Chapter III

Service Composition: Concepts, Techniques, Tools and Trends

Boualem Benatallah, University of New South Wales, Australia
Remco M. Dijkman, University of Twente, The Netherlands
Marlon Dumas, Queensland University of Technology, Australia
Zakaria Maamar, Zayed University, United Arab Emirates

Abstract

This chapter provides an overview of the area of service composition. It does so by introducing a generic architecture for service composition and using this architecture to discuss some salient concepts and techniques. The architecture is also used as a framework for providing a critical view into a number of languages, standardization efforts, and tools related to service composition emanating both from academia and industry and to classify them in terms of the concepts and techniques that they incorporate or support (for example, orchestration and dynamic service selection). Finally, the chapter discusses some trends in service-oriented software systems engineering pertaining to service composition.

Introduction

The last decade has seen organizations worldwide expose their operations on the Web to take advantage of the commoditized infrastructure and the potential for global visibility and increased business process automation that Web technologies offer. An overwhelming number of organizations have reaped the benefits of the Web by making their applications available to their customers and partners through interactive interfaces combining Web forms and dynamically generated Web pages. This has seen the Web evolve from a vehicle for information dissemination to a vehicle for conducting business transactions, albeit in a manual way.

The next step in the evolution of Web technologies is the emergence of Web services (Alonso, Casati, Kuno & Machiraju, 2003). Web services bring together ideas from Web applications on the one hand (for example, communication via document exchange) and distributed computing on the other hand (for example, remote procedure calls and communication middleware). The outcome of this convergence is a technology that enables applications to communicate with each other in a programmatic way through standardized message exchanges. This is expected to trigger a move from a Web of mostly manual interactions to a Web of both manual and programmatic interactions.

There are several definitions of Web services, most of which agree on saying that a Web service is a software application available on the Web (through a URI) whose capabilities and modus operandi are described in XML and is able to communicate through XML messages over an Internet transport protocol. At present, a widely accepted core infrastructure for Web services is the so-called Web Services Stack which is essentially structured around three XML-based standards: SOAP, WSDL, and UDDI (Curbera, Duftler, Khalaf, Nagy, Mukhi & Weerawarana, 2002). These three standards are intended to support the tasks of service description, discovery, and communication.

This basic core infrastructure is currently being used to build simple Web services such as those providing information search capabilities to an open audience (for example, stock quotes, search engine queries, auction monitoring). However, it has rapidly become clear that this core infrastructure is not sufficient to meet the requirements of complex applications (especially in the area of B2B integration) since it lacks abstractions for dealing with key requirements, such as security, reliability, transactions, composition, service level agreements, and quality of service, among others (Medjahed, Benatallah, Bouguettaya, Ngu & Elmagarmid, 2003). In light of this, several efforts are underway to design a standard comprehensive infrastructure for Web services.

In particular, the development of new services through the composition of existing ones has gained considerable momentum as a means to integrate heterogeneous enterprise applications and to realize B2B e-commerce collaborations. Unfortunately, given that individual services are developed using manifold approaches and technologies, connecting and coordinating them in order to build integrated services is delicate, time-consuming, and error-prone, requiring a considerable amount of low-level programming and system administration efforts. This observation has sparked a wave of R&D efforts in an area often known as “service composition.”

Stated in simple terms, service composition aims at providing effective and efficient means for creating, running, adapting, and maintaining services that rely on other services in some way. In order for service composition to deliver on its promises, there is a need for development tools incorporating high-level abstractions for facilitating, or even automating, the tasks associated with service composition. Hence, these tools should provide the infrastructure for enabling the design and execution of composite services.

This chapter provides an overview of the benefits and pitfalls of service composition, the functionalities that the supporting platforms are required to provide, and the extent to which these requirements are addressed by the current state of the art. However, the chapter will not address in detail system issues such as reliability, transactions, and security. We present the main concepts for service composition by presenting a generic tool architecture for service composition, covering aspects such as design of composite services and composite service execution. Based on these concepts, we provide a survey of service composition models, methods, and supporting technologies.

The chapter is structured as follows. The Generic Architecture section discusses the foundation concepts in Web services composition by introducing a generic tool architecture for service composition. The Languages for Service Composition section overviews language support for Web services description and composition, covering the design module of the generic architecture. The Platforms for Composite Service Execution section reviews research efforts and commercial platforms for web services composition by covering the runtime module of the generic architecture. The Trends Relevant to (Web) Service Composition section reviews some trends in Web services technologies, and the last section provides concluding remarks.

Generic Architecture

From an architectural point of view, a tool environment for service composition should provide at least the following modules:

- **Design module.** This module offers a graphical user interface for specifying composite services. The module may also support translation of a composite service design into a description language. More advanced design tools may support the automated verification and/or simulation of composite service designs on the basis of a formal language.
- **Runtime environment.** This module is responsible for executing a composite service and routing messages between its components. It is also responsible for monitoring and fault and exception handling. The runtime environment may additionally support dynamic service selection and binding as discussed below.

Figure 1. Generic architecture for a service composition tool

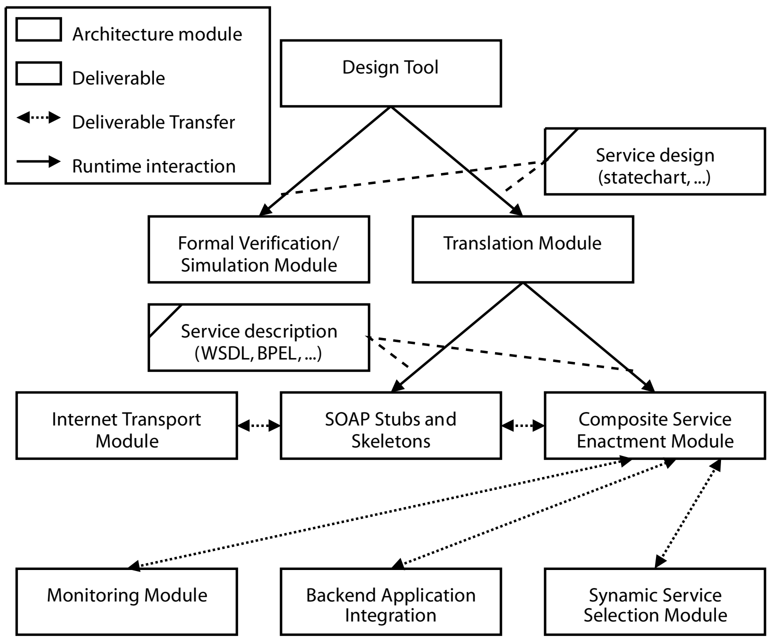

Architecture module
Deliverable
Deliverable Transfer
Runtime interaction
Design Tool
Service design (statechart, ...)
Formal Verification/ Simulation Module
Translation Module
Service description (WSDL, BPEL, ...)
Internet Transport Module
SOAP Stubs and Skeletons
Composite Service Enactment Module
Monitoring Module
Backend Application Integration
Synamic Service Selection Module

Figure 1 represents the generic architecture in more detail. This section explains the generic architecture further.

Composite Service Design

A tool for composite service design supports a composite service design methodology. A methodology consists of design languages, formalisms that are coupled with these design languages, and design approaches. Design languages are graphical notations that can be used by stakeholders in a design process to represent a design from their perspective. They focus on representing a service composition in a way that is easy to understand for the stakeholders. Formalisms are mathematical languages that can be used to represent a particular aspect of a design. As a mathematical language, a formalism provides a mathematical basis for verification and simulation of a design. In a composite service, for example, a formalism provides techniques that allow designers to analyze if two services can be composed. An overview of formalisms that are used in model-driven service composition is given in the Languages for Service Composition section. A design approach prescribes a series of steps that have to be taken to construct a design. In this way, a design approach provides a structured way to construct a design by gradually introducing more detail into user requirements and current business operations until a level of detail is reached at which a design can be directly implemented. For an approach to service composition, this is the level of detail at which a design in a one-to-one fashion corresponds to a description that can be executed by a runtime environment. Such a description is a textual (typically XML-based) representation of the functional and nonfunctional properties of a service or service composition. The Languages for Service Composition section, Description Languages subsection explains some existing techniques for describing service compositions.

From existing description techniques in the area of service composition, we can derive that there are currently two ways to design a service composition, namely *choreography* or *orchestration*. A choreography differs from an orchestration with respect to where the logic that controls the interactions between the services involved should reside.

A choreography describes a collaboration between some enterprise services to achieve a common goal. Hence, it does not focus on a particular service but rather on a goal. Therefore, the control logic is distributed over the involved services and the choreography emerges as the services interact with each other. To design a choreography, we first describe the interactions that enterprise services have with each other to achieve their goal and then the relations that exist between these interactions. A choreography does not describe the actions that are performed internally by the service providers to realize their enterprise services. Figure 2 shows a typical example of a choreography. This example shows a collaboration that relates to buying an item.

An orchestration describes the behavior that a service provider implements to realize a service. Hence, it focuses on a particular service, and the control logic is centralized on the service provider of which we implement the behavior. To design an orchestration, we describe the interactions that the service provider has with other parties and the actions that the service provider performs internally to realize the service. An orchestration is meant to be executed by an orchestration engine, as will be explained in the Composite Service Execution subsection. Therefore, it is also called an executable process.

From these observations we can derive a set of basic concepts that are important in the design of service composition, regardless of whether a choreography — or an orchestration-oriented approach is chosen and of the description or design language that is

Figure 2. An example of a choreography

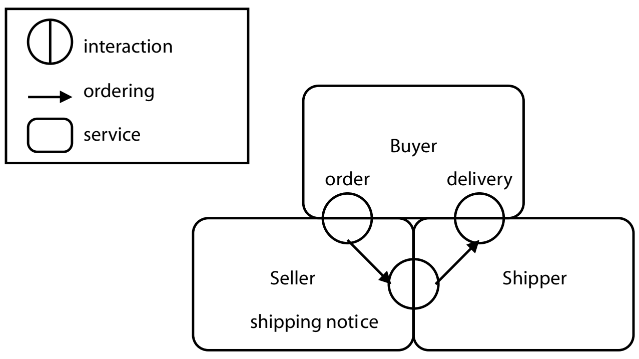
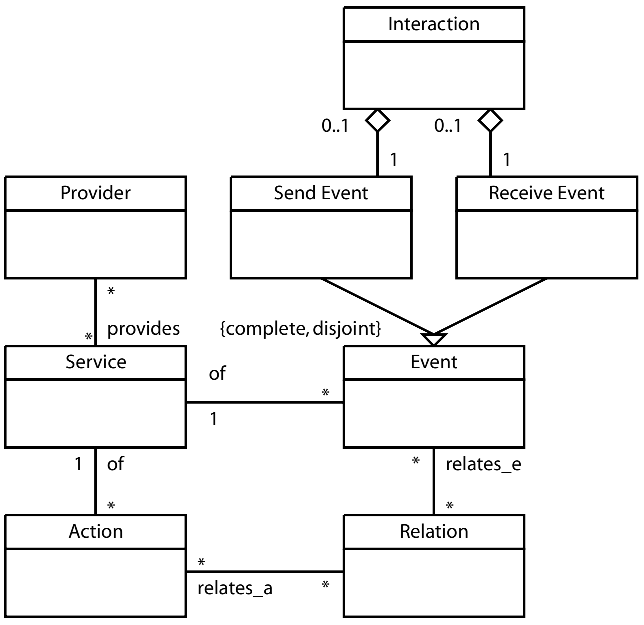


Figure 3. Basic design concepts for service composition design



used. Figure 3 shows a meta-model in which our basic concepts are represented. The figure shows that a service composition consists of a number of services that are provided by service providers. The same service can be provided more than once by different service providers (for example, a flight booking service can be provided by different airlines). A service consists of (internal) actions and events that are part of an interaction with other services. We claim that interactions are based on message passing because this is the basic mechanism for interaction that is used in the mainstream service description languages as they are presented in the Description Languages subsection. Hence, interactions consist of a send event and a receive event. Relations relate actions and interactions to each other. The kind of relation that is used (for example, flow relation, causal relation, state-based relation, and so forth) depends on the language that is used.

Composite Service Execution

The composite service execution engine is the runtime component of a service composition tool. It takes as input a composite service description and coordinates the execution of the composite service according to that description. At least two different execution models can be distinguished:

- **Centralized** (see, for example, Schuster, Georgakopoulos, Cichocki & Baker, 2000 and Casati & Shan, 2001). In this model, the responsibility for coordinating the execution of a composite service relies on a single "scheduler." This scheduler interacts with each of the component services by processing and dispatching messages. The internal architecture of the central scheduler is similar to that of a traditional workflow management system (van der Aalst & van Hee, 2002), except that the resources are all services rather than human actors, and there is no shared database through which information can be implicitly passed from one stakeholder to another. Instead, information must be explicitly passed through message exchanges.
- **Peer-to-Peer** (see, for example, Mecella, Parisi-Presicce & Pernici, 2002 and Benatallah, Sheng & Dumas, 2003). In this model, the responsibility for coordinating the executions of a composite service is distributed across the providers of the component services, which interact in a peer-to-peer way without routing messages through a central scheduler. The composite service execution environment therefore manifests itself in the form of a collection of inter-connected modules, which communicate through an agreed protocol. This execution model bears some similarities with distributed workflow execution models, such as those described in Muth, Wodtke, Weissenfels, Dittrich, and Weikum (1998) and Chen and Hsu (2002).

It is crucial that a mechanism is provided for monitoring the executions of a composite service. Indeed, being able to trace the execution of a composite service is crucial for metering, accounting, customer feedback, adaptation, and service improvement. The monitoring mechanism varies depending on the execution model. In the case of centralized execution, the central scheduler can maintain a database of execution traces. In the case of peer-to-peer execution, however, the information about composite service executions is disseminated across a number of distributed data sources hosted by the providers of the component services. Accordingly, it is necessary either to consolidate these distributed data sources periodically or to be able to answer queries on demand (Fauvet, Dumas & Benatallah, 2002).

A composite service can be linked to its component services either in a static or a dynamic manner. A link between a composite service and a component service is static when it is established at design time and cannot be changed without modifying the design of the composite service. A link with a component service is dynamic when a mechanism selects, at runtime, the actual service that will be invoked to perform a given step in the composite service execution. We call this approach *dynamic service selection*.

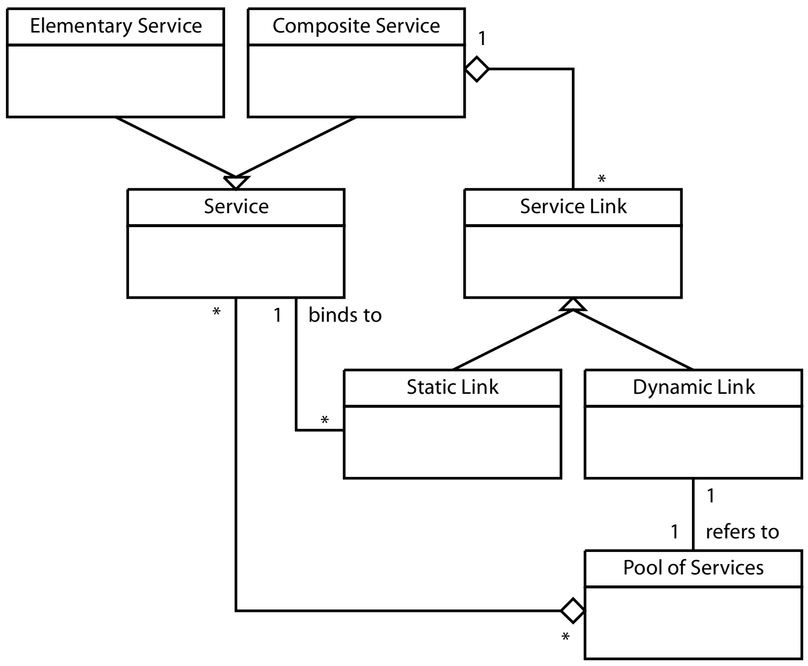
The pool of candidate services over which the dynamic selection occurs may be: (i) determined at design time; (ii) obtained by evaluating a given query over a registry (for example, UDDI registry); or (iii) obtained from an invocation to a brokering service. The selection itself is then performed based on a set of requirements and using a set of preferences expressed in the composite service description. These constraints and preferences may involve both functional attributes (that is, attributes describing the capabilities of the services) and nonfunctional attributes (for example, time, location, price, reliability, trust).

Once a service within the pool of candidate services is selected, it has to be invoked by the composite service. This implies that either all the candidate services for a given task of a composite service offer exactly the same interface (that is, the same set of operations and common constraints on their use) or that some late binding mechanism is used to "homogenize" the interfaces provided by all services so that at the end, the composite service can invoke any of these candidate services.

The CORBA Dynamic Invocation Interface (DII) is an example of a late binding mechanism. Another example from the area of inter-organizational workflow is provided by the CrossFlow system (Grefen, Aberer, Hoffner & Ludwig, 2000). In this system, a task in a workflow can be linked to a contract. When the task needs to be executed, a matchmaking facility attempts to find another workflow that complies with that contract. In the area of Web services, the Web Services Invocation Framework (WSIF) (Apache Web Services Project, 2003) has been developed for the purpose of enabling late binding of Web services.

Figure 4 represents the concepts that are described in this subsection in a meta-model.

Figure 4. Basic execution concepts for service composition design



Languages for Service Composition

The Generic Architecture section explained the need for different types of languages for describing service compositions from different viewpoints. In this section, we present some of these languages. We limit the discussion to description languages and formalisms, leaving aside design languages because there is not yet a widely accepted standard for such languages.

Description Languages

- **BPEL4WS.** The Business Process Execution Language for Web Services (BEA Systems, Microsoft, IBM & SAP, 2003) is a language with an XML-based syntax, supporting the specification of processes that involve operations provided by one or several Web services.

  BPEL4WS is intended to support the description of two types of processes: abstract and executable. An abstract process is a partially ordered set of message exchanges between a service and a client of this service. It describes the *behavioral interface* of a service without revealing its internal behavior. Using the terminology introduced in the previous section, an abstract process is a two-party choreography involving a service provider and a service requestor, described from the perspective of the provider.

  An executable process on the other hand, captures the internal behavior of a service in terms of the messages that it will exchange with other services and a set of internal data manipulation steps. An executable process is composed of a number of constituent activities, the partners involved in these activities, a set of internal variables, and a set of activities for handling faults and transactional rollbacks. Using the terminology of the previous section, a BPEL4WS executable process corresponds to an orchestration specification.

  BPEL4WS draws upon concepts developed in the area of workflow management. When compared to languages supported by existing workflow systems and to related standards (for example, XPDL, WSCI, and ebXML BPSS), it appears that BPEL4WS is relatively expressive (Wohed, van der Aalst, Dumas & ter Hofstede, 2003). In particular, the *pick* construct is not supported in many existing workflow languages. On the negative side, it can be said that BPEL4WS lacks orthogonality, in the sense that it has many constructs with overlapping scope (for example, the *switch* and *sequence* constructs overlap with the *control link* construct).

- **WSCI and BPML.** The Business Process Management initiative (BPMi) is an industry consortium aiming at contributing to the development of (service-oriented) process description standards. The consortium has published a specification for a service-oriented process description language called BPML (Business Process Modeling Language), similar in many ways to BPEL4WS. BPML draws on a previous standard called WSCI (Web Service Conversation Interface) developed by the stakeholders behind BPMi. WSCI integrates many of the constructs found in BPML and BPEL4WS (for example, sequence, choice, parallel execution, send/receive primitives, and so forth). However, it differs from them in its intent: while BPML is mainly intended for describing orchestrations, WSCI is intended for describing choreographies (see Composite Service Design subsection in the previous section for a discussion on this dichotomy). The strong commonalities between these languages suggest that orchestration and choreography correspond to two different (and complementary) viewpoints over the same class of models (that is, composite service models). As discussed earlier, BPEL4WS has been designed with the goal of capturing both orchestrations and two-party choreographies. Peltz (2003) discusses the relationships between WSCI, BPML, and BPEL4WS in more detail.

- **ebXML BPSS.** Electronic Business XML (ebXML) is a series of standards intended to provide an implementation platform for business-to-business collaborations. ebXML adopts a choreography-based approach to service composition. Specifically, a business collaboration is described as a set of Collaboration Protocol Profiles (CPP) (UN/CEFACT & OASIS, 2001a). A CPP describes, among other things, which part of a given business process a given partner is able to provide by referring to a role in a process specified using the Business Process Specification Schema (BPSS) (UN/CEFACT & OASIS, 2001b). A BPSS document specifies a number of transactions, the roles associated with these transactions, the flow of control and flow of documents between these transactions, and the document access rights for the involved documents. Control-flow relationships are described using guarded transitions (like in state machines) and fork/join operators.
- **WS-CDL.** The W3C Web Service Choreography Description Languages (WS-CDL) (W3C World Wide Web Consortium, 2002) is another ongoing standardization effort in the area of service composition. Like WSCI and ebXML, the intent of WS-CDL is to define a language for describing multiparty interaction scenarios (or choreographies), not necessarily for the purpose of executing them using a central scheduler but rather with the purpose of monitoring them and being able to detect deviations with respect to a given specification.
- **RosettaNet.** RosettaNet (RosettaNet, 2004) is an industry consortium, which has developed a series of standards for Business-to-Business (B2B) integration with an emphasis on supply chain management. Among others, RosettaNet defines a notion of Partner Interface Protocols (PIP), which enables the description of interactions between business processes deployed by multiple partners. The notion of PIP is related to the notion of service choreography and has influenced efforts in this area. For details about RosettaNet and its relationship to Web service standards, readers are referred to Bussler (2003).

Formalisms

In an attempt to provide a rigorous foundation to service composition and to enable the use of formal verification and simulation techniques, a number of formalisms for describing composite services have been proposed. One of the earliest proposals in this area is that of Cardelli and Davies (1999), who present an algebra for programming applications that access multiple Web resources (also called *services*). This algebra brings together operators inspired by process algebras (sequential execution, concurrent execution, and repetition) with operators capturing the unreliable nature of the Web (timeout, time limit, rate limit, stall, and fail). Basic services are described using the operator *url*, which attempts to fetch the resource associated with a given URL. Although the algebra is intended for manipulating Web pages, it could conceivably be extended to take into account the richer structural and behavioral descriptions of Web services.

Various authors have advocated the use of Petri nets as a formal foundation for modeling composite services or for defining formal semantics for service composition languages. The VISPO project (Mecella et al., 2002) has advocated the use of Petri nets to model the control flow aspects of composite services. Van der Aalst (2003) examines a number of proposed standards for service composition in terms of a collection of workflow patterns and notes that these proposed standards would benefit from having a formal semantics defined in terms of established formalisms such as Petri nets. Finally, Narayanan and McIlraith (2002) present DAML-S, a language that supports the description of composite services, and defines a mapping from the process-oriented subset of DAML-S to Petri nets.

More recently, Bultan, Fu, Hull, and Su (2003) adopt Mealy machines (a category of communicating automata with queues) to describe the interactions (also called *conversations*) between aggregated services. Each service participating in an aggregation is described as a Mealy machine, which consumes events from a queue and dispatches events to the queues of the other services in the aggregation. The authors study the expressive power of the resulting formalism, measured in terms of the set of traces (that is, sequences of events) that can be recognized by an aggregation of Mealy machines.

There is not yet a widely accepted formal foundation for service composition. It appears that Petri nets, process algebras, and state machines are suitable for capturing at least certain aspects of service composition. Ultimately, however, for a given formalism to be adopted in this area, it is necessary that its benefits are tangible (for example, availability of analysis and simulation tools) and that full mappings between this formalism and concrete modeling and description languages are provided.

Platforms for Composite Service Execution

The previous section explained the structure of an execution environment for composite services. In this section, we review existing implementations that serve as execution environments. We first provide an overview of some research prototypes before looking more closely at the implementations provided by major vendors: IBM, BEA, and Microsoft.

Research Prototypes

CMI (Collaboration Management Infrastructure) (Schuster et al., 2000) provides an architecture for interenterprise services. It uses state machines to describe the behavior of composite services. The concept of placeholder is used to enable the dynamic selection of services. A placeholder is a set of services (identified at runtime) and a method for selecting a service given a set of parameters.

eFlow (Casati & Shan, 2001) is a platform that supports the specification, enactment, and management of composite services. eFlow uses graph-based model in which the nodes denote invocations to service operations, and the edges denote control-flow dependencies. A composite service is modeled by a graph that defines the order of execution among the nodes in the process. The definition of a service node contains a search recipe represented in a query language. When a service node is invoked, a search recipe is executed to select a reference to a specific service. Once a service is selected by the search recipe, the eFlow execution engine is responsible for performing the dynamic binding using metadata that it stores in the service repository.

CrossFlow (Grefen et al., 2000) features the concept of contracts for services cooperation. When a partner wants to publish a collaboration, it uses its contract manager to send a contract template to matchmaking engine. When a consumer wants to outsource a service, it uses a contract template to search for relevant services. Based on the specifications in the contract, a service enactment structure is set up.

SELF-SERV (compoSing wEb accessibLe inFormation and buSiness services) (Benatallah, Dumas, Sheng & Ngu, 2002) specifies composite services using statecharts. Furthermore, SELF-SERV proposes a peer-to-peer model for orchestrating a composite service execution in which the control and data-flow dependencies encoded in a composite service definition are enforced through software components located in the sites of the providers participating in a composition. SELF-SERV refines the concepts of search recipe and placeholder introduced by e-Flow and CMI by proposing the concept of community. A community is an abstract definition of a service capability with a set of policies for (i) managing membership in the community and (ii) selecting at runtime the service that will execute a given service invocation on behalf of the community. Policies for runtime selection of services are formulated using multiattribute value functions. A community is also responsible for performing the dynamic binding of the selected Web service, thereby acting as a dynamic service selector.

DySCo (Piccinelli, Finkelstein & Lane Williams, 2003) is another service-oriented workflow infrastructure, which supports the definition and enactment of dynamic service interactions. DySCo adopts a traditional workflow approach, except with respect to the definition of a task. Instead of corresponding to an activity involving a number of resources, a task in DySCo corresponds to an interaction step between services. In addition, DySCo supports the dynamic reconfiguration of service interactions by allowing a task to be decomposed at runtime into a more complicated structure. For example, a document mailing task in a service-based workflow can be decomposed into two tasks: a document printing task and a document posting task, which can then be assigned to different providers.

Commercial Tools

Typically, a tool claiming to support the Web services stack would minimally provide an API in one or more programming languages (for example, Java) for generating and/or processing SOAP messages. Some tools would go further by supporting tasks such as: (i) generating WSDL descriptions from modules, packages, or classes (for example, from a Java class file); (ii) editing WSDL descriptions through a graphical interface; and/or (iii) extracting information contained in WSDL files in order to dynamically generate stubs and skeletons that provide transparent communication between Web service requesters and providers.

Most tools also provide support for UDDI (both for setting up a registry and for connecting to an existing registry). Few tools currently support composite service description languages, and when they do, they typically only support a subset of these languages.

- **IBM WebSphere.** WebSphere is a family of IBM products for enabling B2B interactions. The application server is the cornerstone of WebSphere. It aims at providing database and backend integration as well as security and performance capability (for example, workload management). The WebSphere application server Advanced Edition adds support for J2EE and CORBA. The advanced edition integrates support for key Web service standards such as SOAP, UDDI, and WSDL. Additionally, it provides distributed transaction support for major database systems. Other products make up the WebSphere platform. These include WebSphere Business Components, WebSphere Commerce, and WebSphere MQ Family. The WebSphere Business Components provides prebuilt and tested components. WebSphere Commerce provides mechanisms for building B2B sites. WebSphere MQ Family is a family of message-oriented middleware products.
- **BEA WebLogic Integrator.** BEA WebLogic Integrator is one of the cornerstones of the BEA WebLogic e-Business Platform. It is built on top of a J2EE compliant application server and J2EE connector architecture and supports current Web service standards such as SOAP, UDDI, and WSDL. It is composed of four major modules:
  - The Application Server, which provides the infrastructure and functionalities for developing and deploying multitier distributed applications as EJB components.
  - The Application Integration Server, which leverages the J2EE connector architecture to simplify integration with existing enterprise applications, such as SAP R/3 and PeopleSoft.
  - The Business Process Management System, which provides a design tool and execution engine for business processes in BPEL4WS.
  - The B2B integration manages interactions with external business processes.

- **Microsoft Web Services Support.** Support for Web services is one of the key aspects of the .Net product series. In particular, ASP.Net provides a programming model for exposing applications as Web services. Briefly, the skeleton of a Web service is encoded as an ASMX file (proprietary Microsoft format), which can be interpreted by the Internet Information Server (IIS) in order to process incoming SOAP calls for the service and generate SOAP responses and faults. A WSDL description and a test page are also automatically generated from the ASMX file. Another Microsoft product, which provides support for Web services, is BizTalk: a middleware platform for Enterprise Application and B2B Integration. Applications in BizTalk are integrated based on an XML message-oriented paradigm. Part of the BizTalk suite is the BizTalk Orchestration Engine, which implements XLANG, a precursor of BPEL4WS. Developers can define processes using a graphical interface and export them as XLANG descriptions, which are then fed into the runtime engine.

Trends Relevant to (Web) Service Composition

While much of the work to date has focused on standards for announcing, discovering, and invoking Web services, there are other significant developments happening in Web services. In this section, we overview some of the developments related to conversation-driven composition, semantic Web services, and wireless Web services (also known as M-services), focusing on those aspects relevant to service composition.

Conversation-Driven Composition

A conversation is a consistent exchange of messages between participants involved in joint operations. A conversation succeeds when what was expected from that conversation in terms of outcome has been achieved. Further, a conversation fails when the conversation faced difficulties (for example, communication-medium disconnected) or did not achieve what was expected.

The use of conversations helps in defining composite services at runtime instead of design time. When a Web service is being executed, it has at the same time to initiate conversations with the Web services that are due for execution. The purpose of these conversations is twofold (Maamar, Benatallah & Mansoor, 2003): invite the Web services to join the composition process and ensure that the Web services are ready for execution in case they accept the invitation. Furthermore, conversations between Web services allow addressing of the composability problem. Medjahed, Rezgui, Bouguettaya, and Ouzzani (2003) note that an issue when defining a composite service is to check if the Web services can actually work together at the information level. Mapping operations of the parameters exchanged between Web services may be required. Ensuring the composability of Web services can be completed using ontologies and conversations. Web services engage in conversations to agree on which ontology to use, what/how/when to exchange, and what to expect from an exchange.

The Web Services Conversation Language is an initiative on the integration of conversations into Web services. This language describes the structure of documents that a Web service is supposed to receive and produce and the order in which the exchange of these documents will occur. The conversation component to embed a Web service is mainly a means for describing the operations that a Web service supports (for example, clients have to log in first before they can check the catalogue).

Ardissono, Goy, and Petrone (2003) observed that current Web services communication standards support simple interactions and are mostly structured as question-answer pairs. These limitations hinder the possibility of expressing complex situations that require more that two turns of interactions (for example, propose/counter-propose/accept-reject). In addition, Ardissono et al. (2003) worked on a conversational model that aims at supporting complex interactions between clients and Web services, where several messages are exchanged before a Web service is completed.

It is stated that the full capacity of Web services as an integration platform will be reached only when applications and business processes integrate their complex interactions by using a standard process integration model such as BPEL4WS. While the orchestration of Web services is a core component to any Web services integration effort, the use of conversations gives more "freedom" to Web services to decide if they will take part in this orchestration. Conversations are more than just combining components; they promote the autonomy of components that act and react according to their environment (Hanson, Nandi & Levine, 2002).

Semantic Web Services

Another major trend is the integration of semantics into Web services. Heflin and Huhns (2003) argue that the goal driving the semantic Web is to automate Web-document processing. The semantic Web aims at improving the technology that organizes, searches, integrates, and evolves Web-accessible resources (for example, documents, data). This requires the use of rich and machine-understandable abstractions to represent the resource semantics.

One of the core components to the widespread acceptance of the semantic Web is the development of ontologies that specify standard terms and machine-readable definitions. Although there is no consensus yet on what an ontology is, most researchers in the field of knowledge representation consider a taxonomy of terms and the mechanisms for expressing the terms and their relationships. Samples of markup language for publishing and sharing ontologies on the 3W include RDF (Resource Description Framework), DAML+OIL (DARPA Agent Markup Language + Ontology Inference Layer), and OWL (Web Ontology Language) (W3C World Wide Web Consortium, 2001).

By combining efforts of Web services and semantic Web communities, it is expected that new foundations and mechanisms for enabling automated discovery, access, combination, and management for the benefit of semantic Web services will be developed.

Paolucci and Sycara (2003) note that the semantic Web provides tools for explicit markup of Web content, whereas Web services could create a network of programs (that is, software agents) that produce and consume information, enabling automated business interactions. There exist various initiatives in the field of semantic Web services such as DAML-S (DARPA Agent Markup Language for Services) (DAML-S Consortium, 2004), WSMF (Web Services Modeling Framework) (Fensel & Bussler, 2002), and METEOR-S (Managing End-To-End OpeRations-Semantic Web Services and Processes) (Sivashanmugam, Verma, Sheth & Miller, 2003).

Wireless Web Services

Besides the Web expansion, a development occurring in the field of wireless and mobile technologies is witnessed (Wieland, 2003). Telecom companies are offering new services and opportunities to customers over mobile devices. The next stage (if we are not already in it), is to allow users to remotely enact Web services from mobile devices (Maamar & Mansoor, 2003).

While Web services provisioning is an active area of research and development (Benatallah & Casati, 2002), little has been done to date regarding their provisioning in wireless environments. This is due to different obstacles including throughput and connectivity of wireless networks, limited computing resources of mobile devices, and risks of communication channel disconnections. In addition, businesses that are eager to engage in wireless Web services activities are facing technical, legal, and organizational challenges. To optimize Web services provisioning in wireless environments, important issues need to be tackled first:

Context-sensitive Web services selection. In addition to traditional criteria such as monetary cost and execution time, the selection of services should consider, on the one hand, the location of requesters and, on the other hand, the capabilities of the computing resources on which these services will be deployed (for example, processing capacity, bandwidth). This calls for context-aware service selection policies that enable a system to adapt itself to computing and user requirements.

Handling disconnections during Web services execution. In a wireless environment, disconnections are frequent. It is noted that to cope with disconnection issues during a service delivery, software agent-based service composition middleware architectures are deemed appropriate as proposed in Maamar, Sheng, and Benatallah (2004).

Conclusion

Web services promise to revolutionize the way in which applications interact over the Web. However, the underlying technology is still in a relatively early stage of development and adoption. While the core standards such as XML, SOAP, and WSDL are relatively stable and are supported in various ways by a number of tools, standardization efforts in key areas such as security, reliability, policy description, and composition are still underway, and the tools supporting these emerging standards are still evolving. In addition (or perhaps as a result of this), relatively few production-level Web services have been deployed and are being used in practice. To some extent, these difficulties can be explained by the fact that businesses have spent considerable resources in the last few years to expose their functionality as interactive Web applications. As a result, they are reluctant to invest more to move this functionality into Web services until the benefits of this move are clear. It will probably take another two years before the technology reaches the level of maturity necessary to trigger a widespread adoption. In the meantime, it is important that middleware platform developers integrate the numerous facets of Web services into their products (for example, facilitating the use of message-oriented middleware for Web service development), while researchers advance the state of the art in challenging issues such as Web service delivery in mobile environments, QoS-driven selection of services, and manipulation of semantic-level service descriptions.

References

Alonso, G., Casati, F., Kuno, H., & Machiraju, V. (2003). *Web services: Concepts, architectures and applications*. Berlin: Springer-Verlag.

Apache Web Services Project. (2003). Web services invocation framework (WSIF). Retrieved August 6, 2004: *http://ws.apache.org/wsif/*

Ardissono, L., Goy, A., & Petrone, G. (2003, July 14-18). *Enabling conversations with web services.* Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), Melbourne, Australia.

BEA Systems, Microsoft, IBM & SAP. (2003). Business process execution language for Web services (BPEL4WS). Retrieved August 6, 2004: *ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf*

Benatallah, B., & Casati, F. (2002). Introduction to special issue on Web services. *Distributed and Parallel Databases: An International Journal, 12*(2-3).

Benatallah, B., Dumas, M., Sheng, Q., & Ngu, A. (2002, February 26-March 1). *Declarative composition and peer-to-peer provisioning of dynamic Web services.* Proceedings of the 18th IEEE International Conference on Data Engineering (ICDE), San Jose, CA.

Benatallah, B., Sheng, Q., & Dumas, M. (2003). The SELF-SERV environment for Web services composition. *IEEE Internet Computing, 7*(1), 40-48.

Bultan, T., Fu, X., Hull, R., & Su, J. (2003, May 20-24). *Conversation specification: A new approach to design and analysis of e-service composition.* Proceedings of the 12th International Conference on the World Wide Web (WWW'03) (pp. 403-410), Budapest, Hungary.

Bussler, C. (2003). *B2B integration: Concepts and architecture*. Berlin: Springer-Verlag.

Cardelli, L., & Davies, R. (1999). Service combinators for Web computing. *IEEE Transactions on Software Engineering*, *25*(3), 309-316.

Casati, F., & Shan, M.-C. (2001). Dynamic and adaptive composition of e-services. *Information Systems*, *26*(3), 143-162.

Chen, Q., & Hsu, M. (2002, October 28-November 1). *CPM revisited – An architecture comparison.* Proceedings of the Confederated International Conferences CoopIS, DOA, and ODBASE (pp. 72-90), Irvine, CA.

Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., & Weerawarana, S. (2002). Unraveling the Web services web: An introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, *6*(2), 86-93.

DAML-S Consortium. (2004). DAML services. Retrieved August 6, 2004: *http://www.daml.org/services*

Fauvet, M.-C., Dumas, M., & Benatallah, B. (2002, October 28-November 1). *Collecting and querying distributed traces of composite service executions.* Proceedings of the Confederated International Conferences CoopIS, DOA, and ODBASE (pp. 373-390), Irvine, CA.

Fensel, D., & Bussler, C. (2002). The Web services modeling framework WSMF. *Electronic Commerce Research and Applications*, *1*(2), 113-137.

Grefen, P., Aberer, K., Hoffner, Y., & Ludwig, H. (2000). CrossFlow: Cross-organizational workflow management in dynamic virtual enterprises. *International Journal of Computer Systems Science & Engineering*, *15*(5), 277-290.

Hanson, J. E., Nandi, P., & Levine, D. W. (2002, June 24-27). *Conversation-enabled Web services for agents and e-business.* Proceedings of the International Conference on Internet Computing (IC), Las Vegas.

Heflin, J., & Huhns, M. (2003). The Zen of the Web. *IEEE Internet Computing*, *7*(5).

Maamar, Z., Benatallah, B., & Mansoor, W. (2003, May 20-24). *Service chart diagrams: Description and application.* Proceedings of the 12th International Conference on the World Wide Web (WWW'03), Budapest, Hungary.

Maamar, Z., & Mansoor, W. (2003). Design and development of a software agent-based and mobile service-oriented environment. *e-Service Journal*, *2*(3).

Maamar, Z., Sheng, Q.Z., & Benatallah, B. (2004). On composite web services provisioning in an environment of fixed and mobile computing resources. *Information Technology and Management Journal, 5*(3).

Mecella, M., Parisi-Presicce, F., & Pernici, B. (2002, August 23-24). *Modeling e-service orchestration through Petri nets.* Proceedings of the 3rd International Workshop on Technologies for E-Services (TES) (pp. 38-47), Hong Kong.

Medjahed, B., Benatallah, B., Bouguettaya, A., Ngu, A., & Elmagarmid, A. (2003). Business-to-business interactions: Issues and enabling technologies. *The VLDB Journal, 12*(1), 59-85.

Medjahed, B., Rezgui, A., Bouguettaya, A., & Ouzzani, M. (2003). Infrastructure for e-government Web services. *IEEE Internet Computing*, *7*(1).

Muth, P., Wodtke, D., Weissenfels, J., Dittrich, A., & Weikum, G. (1998). From centralized workflow specification to distributed workflow execution. *Journal of Intelligent Information Systems*, *10*(2).

Narayanan, S., & McIlraith, S. (2002, May 7-11). *Simulation, verification and automated composition of Web services.* Proceedings of the 11th International Conference on the World Wide Web (pp. 77-88), Honolulu.

Paolucci, M., & Sycara, K. (2003). Autonomous semantic Web services. *IEEE Internet Computing*, *7*(5).

Peltz, C. (2003). Web services orchestration and choreography. *IEEE Computer*, *36*(8), 46-52.

Piccinelli, G., Finkelstein, A., & Lane Williams, S. (2003, September 1-6). *Service-oriented workflow: The DySCo framework.* Proceedings of the 29th EUROMICRO Conference (pp. 291-297), Belek-Antalya, Turkey.

RosettaNet (2004). RosettaNet home page. Retrieved August 6, 2004: *http://www.rosettanet.org*

Schuster, H., Georgakopoulos, D., Cichocki, A., & Baker, D. (2000, June 5-9). *Modeling and composing service-based and reference process-based multi-enterprise processes.* Proceedings of the 12th International Conference on Advanced Information Systems Engineering (CAiSE) (pp. 247–263), Stockholm, Sweden.

Sivashanmugam, K., Verma, K., Sheth, A., & Miller, J. (2003, October 20-23). *Adding semantics to web services standards.* Proceedings of the 2nd International Semantic Web Conference (ISWC), Sanibel Island, FL.

UN/CEFACT, & OASIS (2001a). Collaboration-protocol profile and agreement specification. Retrieved August 6, 2004: *http://www.ebxml.org/specs/ebCCP.pdf*

UN/CEFACT, & OASIS (2001b). ebXML business process specification schema. Retrieved August 6, 3004: *http://www.ebxml.org/specs/ebBPSS.pdf*

van der Aalst, W. (2003). Don't go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems*, *18*(1).

van der Aalst, W., & van Hee, K. (2002). *Workflow management: Models, methods, and systems*. Cambridge, MA: MIT Press.

W3C World Wide Web Consortium. (2001). Semantic Web activity. Retrieved August 6, 2004: *http://www.w3.org/2001/sw*

W3C World Wide Web Consortium. (2002). Web services choreography working group. Retrieved August 6, 2004: *http://www.w3.org/2002/ws/chor*

Wieland, K. (2003). The long road to 3G. *International Telecommunications Magazine*, *37*(2).

Wohed, P., van der Aalst, W., Dumas, M., & ter Hofstede, A. (2003, October 13-16). *Analysis of Web services composition languages: The case of BPEL4WS.* Proceedings of the 22nd International Conference on Conceptual Modeling (ER). Chicago.