

Engineering Well-Formed Services

Sankaran Prithviraj

Satyam Computer Services Limited
Chennai, India

Prithviraj_S@satyam.com

Abstract

This paper brings attention on a need for engineering web-services design while composing (or orchestrating) composite-services from component services. Both manual and static - or - automated and dynamic service compositions demand special attention to apply all known principles to engineer well formed services. This paper points out lack of engineering can result in unpredictable erroneous behavior. A few such errors are pointed out. Such situations can be semantic errors too. This paper proposes a solution by analyzing service-tree structure, which models constituent components of a service. This paper further goes to identify a few attributes that can be used to describe well formed services; and also illustrates how tree data structure can be used to analyze and improve services' design. Literature survey gives good hope to couple certain mechanism with the proposed service tree to address dynamic situation requirements too.

Keywords

Service composition; automatic composition of Web services; Semantic compositions of WS

1. Introduction: Problem Context

As service orientation has been gaining importance in enterprises, it becomes important to engineer well formed services. By engineering we mean to apply all formal principles to the design of services. 'Design of services' refers to composing services from atomic component services or components. OASIS-Open organization's 'Reference Model of services' (1) will serve as reference for all terms used in this paper related to 'service.' Web services are collections of

business functionalities in software modules that are net-work accessible through standardized XML based messages (1), (2).

There are two popular service composing techniques: 1. SCA - Service Component Architecture (3), 2. WS-BPEL (4).

SCA addresses concern of agility and reusability; it provides facilities to produce composite applications from functionalities of existing enterprise systems mostly within or outside an enterprise.

WS-BPEL or simply BPEL takes a process approach to compose services that are essential to execute steps of a process. It has provision to describe a process in abstract or executable way. BPEL focuses on representing compositions where flow of process and bindings between services are known a priori. A more challenging problem is to compose services dynamically, on demand (5).

Both SCA and BPEL make use of XML and passing messages between service consumer and provider.

Some of major reasons for issues creeping in service compositions are:

1. Services expose only their interfaces and not their implementation details to composer or consumer. This becomes source for semantic errors on composition.
2. A large number of services are available, both within enterprise as well as in the community outside, from which one can pick needed components to compose required service. Both for manual or automatic service composition, this poses challenges (6). One needs an efficient search, filtering and service comparison and equi-

valence detection facilities to compose services from these component services.

- Loose coupling by message exchanges between consumer and provider, can lead to a lot of messages being exchanged between them due to faulty design of composition, including those pointed out in this paper,

The ‘languages’ that are available for service choreography or orchestration or composition such as BPEL or SCA are aimed to provide facilities for easy service composition using interfaces exposed by service providers. However the language constructs themselves won’t prevent an absurd or bug ridden composition. Also as we know, semantic bugs are hard to detect by machines and hence the bugs creep-in. The consolation is, for example ‘abstract description’ of process in BPEL (6), helps in automating analysis of the composite services. Characteristics of composite services can be analyzed statically or to a limited extent dynamically (6), (9) -outside execution environment- for errors such as recursion, dead-lock, and also for SLA and cost, scalability, impact on maintainability, through use of dedicated special purpose tools or simulators. Engineering principles discussed in this paper and others- (5), (6), (7), (9), (10), (11) -can form basis for designing such tools and hence designing well formed services.

This paper points out one such error situation that can occur. It also proposes solutions for analyzing such possible errors in composite-services by modeling them as ‘service tree.’ Right now the paper confines only to manual analysis; it discusses possibilities for automation. Also this paper discusses usage of service-tree to analyze various measures for well-formed services.

This paper is organized in the following way: the above introduction describes the context of the problem taken for analysis in this paper. Section-2 states the problem. Section-3 explains the problem through illustrations. Section-4 summarizes related work in literature. Section-5 explains proposed solution using service tree analysis. Section-6 discusses how to use service tree for calculating and measuring through attributes for well-formation of services. Section-7 describes limitations and extents of this approach. Section-8 presents plan for future work.

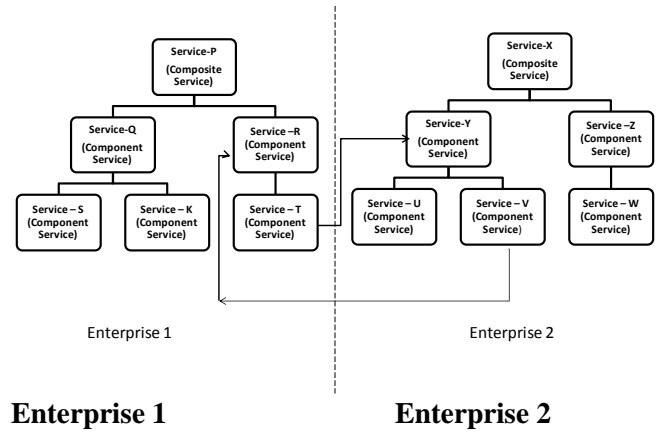
2. Problem Statement

During service composition one can end up with ‘entanglements’ as explained below.

The word entanglement is used in analogues to a rope or thread getting entanglement. The sequence of execution of component-services in a composite-

service is equated to the rope that can be ‘entangled.’ (SCA uses a term ‘wire’ for composing services and this term ‘wire’ is closest to thread and lead us to visualize entanglement.)

Here figures 1 and 2 describes some such entanglement. The key reason for such inadvertent entanglement is lack of visibility for consumers, beyond interface, how a service is implemented.



Enterprise 1 **Enterprise 2**
Figure 1: Indirect Recursion

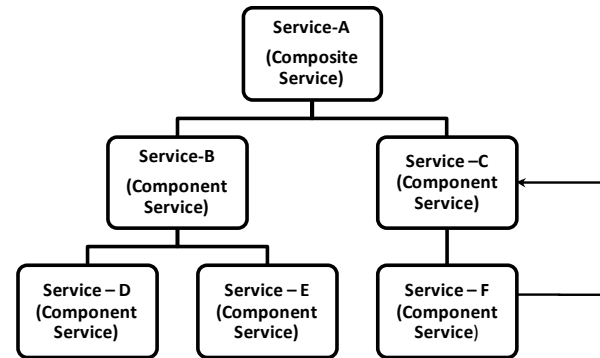


Figure 2: Mutual Recursion

The figure-1 explains sequential execution of services. Composite-Services at a higher level, in a tree like structure, take the help of component services at lower level. Nodes are coordination points or orchestration points of all constituent component services those roll up to the node.

This service-tree structure is description of flow of connections and components known a priori. In dynamic - detect, compose and consume - situation, the above picture can be thought of as a snap shot at execution time when all those (component) services come together to render the intended final composite-service.

There are many entanglement situations such as recursion or mutual-wait; some recursion situations are:

1. Self recursion
2. Mutual recursion on the same service-tree
3. Indirect recursion between trees.

A mutual recursion is as described in figure-2 where service- C takes help of Service-F; but the service-F itself is defined in terms of Service-C. For example, to pay mobile phone charges, we use mobile phone to access banking services as mobile banking service and initiate payment to mobile phone service provider. If this is automated, the service will not get entangled under several conditions: when mobile phone does not have enough balance left out or becoming zero (in pre-paid service connection), as the fund transfer is about to happen from bank account to mobile service provider.

Indirect recursion can happen between two service-trees as shown in figure 1. The lack of knowledge how a service is implemented can be consumed in this way to result in an entanglement especially if these two trees are in two different organizations.

References (5), (7) give life situations that can end up in entanglement.

2.1 Problem Description

The entanglement problem is unique in composing services and not in other related areas such as ‘composing’ objects using conventional programming languages.

Language compilers provide explicit or implicit recursion construct. Also the caller and called module both execute in a common environment and hence the execution environment (compiler, operating system) can deduct these kinds of abnormal recursions and can terminate the execution. Each recursion will take additional stack space and finally stack will over flow and hence execution environment will terminate the process. (Debugging may be a difficult.)

But in Service provider-consumer situation, for each recursion messages are sent or received. There is no common execution environment comprising both consumer and provider. They are loosely coupled. And hence no option to recognize and create stacks necessary for recursion operations. Also composing languages such as BPEL do not provide ‘recursion’ as a construct. In spite of it, one could create recursion and hence as a consequence recursions will create a lot of messages that may queue both at the consumer (as results message) as well the provider (as request message) side as explained in section 3.

Also one need to comprehend that the recursion pointed out could be semantic because of transparen-

cy of how services are implemented and composed using only knowledge of their interfaces.

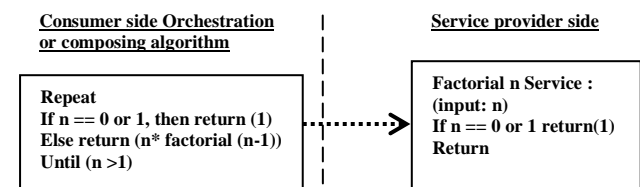
The semantic recursions cannot be deducted during syntax verification or compilation; and this also can occur whether language constructs for recursion are available explicitly or not for service composition as mentioned in section ‘introduction.’

3. Illustrating situations to explain problem Statement

We will examine whether to use recursion or not in service composition. SCA or BPEL does not provide recursion as a structured construct. Recursion is unnatural in process modeling environment whereas it cannot be ruled out in composite applications composing environment.

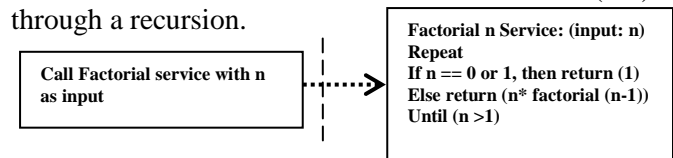
If need to use recursion in service composition, following illustrates right or wrong implementation. The same is illustrated for factorial (n) that needs either explicit or implicit recursion. (The structured constructs used in the following figures are not exact constructs either in SCA or BPEL but it is to illustrate the point.) We are composing services that offer to return value for factorial (n).

One wrong (Implementation 1) way is recursion is on consumer side; Also note the value of service provided is very low and consumer need to be almost aware of details of factorial; but this example is chosen to illustrate a major issue - the number of messages that will go back and forth between consumer and provider:



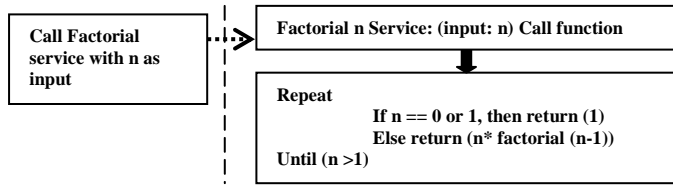
Implementation-1 (wrong Way)

Another wrong way is Implementation 2: recursion is on service provider side; and the service provider calls his/her own service to determine factorial (n-1) through a recursion.



Implementation-2 (wrong Way)

The correct way to implement is (Implementation-3) to implement the recursion in a function in a language based component and call it at the service provider end.



Implementation-3 (Correct Way)

Services are not meant to be used as outlined in implementation 1 or 2. So designers, composing services, need to be aware of such erroneous implementations and avoid them at design stage.

4. Solutions in Literature

We recognized that we need solutions for two different situations: 1. for static situations or manual composing of services, 2. for dynamic situations or automated composing of services

BPEL and SCA provide facilities for category one (5) where flow of the process and the bindings between services are known a priori.

4.1 Related Works

Reference-(8) points out similarities between objects and services. It argues that services are one level higher abstraction of objects or components. Also it attempts to characterize well-formed service. This paper attempts to combine business views of services to ITs view. But the treatment is mostly at business perspective whereas the semantic-recursion problem can occur during implementation.

Reference (7) brings certain key attributes of well formed services. But the reference views service composition as QoS routing problem and analyzes impact of services on underlying network. Also this reference uses the term service tree; one needs to work a lot to stretch these concepts to apply to software view of services.

(9), (10) bring a formal method based on PSM – problem solving method. This reference considers this class of problems as semantic issues. Hence recommends Web Services to provide semantic information to consumer. In this method services (re-usable components) and tasks are brought as a tree; the input output interactions between the tasks; the flow control that describes the task execution. Based on PSM, the paper demonstrates an infrastructure for Semantic web services development. This infrastructure is a kind of language environment and it is ODE-SWS. What is relevant and interesting to us is, it allows to describe service compositions and the same is made available to the service consumer. This frame work

verifies completeness and consistency of composed services.

However process algebra approach given in (11), (6) is easier to understand, amenable for machine-comprehension and analysis of to-be-consumed service trees. The process algebra notation is similar to abstract model of WS-BPEL. Hence after analysis it can be converted into WS-BPEL.

5. Proposed Solution: Service Tree Analysis

We propose visualizing a composite service as service-tree.

Service Tree structure

1. Depicts hierarchy of component-services that make up composite-service
2. Gives a lot of ease to analyze characteristics for well formation
3. Reveals many aspects of composite-services
4. Lists all components on which the composite service is built
5. Describes whether a tree connects to other tree or trees inside or outside parent organization
6. Facilitates manual or automated analysis for errors such as recursions or dead lock
7. Helps to identify critical path of service execution
8. Indicates parallel execution paths
9. Leaves represent component services
10. Depth indicates possible dependency of the composite-service
11. Nodes are orchestration points
12. Can be balanced using suitable algorithms for ‘balancing’ (out of scope for this paper)

5.1 Execution Entanglement

Presence of semantic recursions or such entanglements in a given service tree can be detected by manually going through the service tree. In dynamic situations, one need automated recognition of equivalence of component-services. See also discussions in ‘Future Work’ section of this paper.

6. Measuring well formation of services using ‘Service Tree’ Analysis

6.1 Attributes of well formed services

Services can be characterized as well formed through a set of attributes. Reference (8) points out following attributes for well formed services: re-usable, abstracted, published, formal contract, technology neutral, or consumable. (7) Identifies and discusses – Efficient, Robust, Adaptive, Scalable – as attributes.

From software perspective we propose that following attributes could characterize well formation of

services: Execution entanglement, Maintainability, Performance, Scalability, and Reliability.

For each of these attributes one need to define critical measures along with what is minimal acceptable or a highly desirable value. Health of wellness can be monitored based on these accepted values; an agreed formula for aggregating them will also reveal health of total wellness.

6.2 Illustrating Use of Service Tree for analyzing measures of well formed Service

Of course, tree structure of modeling services makes them amenable for calculating these measures.

6.2.1 Maintainability, Reliability and Dependability

If one of the component-services is pulled off or decommissioned from service or under maintenance, then the composite-service cannot deliver its intended service. In a dynamic situation the system should locate an equivalent alternative service.

Thus number of component-services a parent has, directly reveals maintainability as well its dependability or reliability. Maintainability is also affected by accessibility of its components for maintenance. If component-services are not in same organization then accessing failed services to maintain is not that easy.

Let us see how Service-tree structure can help in comprehending these facts:

Maintainability of a parent-service = \sum (maintainability of its components).

The depth of a tree indicates ease-of-access of components for maintenance. Difficulty to maintain is directly proportional to number of levels in the service-tree. Number of nodes indicates service orchestration points. Any orchestration point is difficult to maintain since all parameters may not be within ones control to maintain. Maintain to fine-tune SLA (Service level Agreement) is also as complex as the tree is.

This is true for reliability too. The reliability of composite service depends on the weakest reliable element it constitutes. The classical definition of dependability of composite-service = π (Probability of failure of each component). The depth of the service-tree also indicates the reliability factor. Lesser the level better the reliability and hence the dependability of the service.

6.2.2 SLA (Service Level Agreements) as Performance Measures

SLA of a service is determined by the longest path (critical Path) in a service-tree; to say in popular way,

the speed of the relay race is determined by the slowest runner.

Slowest in the chain determines scope for improving SLAs. If combined SLA of a tree is more than desired value of SLA for the composite-service, then the composite-service needs to be re-composed by some other way or with different faster components.

Also one cannot improve SLA beyond sum total of all SLAs of components in critical path in a service-tree. This gives upper bound for performance improvements.

One way to improve SLA is to reduce number of nodes in critical path of the service-tree. Also one can not commit an SLA for parent better than the slowest component in the critical path. Increasing parallel execution of components will improve the SLA. This means 'balancing' the tree as we do in balancing the tree data structure (8), (9). Also decreasing the level of tree will improve the SLA.

Quantitatively one can assign SLA value to each node or service component and by traversing the tree in the critical path one can arrive at SLA of composite-services. Thus use of tree structure gives a lot of advantages to induct quantitative techniques.

6.2.3 Scalability

The service-tree is useful in analyzing points to improve scalability. Each node can be supported to handle more requests for service. Balancing the service-tree will induct parallel execution of component-service execution paths and thus it can reduce bottle necks. Thus it will improve scalability. Providing execution support at each node also will improve scalability. By analyzing the critical path for taking longer time to execute service, one can augment those nodes with additional resources using queuing theory principles too. (7) is a good resource for detail.

7. Limitations of Service Tree Analysis

Experience indicates (7), (12) that all services need not necessarily form a tree structure. Literature indicates a network is an alternative structure to represent services the way they are free to get combined to form complex services. Hence many [Petri net based] approaches are also being applied to analyze the services. Tree is a special case of network.

Taking clue from most successful field of data bases, one can investigate application of relational model to services too. The Object tree has been converted into relational model for applying proven formal methods such as relational algebra. This is close to process algebra discussed in section 'Related work' and also in 'Future Work.'

8. Future Work

It is planned, as future work, to use process algebra (6) for issues pointed out in this paper.

Interestingly this reference (6) uses process-tree model to evaluate semantic equivalence of two processes and this is essential for automation.

To effectively use the process algebra we need services to provide, as semantics, the implementation detail of to-be-consumed service as service-tree to the consumer. Reference (5) provides a solution for the same. OWL (or OWL-S) is XML based ontology to describe service semantics. This semantic description will facilitate external agents to understand both functionality and internal (service tree) structure of SWSs (Semantic Web Services) to be able to discover, compose, and invoke SWSs automatically. DAML-S partitions a semantic description of a web service into three components: one of them - the process model component- can be used to send service tree. Also this is similar to the business process model in WS-BPEL. Hence process algebra can be used to extract the service tree or process algebra of the service to-be consumed (6).

9. Summary

This paper brings the attention of engineering design of services.

1. Loose coupling by message exchanges between consumer and provider,
2. A large number of services from which the consumer need to pick the right one to consume and
3. Interface is the only thing services expose –

are sources of challenges in composing a correct service both in static as well as in dynamic situations. By identifying specific problem that can occur during service composition, it proposes solution based on use of ‘service tree’ depicting hierarchy of component services that make up a composite service. It also illustrates use of service tree to calculate measures of attributes of well-formed services. This paper reviews a few key approaches that are available to automate such verification and validation. The proposed service tree approach is also amenable for automation.

Currently most of the solutions are suitable for manual or human-assisted automation for discovering, designing and consuming of services.

Evolution of Semantic Web services and other related mechanisms give a good hope for complete automation in the near future.

10. Acknowledgements

I thank all reviewers of the draft for their valuable comments; I thank also Ram_Dixit, who has spent a

lot of effort in pointing out areas to improve in this paper. Also I thank Sripriya_Krishnan for her valuable contribution to bring this text into prescribed format.

References

1. *Reference Model for Service Oriented Architecture 1.0 OASIS Standard.* : <http://www.OASIS-open.org>, October 2006.
2. *Web Services Conceptual Architecture (WSCA 1.0).* : <http://www.ibm.com/software/solutions/webservices/pdf/WSCA.>, SCA., May 2001.
3. *Service Component Architecture Assembly Model Specification Version 1.1.* : <http://www.OASIS-open.org>, March, 2008..
4. *Web Services Business Process Execution Language Version 2.0.* s.l. : WWW.OASIS-open.org, April 2007.
5. **Evren Sirin, James Hendler, and Bijan Parsia** *Semi-automatic Composition of Web Services using semantic descriptions.* : Web Services: Modeling, Architecture and Infrastructure Workshop in Conjunction with ICEIS, 2003.
6. **Gwen Sala, et.al.** *Describing and Reasoning on Web Services using Process Algebra:* www.inrialpes.fr/vasy/people/Gwen.Salaun/Publications/Papers/icws04-ext.pdf, 2006.
7. **Jingwen Jin, Klara Nahrstedt.** *ERAS: Efficient, Robust, Adaptive, and Scalable Service Composition in Overlay Networks.* : cairo.cs.uiuc.edu/publications/papers/tr_eras_jin.pdf.
8. **Wikies, David Sprott and Lawrence.** *Understanding Service-Oriented Architecture.* : CBDi Forum, January 2004.
9. **Oscar Corcho1, et.al.** *ODE-SWS: A Semantic Web Service Development Environment.* Spain : International Workshop on Semantic Web and Databases (SWDB),wotan.liu.edu, 2003.
10. **Asunción Gómez-Pérez, et.al.** *A Framework for Design and Composition of Semantic Web Services.* : IEEE INTELLIGENT SYSTEMS, 2004.
11. **Andrea Ferrara** *Web Services: A Process Algebra Approach.* DIS - Universit'a di Roma "La Sapienza" Via Salaria 113, 00198 Roma, Italia June 2004.
12. **Rachid Hamadi, Boualem Benatallah** *A Petri Net-based Model for Web Service Composition.*. Proceedings of the 14th Australasian database conference, 2003