden whether the implementation of the Web service expects ASCII files or Java objects, or whether it is invoked via a local call, an RPC or via a message queue, for example.

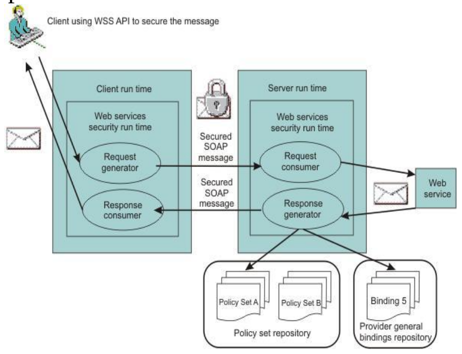

Fig- Web service as virtual component

The concept of a WSDL port type is used to define what functions a Web service pro- vides, i.e. a port type specifies the interface of a Web service. Different WSDL bindings can be used to specify how these functions can be accessed via different formats and protocols, e.g. via SOAP over JMS, or via Java objects via method call. And a WSDL port defines an actual endpoint where these functions can be accessed according to a certain format and protocol, e.g. a queue name, or a class name and JNDI name. In this sense, a Web service is a virtual component that can be implemented in many different ways, e.g. by real components or by any other piece of executable code. Especially, a Web service is not at all coupled with any kind of Web technology; be- cause of this we will often simply use the term service instead of the Web service and we will use both terms interchangeably.

2.1 Life Cycle

A service can be state full or stateless. For our discussion it is not important whether state is introduced via persistent instances or via session-like interactions. It is more important for our discussion whether or not the fact that a service is state full or not is hidden from or visible to its clients: This has impact on the client programming model, i.e. whether a client has to explicitly manage the lifecycle of a service or not. When services are dynamically discovered, having to distinguish between state full and stateless services causes complexity. Today, as a matter of fact, different application areas follow one approach or the other: In an OGSA Grid environment state full services are explicitly dealt with, while a BPEL business process environment implicitly manages the state fullness of a service on behalf of a client.

At the level of details sufficient for us, OGSA uses an explicit factory-based approach to deal with the lifecycle of a Web service: A client uses a factory to create “an instance” of a particular kind of service. The client can then explicitly manage the destruction of such an instance, or it can be left to the Grid environment. In the latter case, a client registers its interest in the instance for a particular period of time (which can be extended). When no client is any longer interested in a given instance it can be destructed.

2.2. Polices

Services need to describe their capabilities and requirements to their environment and potential users. A collection of capabilities and requirements is referred to as a policy. A policy may express such diverse characteristics as transactionality, security, response time, pricing, etc. For example, a policy of a service may specify that all inter- actions must be invoked under transaction protection, that incoming messages have to be encrypted, that outgoing messages will be signed, that responses may only be accepted within 5 seconds, and that certain operations are subject to a fee to be paid via credit card by the invoker. Since policies might get quite complex they should be reusable. For this purpose, a policy can be specified as a separate document. Such a document can be associated with (constituents of) a Web service via an attachment. Basically, an attachment consists of both, a policy and a subject the policy applies to (“resource”). Such subjects include port types, operations, messages, and also endpoints, i.e. individual ports or Web ser- vices, respectively. Attachments can be specified as follows (see Figure 3):

i. Policies can be referenced out of the WSDL definitions of subjects. This method is suited to attach policies at the time when Web service resources are specified.

ii. Web services resources that are already deployed can be associated with policies by simply pointing to these resources and to the policies to be applied. Pointing to resources can be done based on domain expressions that describe the subjects and that have to be resolved in order to find the resources characterized by the policies. This method is especially suited to attach policies to existing resources.

iii. Finally, a policy can be registered itself in UDDI (as t Models). It can be associated with a UDDI business service (as key in a category bag).

2.3 Services Bus

Web service technology enables a new kind of architecture for composing applications referred to as service oriented architecture (SOA). In SOA, services are registered in a service directory (e.g. in UDDI). Requestors find services they are interested in by enquiring service directories. The information they retrieve from a directory suffices to bind to a service and use it .When a service provider publishes a service in a service directory he specifies technical information about the service as well as business relevant information. Technical information about a service includes its interfaces, supported bindings, and endpoint information (e.g. the corresponding WSDL definitions). Business relevant information about a service falls into two categories: One category contains information about the suitability of a service from a functional perspective; the other category contains information about the suitability of a service from an operational perspective. The first category helps to understand whether a service is instrumental in achieving a business goal, e.g. buying a certain kind of sheet metal that is available within a certain period of time at a given price. Information provided are semantic descriptions about the kind of service facilitated by each of its interfaces, information about the service provider itself etc. The second category helps to understand whether a service satisfies the business policies of the requestor, e.g. all data are exchanged in an encrypted manner and are deleted once the trade is settled, messages are exchanged via reliable protocols, and payment is can be done once a month collectively for all orders. Information provided in this category includes payment methods, charging models, quality of services supported.

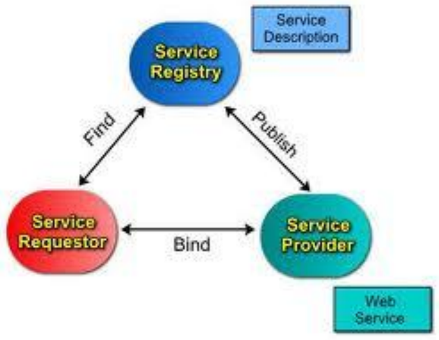

Fig- Service Bus

This infrastructure is called service bus. The service bus receives the request and peels off the declarative description of the service required .The description contains both, the business goals as well as the business policies of the requestor, and this description is used to derive the set of matching services offered by various service providers SP^j. From a requestor's perspective, all qualified services are equivalent; i.e. the set of qualified services represents the virtual service described by the requestor by his request. If more than one service has qualified the service bus will decide on one of them; this decision will be based on overall environmental properties like actual workload at the service provider side, average response time etc (e.g. measured or based on service level agreements with the service providers). Finally the service bus will bind to the service selected, pass the request message proper to it, and deliver the response to the requestor. Note that during step the invocation component sketched in section involved.

2.4 Clarification

It should be clear until now that the sometimes-heard belief, Web service technology is all about SOAP, is erroneous. As shown above, Web Service technology is about SOA, a certain architectural style, which is far more than just SOAP: SOAP is primarily one particular wire-format used to exchange data as well as a set of conventions about how to appropriately process SOAP messages. The acronyms are close, but the goals are at different scale.

III. VIRTUAL OPERATIONAL ENVIRONMENT

The service bus introduced above virtualizes services: As long as a service qualifies under a request the service bus has the liberty to target the request to it. In doing so, the service bus can optimize the execution of a single request having the optimal exploitation of the overall environment in mind. It will use algorithms and mechanisms from scheduling, workload management etc that apply to the heterogeneous and distributed environment of Web services.

1.1 Grid Services

Middleware for scientific computing with similar goals has already been developed in the Grid computing area. It thus seems only natural to bring the area of Grid computing and Web services together: outlines architecture for such a combined environment called Open Grid Services Architecture (OGSA). The most fundamental aspects of the special kind of Web services, called Grid Services that are hosted in such a combined environment are under specification.

In order to become a Grid services, a Web service has to support a set of pre-defined interfaces and has to comply with some conventions. The interfaces to be supported facilitate the discovery, creation, and lifetime management of services; they further facilitate a notification mechanism to especially enable the manageability of services. The conventions deal primarily with naming services. Based on these interfaces and conventions a standard semantics for interacting with a Grid service is defined: How services are created, how their lifetime is determined, how to invoke functions of a service etc.

3.2 Grid Services Stack

Based on depicts the stack building the overall environment for applications of Grid services. At the bottom, it shows a Grid service container based on an environment like an application server; the container provides the functions discussed before. But the overall environment might consist of many different Grid service containers that are hosted on different autonomous and heterogeneous application servers. Thus, clustering capabilities are needed to "federate" the different Grid service containers resulting in a virtual environment for scalability and resource sharing. Also, such a virtual environment has to support distributed and heterogeneous problem determination and logging, the association of policies with Grid services as a base for request scheduling etc. The corresponding functions are referred to a meta-operating system services.

At the top layer functions are shown that represent various autonomic services of the Grid: For example, Grid-wide workload management that enable a broad range of mechanisms for scheduling requests in the Grid reaching from simple round-robin schedulers to policy-based meta-schedulers in hierarchical Grid topologies enhancing overall availability and scalability within the Grid. Also, functions enabling utility computing (see next section) are at this layer.

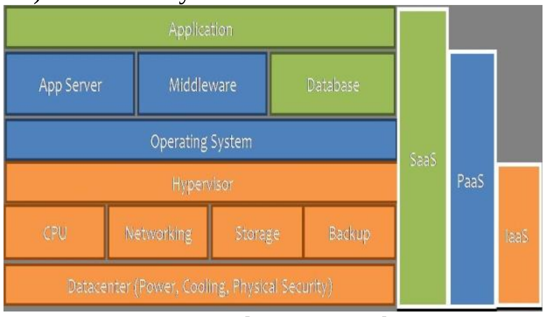

Fig- Grid service stack

3.3 Web service Demand

Finally, such an environment will enable a new computing model called on-demand computing. In a nutshell, this term refers to the ubiquitous availability of compute resources whenever needed and wherever needed. This bares the potential to turn computing into a public utility like water, power, gas, and telephone connections – which is why this model is also referred to as utility computing.
An important step on this path is represented by the concept of a hosted e-utility. A hosted e-utility is a collection of application-related services (both, hardware as well as all required software) that is made available by a service provider to a request or on demand based on particular service level agreements for a certain fee. For example, a request or wants to analyze new genomic data and needs for this purpose a set of certain algorithms, large amount of temporary storage, a set of servers to provide the corresponding compute power, as well a high-

bandwidth connections to the Internet for access to public genomic data. A service provider can provide all of this as a collection of Grid services.

IV. APPLICATION STRUCTURE

Services are either fine grained or coarse grained. From a requestor's perspective, a fine grained service achieves a business goal based on a single interaction, while a coarse grained service typically requires multiple interactions to achieve a business goal. Be- cause a single interaction with a fine grained service suffices, a fine grained service typically does not reveal any of its inner structure, i.e. it is opaque hiding its implementation details. In contrast to this, a coarse grained service does reveal implementation de- tails, especially the set of interactions required as well as their order, i.e. it is transparent making some of its inner structure visible to a requestor. The implementation details revealed by a coarse grained service describe its potential message exchange with the outside world, i.e. business rules that specify in which order and under which conditions which messages are sent to or expected from the requestor and perhaps other third party Web services.

4.1 Two Level Paradigm

In a Web services world actual messages are sent to ports via their corresponding operations. Thus, at the type level a potential message exchange can be specified by defining the potential order in which operations of port types are used and under which conditions. As depicted in this is the same as specifying a business process or a work-flow, respectively, the activities of which are realized by operations of port types. Especially, a coarse grained service appears to be composed of the corresponding services, and consequently coarse grained services are also referred to as composite services. Vice versa, fine grained services are also referred to as elemental services.

This introduces the paradigm of two-level programming to Web services: Programming in the small for implementing the elemental services used by a composite service, and programming in the large for specifying the composite service itself. Programming in the small, i.e. the implementation of elemental services, is done based on usual programming languages (e.g. Java, C#), and based on known component technologies and application server environments (e.g. J2EE, .NET). The corresponding components are hosted and rendered by the environment as Web services, i.e. the elemental services. Programming in the large is done based on a business process language (e.g. BPEL) hosted and run by a workflow system. The corresponding business process is rendered again as a Web service resulting in a composite service.

4.2 Reuse

The two-level programming paradigm introduces reuse at both levels: At the component level, i.e. elemental service level, and at the business process model level, i.e. composite service level. In practice, a vast number of isolated component functionalities does already exist in an enterprise, e.g. in form of purchased standard applications or home grown special applications. Typically, it is the knowledge of how to integrate these component functionalities into a business process that solves a (new) business problem. As a consequence, to become an artifact of reusability a business process model has to have the ability to be easily linked to the component functionalities available at an individual enterprise; a business process model with this property is sometimes called a solution template

V. WEB AGGREGATION

The model of building a composite service as introduced in is one example of an aggregation model for Web services. In this model aggregation is done at the port type level by specifying both, the port types offered as well as required by the aggregate. Furthermore, the aggregation is very much structured and constrained in its behavior by the associated business process model, i.e. it is "choreography"-centric: It prescribes the potential order in which the operations of the aggregated port types are to be used. And it is "pro-active" by defining an execution model that actually drives the usage of the aggregated port types. On the other hand, it is non-recursive in the sense that defining new port types based on its aggregated port types is not its focus.

5.1 Global Model

The definition of a recursive aggregation model (called global model) for specifying collections of new port types is included. This model defines the notion of a service provider type as a set port types. The only structural relation between service provider types is that they make use of each other's services. The relation between service providers and the aggregate itself is that the aggregate inter- face is built from the service provider types' interfaces. Operations of port types of different service provider types can be connected via a directed plug link. A plug link defines a client-server relationship between operations specifying who the initiator is and who the follower within an interaction is. For example, the out-operation op_3 of port type of service provider SP^b is the source of a message sends to the in-operation op_1 of port type of service provider SP^a that consumes this message. It is not required that all operations are source or target of a plug link, i.e. a service provider might offer operations that are not used by other service providers of the Aggregate.

5.2 Web Service Domain

In some application scenarios, a request or needs a collection of related services that he will use in a non-predefined manner. Properties beyond the signature level of a concrete service are irrelevant to a requestor, i.e. individual ports providing the same service are indistinguishable from a requestor's point of view. Specifies a complete environment for such aggregations; the corresponding aggregation model is referred to as service domain. For conciseness reasons, we will take the liberty here to use the same name but describe a variant of this aggregation model.

Basically, a service domain is a set of ports implementing a predefined set of port types. In general, for each

particular port type associated with a service domain there is more than one port implementing this port type. A service domain aggregates these ports by providing for each of its port types a port that functions as a proxy for the set of ports implementing the same port type. When a request or sends a message to this proxy the environment will select one implementing port and dispatch the message to it.
Often, the final outcome of the usage of some services is dependent on the final outcome of the usage of some other services. As a result, an aggregation model is needed that allows dynamically creating temporary collections of services the joint outcome of their usage is determined once the period of usage of the services within the collection is over. The determination and dissemination of the joint outcome is based on a collection- specific set of protocols supported by the participating services, i.e. member of the collection.

VI. CONCLUSION

In this paper we have demonstrated that Web services are the base for a new era of distributed computing. Web services are virtual components hiding from their user's idiosyncrasies of the concrete (application server) technology chosen to implement the Web service. Especially, users can easily mix and match functions from heterogeneous environments into a single application if those functions are rendered as Web services. Based on a service-oriented architecture a user does not even have to care about a particular Web service he is communicating with because the underlying infrastructure, i.e. the service bus, will make an appropriate choice on behalf of the user. This choice is based on policies of both, the user and the Web services qualifying under the user's functional request, and the choice is also influenced by service level agreements and demand for an optimal utilization of the overall environment. We have shown that Grid computing technology and Web service technology are about to converge to provide these features and more, enabling utility computing and on-demand computing. Aggregations of Web services support a broad spectrum of requirements reaching from recursive component construction over advanced provisioning of groups of services to transaction management.

REFERENCES

[1] K. Ballinger, D. Ehnebuske, M. Gudgin, M. Nottingham and P. Yendluri, Basic Profile Version 1.0, http://www.ws-i.org/Profiles/Basic/2002-10/BasicProfile-1.0-WGD.htm
[2] W. Beer, D. Birngruber, H. Mössböck and A. Wöß, Die .Net Technologie, dpunkt Verlag, 2003.
[3] T. Belwood et al, UDDI Version 3.0, http://uddi.org/pubs/uddi-v3.00-published- 20020719.htm
[4] V. Berstis, Fundamentals of Grid computing, IBMCorporation(2002) http://www.redbooks.ibm.com/redpapers/pdfs/redp3613.pdf
[5] D. Box et al, SOAP 1.1, http://www.w3.org/TR/SOAP
[6] C. Boyens and O. Guenther, Trust is not enough: privacy and security in ASP and Web services environments, Proc. ADBIS 2002 - 6th East-European Conference on Advances in Databases and Information Systems (September 8-11, 2002, Bratislava, Slovakia).
[7] S. Burbeck, The Tao of e-business services, IBM Corporation, 2000,http://www-4.ibm.com/software/developer/library/ws-tao/index.html
[8] F. Cabrera, G. Copeland, T. Freund, J. Klein, D. Langworthy, D. Orchard and J. Shew- chuk, Web Services Coordination, BEA Systems & IBM Coporation & Microsoft Corpo- ration, 2002, http://www-106.ibm.com/developerworks/library/ws-coor/
[9] F. Cabrera, G. Copeland, B. Cox, T. Freund, J. Klein, T. Storey and S. Thatte, Web Ser- vices Transactions, BEA Systems & IBM Coporation & Microsoft Corporation, 2002, http://www-106.ibm.com/developerworks/library/ws-transpec
[10] M. Champion, Ch. Ferris, E. Newcomer and D. Orchard, Web Services Architecture, http://www.w3.org/TR/ws-arch/
[11] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, WSDL 1.1,http://www.w3.org/TR/WSDL
[12] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte and S. Weerawarana, Business Process Execution Language For Web Services, BEA Systems & IBM Copora- tion & Microsoft Corporation, 2002, http://www-106.ibm.com/developerworks/library/ws-bpelwp
[13] B. Daum and U. Merten, System architecture with XML, Morgan Kaufmann Publishers, San Francisco, CA, 2003.
[14] D. Fensel, Ontologies: A silver bullet for knowledge management and electronic com- merce, Springer, 2001.
[15] I. Foster and C. Kesselman, The Grid: Blueprint for a new computing infrastructure, Morgan Kaufmann Publishers, San Francisco, CA, 1999.
[16] I. Foster, C. Kessleman, J.M. Nick and S. Tuecke, The physiology of the Grid – An open Grid services architecture for distributed systems integration, Open Grid Service Infra- structure WG, Global Grid Forum, June 22, 2002, http://www.globus.org/research/papers
[17] T. Freund and T. Storey, Transactions in the world of Web services,http://www-106.ibm.com/developerworks/webservices/library/ws-wstx1
[18] J. Gray and A. Reuter, Transaction processing, Morgan Kaufmann Publishers, San Mateo, CA, 1993.
[19] V. Gruhn and A. Thiel, Komponentenmodelle, Addison-Wesley, 2000.