Building Applications Using End to End Composition of Web Services

Vikas Agarwal, Girish Chafle, Koustuv Dasgupta, Neeran Karnik, Arun Kumar,

Ashish Kundu, Anupam Mediratta, Sumit Mittal and Biplav Srivastava
IBM India Research Laboratory
Block 1, IIT, New Delhi 110016, India
{avikas,cgirish,kdasgupta,kneeran,kkarun,kashish,anupamme,sumittal,sbiplav}@in.ibm.com

Description

Web services have received much interest in industry due to their potential in facilitating seamless business-to-business or enterprise application integration (S. Staab et al. 2003; Srivastava & Koehler 2003). Web services offer standardized interface description, discovery (using a registry like UDDI) and messaging mechanisms. Also, the programming tools and runtime environments for web services have now matured. A component-oriented software development approach where each piece of software is wrapped as a web service would offer substantial benefits in application integration and we demonstrate this for a mobile service provider scenario.

Given the intense competition in the telecom sector, mobile telephony service providers need to continually develop compelling applications to attract and retain end-users, with quick time-to-market. Often, if a competitor introduces a new service, the service provider must offer a similar or better service within days/weeks, to avoid losing customers. Also, a service provider can attract enterprise customers by offering custom-developed value-added services that leverage its telecom and IT infrastructure. Enterprise customers typically offer significantly higher margins than consumers, and are thus more attractive. Service providers therefore need tools and standards-based runtime platforms to quickly develop and deploy interesting applications for their clients. This would assist in their transition towards "on demand", responsive businesses.

Mobile user applications often rely on several, relatively simple building blocks – user profile look-ups, address books, location-tracking services, accounting and billing services, etc. Many of these building blocks are already in place, but they are not easy to reuse and integrate into new applications because they are not built using standardized frameworks or component models. This leads to high development costs, and substantial time-to-market for new services. This could be alleviated by building applications using the service-oriented architecture (SOA) paradigm, using web services as the underlying abstraction.

Two different approaches have been taken to standardize and compose web services. The business world has adopted a distributed systems approach in which web service instances are described using WSDL, composed into flows with a language like BPEL,and invoked with the SOAP protocol. Academia has propounded the AI approach of formally representing web service capabilities in ontologies, and reasoning about their functional composition using goal-oriented inferencing techniques from planning (McIlraith, Son, & Zeng 2001). These approaches by themselves are piecemeal, and insufficient. The former has focused on the execution aspects of composite web services, without much consideration for requirements capture and the development process. The latter approach has stressed the feasibility of service composition based on semantic descriptions of service capabilities, but its output cannot be directly handed off to a runtime environment for deployment.

Our service creation methodology, based on web service composition techniques, consists of the following steps (Agarwal *et al.* 2005):

1. *Service Representation:* Representing the available services and their capabilities.
2. *Requirements Specification:* Specifying the desired functionality of a new service.
3. *Composition:* Constructing a composition of available services that provides the desired functionality.
4. *Composite Service Representation:* Representing the new composite service and its capabilities so that it can be programmatically deployed, discovered and invoked.

Our system takes an end to end view that synergistically combines the AI approach and the distributed programming approach currently adopted by academia and industry respectively. It drives the composition process right from specification of the business process, through creation of desired functionality using planning techniques, through generation of a deployable workflow by selection and binding of appropriate service instances, to finally deploying and running the composite service. This integrated solution achieves the best of both worlds and provides scalability to the composition process. We have built a service creation environment that realizes this approach in terms of the following phases of composition:

1. **Logical Composition:** This phase provides functional composition of *service types* to create new functionality

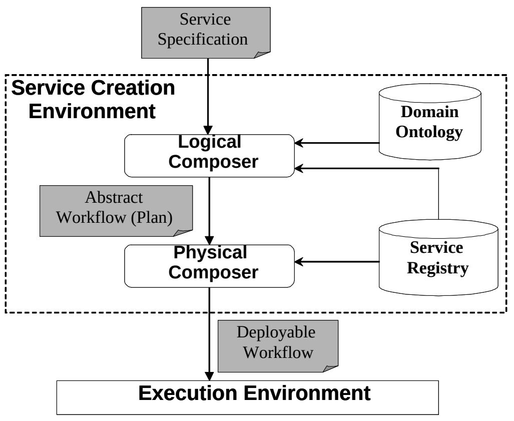

Figure 1: System Overview

that is currently not available.

2. **Physical Composition:** This phase enables the selection of component *service instances* based on non-functional (e.g. quality of service) requirements, that would then be bound together for deploying the newly created composite service.

This basic approach to automating the process of service creation is illustrated in Fig. 1. A **Service Registry** contains information about services available in-house as well as with participating 3rd-party providers. The *capabilities* of each available service *type* are described formally, using domain-specific terminology that is defined in a **Domain Ontology**. When a new service needs to be created, the developer provides a **Service Specification** to the **Logical Composer** module. Driven by the specified requirements, the Logical Composer uses generative planning-based automated reasoning techniques to create a composition of the available service types. Its goal is to explore *qualitatively* different choices and produce an abstract workflow, i.e. a plan (assuming a feasible plan exists) that meets the specified requirements.

In order to turn the plan into a concrete workflow that can be deployed and executed, specific instances must be chosen for the component services in the plan. The **Physical Composer** uses scheduling and compilation techniques in selecting the best web service instances to produce an executable workflow. The focus is now on *quantitatively* exploring the available web service instances for workflow execution. It queries the registry for deployed web service instances, to accomplish this task.

The workflow generated by the service creation environment must then be deployed onto a runtime infrastructure, and executed in an efficient and scalable manner. This is especially important in environments like that of a mobile service provider, where the number of end-users is likely to be very high. The state of the art is to execute the workflow using a workflow engine such as WebSphere Process Choreographer[1], with data flowing back and forth from this engine to the component web services. Our **Execution Environment** instead orchestrates the workflow in a *decentralized* fashion, with partitions of the flow executing concurrently, in network-proximity with the component services they invoke. These flow partitions are generated automatically by a Decentralizer tool, using static analysis of the input BPEL flow. The communication amongst these partitions is designed to minimize network usage, while retaining the original flow semantics. This, in conjunction with the added concurrency, results in better scalability and performance. For more details on our Execution Environment, please refer to (Chafle *et al.* 2004). The demo focuses on the Logical and Physical composition stages.

In summary, we demonstrate an end to end working prototype of how web services composition can be leveraged for business process integration by synergistically combining the strengths of goal-oriented reasoning and performance driven instance selection (Agarwal *et al.* 2005) by: (a) Ontology matching, composition at the type level with service matchmaking (b) Composition at the physical level with instance selection (c) Deployment onto a decentralized workflow orchestration infrastructure. In AI planning, the potential advantage of resource abstraction, whereby causal reasoning is decoupled from resource reasoning, is well-established (Srivastava, Kambhampati, & Do 2001). Our work can be seen as applying the same idea to web services composition. Specifically, we differentiate web services at the twin levels of web service types and instances. Our phased approach is easier for the user to work with and limits the impact of frequent deployment and runtime changes on the goal-driven composition.

[1]http://www.software.ibm.com/wsdd/zones/was/wpc.html

References

Agarwal, V.; Dasgupta, K.; Karnik, N.; Kumar, A.; Kundu, A.; Mittal, S.; and Srivastava, B. 2005. A service creation environment based on end to end composition of web services. In *Proceedings of the 14th International World Wide Web Conference, Japan*.

Chafle, G. B.; Chandra, S.; Mann, V.; and Nanda, M. G. 2004. Decentralized Orchestration of Composite Web Services. In *Proceedings of the 13th International World Wide Web conference, NewYork*.

McIlraith, S.; Son, T. C.; and Zeng, H. 2001. Semantic Web Services. *IEEE Intelligent Systems, Special Issue on the Semantic Web.* 16(2):46–53.

S. Staab et al. 2003. Web services: Been there, done that? *IEEE Intelligent Systems* 72–85.

Srivastava, B., and Koehler, J. 2003. Web Service Composition - Current Solutions and Open Problems. ICAPS 2003 Workshop on Planning for Web Services.

Srivastava, B.; Kambhampati, S.; and Do, M. B. 2001. Planning the project management way: Efficient planning by effective integration of causal and resource reasoning in RealPlan. *Artificial Intelligence* 131(1-2):73–134.