

Service Oriented Dynamic Decoupling Metrics¹

Taixi Xu
Dept. of Mathematics
Southern Polytechnic State
University
1100 S Marietta Pkwy
Marietta, GA 30060, USA

Kai Qian
Dept. of Software Engineering
Southern Polytechnic State
University
1100 S Marietta Pkwy
Marietta, GA 30060, USA

Xi He
School of Computing
South China Normal
University
Guangzhou, China

In the present paper, we develop the service oriented dynamic decoupling metrics at the design phase. The decoupling metrics can be used to measure and evaluate the decoupling attributes of a distributed, service-oriented software architecture that has very significant impacts on the understandability, maintainability, reliability, testability, and reusability of software components. The decoupling metrics can also be used as a criterion for selection of existing service components for compositions. This paper provides a practical guide for evaluating dynamic decoupling between service-oriented components in the service composition such as Business Process Execution Language (BPEL) in software design. A lower decoupled distributed software application will be much easier to understand, update, and expand in the future. The metrics proposed in this paper are based on executable design models from which the dynamic behavior of applications can be inferred.

Keywords: dynamic, coupling, decoupling, metrics, measurement, service-oriented.

1. Introduction

It is well known that the service technology is a component-oriented SOAP based interoperable technology widely adopted in Enterprise Application Integration (EAI) and Business to Business (B2B). The decoupling is one of the most important features of service components that make Web services successful in future business process applications. Software measurement plays an important role in today's software development. Coupling was originally defined as the measure of the strength of association established by a connection from one module to another. Most of the existing techniques and measuring coupling metrics are classified by procedural-oriented and object-oriented static metrics, and are not specific for service component decoupling. In this paper we propose a decoupling metrics based on the black-box parameters of service stateness (stateless/stateful), interaction (one-way, two-way), service interface required (service interface provided (supporting/supported interface), invocation modes (synchronous/asynchronous), self-containment (stand-alone/indirect dependent), implicit invocation (blocking/non-blocking), and binding modes (static/dynamic). [10]

Web service is an implementation of Service Oriented Architecture (SOA). A grid service is another implementation of SOA. Grid enables an effective collection of distributed resources. Grid defines the virtualization of data and resources as well as mechanisms for resiliency, including monitoring, resource and data discovery, and security. [11] In this paper we focus on the decoupling for Web services components. Because all service components provide interface ports to expose its operation services to be used by clients and to take messages or

¹ The 2006 International Conference on Semantic Web and Web Services (SWWS'06)
June 26-29, 2006 WORLDCOMP'06, Las Vegas, USA

notifications from clients. Here the client can be a real application client or a Web service component itself as well. Regardless the type of services, the service component always wraps and encapsulates its service implementations and just provides a public service specification via XML based WSDL or GWSDL interface description specifications. Compared with other component technology, Web services have its distinguished features over other distributed technologies in loose coupling of service compositions. Web services deliver application services on Web and enable a programmable Web not just an interactive Web. Web service is the third generation in Web evolution after static HTML and interactive Web development such as PERL, ASP, JSP, and others. Web services are typical black box reusable building block components in distributed computing. There are many other distributed component frameworks available in industry such as CORBA, MS DCOM, .NET, EJB but only Web service provides a real cross-platform, cross programming languages, cross proprietary restriction, and internet firewall friendly solution to interoperable distributed computing. We can simply say that Web services is a self-descriptive on-line distributed component which exposes its services and functionality via its interface on-line and can be published, located and invoked programmatically over Internet. A key point of the importance of Web services is its ability to support programmatic end points so that developers can use BPEL or other development tools to compose all related Web service together. [2][3]

In this paper we propose a new dynamic decoupling metrics at the design phase to measure the quality of service-oriented enterprise application in the service composition. It focuses on evaluations of coupling at run time rather than evaluations at static coupling at static time. We attempt to predict what coupling will happen instead of what coupling may happen. It can be used as a criterion in the selection of reusing existing services components and design new reusable components, and evaluate the decoupling feature of the whole service-oriented system. Loose coupling will result in a better decoupling measurement. [4][9]

We know that coupling between services come from many aspects. If each service component is completely independent, stateless, and self-contained then the composition of such services components will result in a very low coupling and high decoupling. Theoretically every service component is a stand-alone unit, but in reality many service components rely or depend on other component either they require services from other service components or share with other service components or component itself has its state just like a stateful EJB component. So the service dependency, such as sharing state indirectly and keep state directly, or required services to provide its own services, contributes coupling to the service composition. Second, the service connection modes also impact the coupling attributes such as tight synchronous service invocation vs. loose asynchronous service invocations; non-blocking vs. blocking in asynchronous invocations; loose document vs. tight RPC messaging. We develop a separate metrics for above service request styles. The combination of these two metrics can be used to measure the decoupling feature in a service composition. [5][6][7]

2. Service Dependency Coupling Metrics

All services component are supposed to be stateless so that it can work in a request/response mode without knowing who made the request and remembering what happened in the past. An ideal services component is a completely independent and deployed software unit. But for transaction-oriented business process the application do need to have a repository to keep track of the transaction status either in cache or in a permanent storage. In other word, a stateful service component either ties with its own state or share states with other service components.

Because a XML based on WSDL of a Web service component is just like a black box which provides all exposed information necessary for its client to use, how to find stateness of a service component is an issue. We all know that if a service requires a client to provide its identification than the service component must have its state. If many services require the same identification information that indicates these services shares a same state or repository.

2.1 State Dependency Metric

The Degree of State Dependency (DSD) is defined as:

$$DSD = \frac{1}{n} \sum_{k=1}^n C_k .$$

Where n is total number of components in the domain and k is the component. $C_k = 1$ if the component k has its states, otherwise $C_k = 0$.

DSD is between 0 and 1. The lower the DSD is, the looser the coupling between services components may be. At most of time this type of data is not persistent and should be kept during the session of request/response such as shopping cart.

2.2 State Persistent Dependency Metric

There is another indirect state dependency that multiple services share a same state which can be updated and retrieved by these service components. These types of data are persistent and need to be there for long time such as account information. [2]

Figure 1 shows three Web services ws1, ws2, and ws3 share a repository in a business transaction.

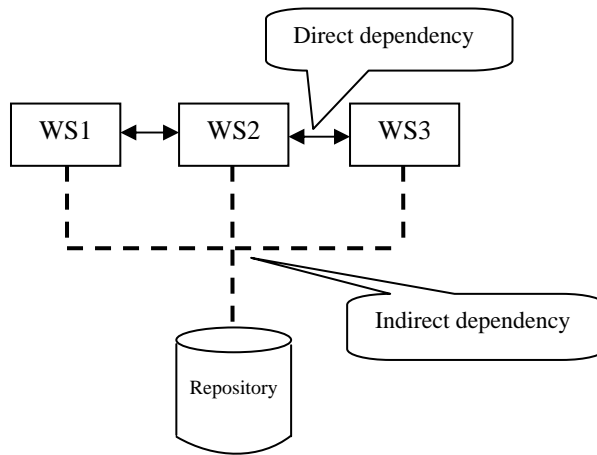


Figure 1 Service Persistent Dependency

We define such state dependency as Degree of Persistent Dependency (DPD):

$$DPD = \frac{1}{n(n-1)} \sum_{i=1}^n \sum_{j=1}^n P_{ij}$$

Where P_{ij} is 1 if service component i participates the persistent data j sharing otherwise is 0, n is the number of service components in the domain and m is the numbers of various persistent repositories. This index shows the average participation times for a service component to tie with other service components indirectly.

DPD is between 0 and 1. The lower the DPD is, the looser the coupling may be.

2.3 Required Services Dependency Metric

A service component can be a composite component which is composed by many other service components by BEPL or other composition tools. A simplest service component is a basic component which does not have any built in components. Following diagram shows the structure design pattern for a composite service component. We can see that one compose component consists of basic = components and composite components as well. (See Figure 2)

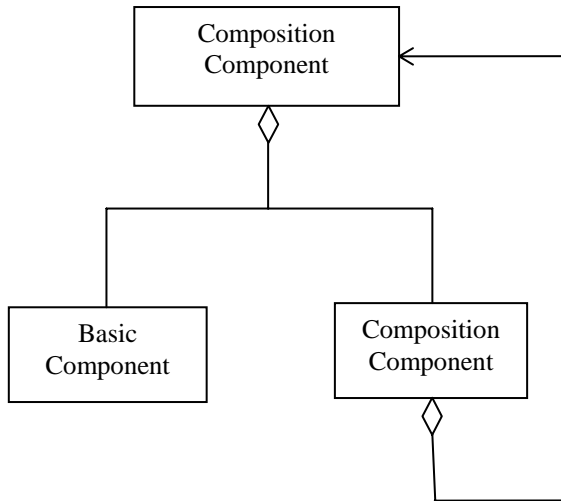


Figure 2 Patterns for Composite Service Components

Reuse a composite component or a basic component will result in different coupling measurement. If a service component of a composite service component is not available on line or the interface of that component is changed it may affect more services than a basic service component.

We define the Average Required Service Dependency (ARSD) as follows:

$$ARSD = \frac{1}{n} \sum_{i=1}^n R_i .$$

Where n is the total number of components in the domain and R_i is the number of required services the service component i requires providing its services.

The lower the ARSD is the looser the coupling will be.

How to find all required services for a service component is a challenge because they are not available in WSDL or GWSDDL interface description.

There are tools available to test a service component to reports all services components used for a specific service component.

3. Web Service Invocation Coupling

Web service invocation is done in either synchronous or asynchronous mode. In the synchronous interaction, a client sends a Web service request and halts its operation while waiting for a response. If the service takes a tremendous computation time to complete or the service is not needed until unpredictable events are triggered an asynchronous interaction must be used instead. The synchronous interaction is the default interaction in Web service operations. Another reason is in some cases there is no backward channel available for the responses to come back synchronously such as e-mail response to a HTTP request.

Asynchronous message-based operation is a single message passing operation, which can be one-way incoming notification and one-way outgoing notification. One-way message passing operation never expects any

response right away. The message may be a signal, a notification, or data to be processed by the target of the message. The two-way out/in request/response operation is a RPC-based operation with combination of incoming and outgoing SOAP message. The source of the message sends out an outgoing SOAP message and expects to get an incoming response SOAP message back right away. In the in/out (solicit/response) mode the target of message gets incoming SOAP message and responds an outgoing SOAP message. Obviously, they are synchronous operations. Figure 3 shows these four types of interactions between any two Web service components or between Web service clients (may be a Web service itself) and a Web service component. [4]

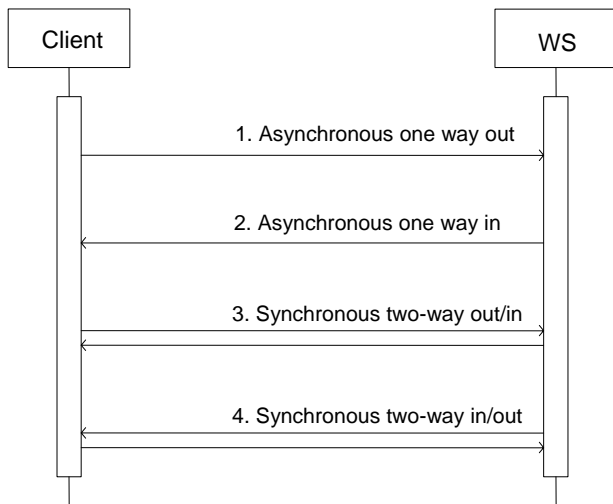


Figure 3. WSDL supported interaction pattern

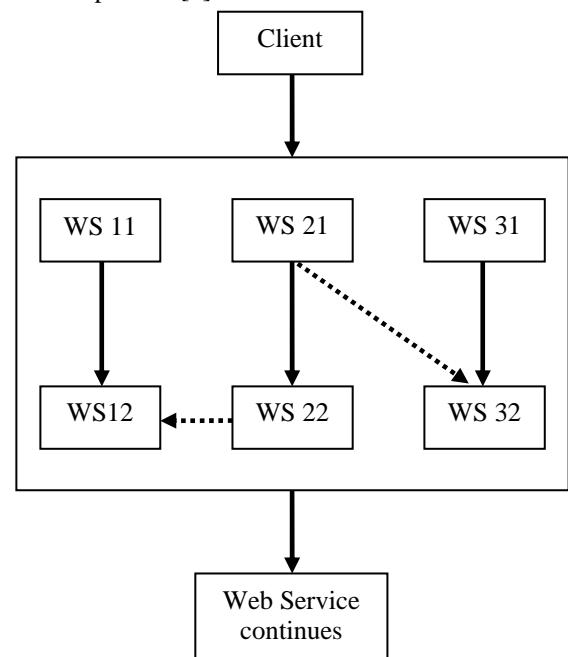


Figure 4: Synchronous vs. Asynchronous Service Invocations

Figure 4 shows the fundamental concepts of Web services composition by BPEL process. The solid arrows represent synchronous connection by sequence. The three parallel sequences represent the service concurrency by flow. The dotted lines are asynchronous one way notification from one service to another service which is also used to control the completion of target service for synchronization. On starting or in the middle of a Web business process, three tasks fork concurrently: Web services WS11-WS12, WS21-WS22, and WS31-WS32. When the three sequence flows are joined, the process continues.

The interaction operation types of Web service component or Grid services components can be derived from the specification of service operations, port types, binding, and services tags in WSDL or GWSDL that can be used for Average Service Invocation Coupling (ASIC) metrics evaluation which is discussed next in this section.

Asynchronous invocation connection has much lower coupling between service requester and service provider than synchronous invocation. An asynchronous requester can register and deregister an event provided by the service provider dynamically so that the impact of the service provider on the requester is much lighter. For asynchronous invocation the non-blocking callback has much lower coupling than blocking callback because the requester must block itself until callback comes back.

The port types of services are specified in WSDL or GWSDL which can be looked up and we can determine the invocation modes.

We define the Average Service Invocation Coupling (ASIC) as follows:

$$ASIC = \frac{1}{n} \sum_{i=1}^n IC_i = \frac{1}{n} \sum_{i=1}^n (W_{i, async} N_{i, async} + W_{i, sync} N_{i, sync})$$

Where IC_i is the invocation coupling for component i ; n is the total number of components in the domain; $W_{i,async}$ and $W_{i,sync}$ are weights of component i satisfying

$$W_{i,async} < W_{i,sync} , \quad W_{i,async} + W_{i,sync} = 1 , \quad i = 1, 2, \dots, n .$$

$N_{i,async}$ and $N_{i,sync}$ are the numbers of non-blocking asynchronous operations and synchronous operations respectively defined in service component i .

The lower the ASIC is the looser the coupling will be there between service components. This index also measures the portability and performance quality attributes. The lower ASIC indicates better system performance in term of time and high maintenance ability.

Because DSD, DPD, ARSD, and ASIC are all estimated as average coupling metrics per component and all of the parameters of these metrics can be identified by either WSDL service interface descriptions or detecting tools or development kits, these metrics can be used as a practice guide in service composition at the design phase to improve the service-oriented enterprise application software quality.

The service dependency metrics discussed in section 2 has more weights over the invocation coupling metrics in the measurement of decoupling quality attribute for service-oriented software design.

4. Conclusion

Coupling is considered to be an important metrics measuring design quality. There are many aspects of coupling affect design quality or other external attributes of software.

In this paper we present new service decoupling metrics for service-oriented, dynamic, and distributed software composition. We can apply these metrics in the selection of service component in the service-oriented software design process and evaluate the service-oriented software as a whole in term of decoupling quality attribute for better software understandability, maintainability, reliability, testability, and reusability

5. References

- [1] H. Washizaki, Y. Yamamoto and Y. Fukazawa, "A Metrics Suite for Measuring Reusability of Software Components", 9th IEEE International Symposium on Software Metrics, 2003
- [2] Rebecca Berrigan, Ewan Tempero, "Understanding Indirect Coupling". Report Number UoA-SE-2003-4. 2003, Software Engineering. University of Auckland.
- [3] Lixin Tao. "Agent-Enable Transformation of E-Commence Portals with Web Services", proc. WSEAS , 2006
- [4] K. Qian, "Design Pattern for Web Services", Proc. of SPND 2004
- [5] F. Curbera, I. Silva-Lepe, and S. Weerawarana, "On the Integration of Heterogeneous Web Service Partners", 2001
- [6] H. Deitel, "Java Web Services for Experienced Programmers", Prentice Hall, 2003
- [7] H. Deitel, "Web Services, a technical Introduction", Prentice Hall, 2003
- [8] M. Clark, P. Fletcher, J. J. Hanson, R. Irani, M. Waterhouse, and J. Thelin, "Web Services Business Strategies and Architectures", 2002
- [9] Gamma, Helm, Johnson, Vissides, "Design patterns", Addison-Wesley, 1998

[10] S. Yacoub, H. Ammar and T. Robinson, West Virginia University, "Dynamic Metrics for Object Oriented Design", Sixth International Software Metrics Symposium (METRICS'99)

[11] Ian Foster, "Grid's place in the service-oriented architecture",
<http://www.computerworld.com/databasetopics/data/story/0,10801,97919,00.html>