Service Oriented Architecture: Overview and Directions

Boualem Benatallah and Hamid R. Motahari Nezhad

School of Computer Science and Engineering
The University of New South Wales
Australia
{boualem,hamidm}@cse.unsw.edu.au

1 Introduction

The push toward business automation, motivated by opportunities in terms of cost savings and higher quality, more reliable executions, has generated the need for integrating the different applications. Integration has been one of the main drivers in the software market during the late nineties and into the new millennium. It has led to a large body of research and development in areas such as data integration [26], software components integration, enterprise information integration (EII), enterprise applications integration (EAI), and recently service integration and composition [2, 11, 16, 12].

Service oriented architectures (SOAs) provide an architectural paradigm and abstractions that allow to simplify integration [2, 21]. There a number of technologies available to realize SOA. Among them, Web services and the set of related specifications (referred to as WS-* family), and also services that are built following the REST (REspresentation State Transfer) architecture [8] (called RESTful services) are gaining the momentum for integration at the data level.

One of the main facilitators of integration in WS-* approach is standardization. Standardization is a key to simplifying interoperability: instead of having to interact with heterogeneous systems, each with its own transport protocol, data format, interaction protocol, and the like, applications can interact with systems that are much more homogeneous. More specifically, Web services standards foster support of loosely coupled and decentralized interactions mainly at the application level. The main feature of RESTful approach is the simplicity of service development and usage. This architectural style has been adopted in the offering of data services [4, 1] which is a major advance in data-level integration. AJAX [9], which is an enabler of an ad-hoc service composition approaches known as *mashups* [17], is also based on REST. Mashup applications enable integration at the presentation level. This refers to integration of graphical user interfaces (GUIs) of applications.

In this chapter, we briefly survey the different specifications and approaches in SOA and evaluate them in terms of their contributions to integration. We propose a conceptual framework for understanding the integration problem as well for analyzing existing solutions. We believe that viewing the different approaches to interoperability in the context of this framework will make it easier

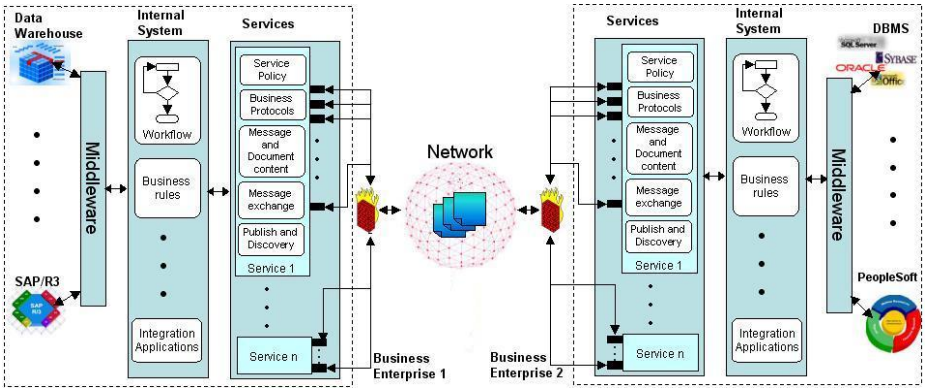

Fig. 1. Integration scenarios: EAI (enterprise application integration) and B2B (Business-to-Business)

to identify the commonalities and contrasts of existing standards and specifications, discover gaps, and better leverage existing standards to provide automated support to Web service interoperability.

In the following, in Section 2, we present the conceptual framework in terms of integration layers going from low level, which are horizontal, to higher levels, which may not be needed in all integration scenarios (Section 2.2). Next, we provide an overview of integration solutions before SOA in the context of proposed architecture (Section 2.3). In Section 3, we introduce the main existing technologies for realization of SOA (Section 3.1). Then, we use the proposed integration layers to evaluate SOA realization approaches (Section 3.2). In Section 4, we outline future directions in SOA to further help developers and users in simplifying the problem of integration, and conclude this chapter.

2 Software Integration

2.1 Motivating Example

As an example, consider the two business enterprises depicted in Figure 1, exposing their functionalities as services. Integration is important in two different scenarios: integrating internal systems of each enterprise (referred to as "enterprise application integration" (EAI), as well), and integration with external entities (referred to as "business-to-business integration" (B2B), as well). In the internal of an enterprise, there is a need to integrate data and applications related to various systems. For instance, if these enterprises work in the domain of procurement and sales in a supermarket chain, then they may maintain a database for storing procurement data and a database for storing inventory and sales data. They may also need a data warehouse (DW) for storing historical sales transactions, which needs to collect and integrate data from these two data sources. Each of these enterprises may also implement a business process for

fulfilling its business objectives, which describes how, e.g., orders are processed from the time that they arrive up to the time that goods are shipped, in terms of data and control flows.

From the perspective of external interactions, the pre-requisite is that services on each enterprise can communicate seamlessly (e.g., exchange messages) with those of its partner. In addition, there may be a need for properties such as security and guaranteed delivery. Furthermore, advanced features such transactions may be needed in the interactions between the two enterprises. Moreover, they should be able to understand the content of exchanged messages. The developers of the client enterprise also need to understand the order in which messages are expected by the partner services, which is captured in the business protocol of the services. Similarly, policies that govern the interactions between services should be known to service clients. Finally, developers in each enterprise may need to integrate results retuned from partner services, e.g., the current location of a shipment, with other external services, e.g., Google map, to visualize it at the presentation level.

In the following, we use the above description of requirements to present different layers of integration.

2.2 Integration Layers

By analogy with computer networks, we believe it is useful to study the integration in terms of layers, which address various parts of problem at different level of abstractions (Figure 2). Typically, the structuring in layers goes from lower level layers, which are more horizontal (needed by most or all interactions), to higher layers, which build on top on the lower ones and may or may not be needed depending on the application.

Communication layer. The first step for any application to interact to each other is to be able to exchange information. This is mainly achieved through the definition and using a protocol for transporting information, regardless of the syntax and semantics of the information content. Examples of such protocols are HTTP for transforming information on the Internet, IIOP in CORBA, and VAN in EDI standards [16].

Data layer. The integration at this layer means that the applications should seamlessly understand the content of data (documents and messages) that are exchanged between them. The interoperability issues at this layer occur in the syntax, structure and semantics of the data elements in the exchanged information. Integration at this layer is mainly achieved through offering mediators and languages (e.g., ETL) for transformation and mapping to convert data from one format to another. Although much progress has been made to facilitate data-level interoperation, however, still there is no silver bullet solution [26]. Given the current advances, users still play a major role in identifying the mismatches and developing their mappings and transformation.

Business logic layer. Integration at this layer refers to integrating standalone applications with defined interfaces (APIs), behavioral constraints, and

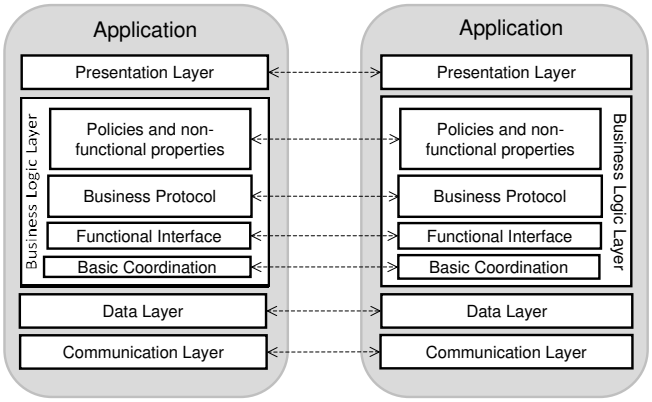

Fig. 2. Application integration layers

also non-functional constraints. Integration at this layer can be divided into the following sub-layers:

- *Basic coordination*. This layer is concerned with requirements and properties related to the exchange of a set of message among two or more partners. For example in both EAI and B2B scenarios, two services may need to coordinate to provide atomicity based on 2-Phase commit. Other examples of specifications at this layer are federated security management specifications. We consider this kind of coordination to be horizontal, meaning that such coordination are generally useful and can be applied in many business scenarios, and that is why we consider it at a lower level of abstraction in the business logic layer.
- *Functional interfaces*. The interface of a service declares the set of operations (messages) that are supported by the application. For example, a procurement service provides operations to lodge an order, and track its progress. As another example, a data service provides operations to access and manipulate data. The integration at this layer may imply finding correspondences and the mappings between the signatures of operations of the two services to be integrated.
- *Business protocol*. The business protocol gives the definition of the allowed operation invocation (or message exchange) sequences. Heterogeneities between business protocols can arise due to different message ordering constraints, or due to messages that one service expects (sends) but that the interacting partner is not prepared to send (receive). For example, a service may expect an acknowledgment in response to a sent message, while the partner does not issue such message.
- *Policies and non-functional properties*. The definition of an application may include policies (e.g., privacy policies) and other non-functional properties (e.g., QoS descriptions such as response time) that are useful for partners to understand if they can/want interact with the application. The interoper-

ability issues at this layer can be categorized into two classes: in expressing policies, in which case they are similar to those of data layer. For example, two applications may declare conceptually equivalent policy assertions, but using different syntax (i.e., element names), structure (i.e., element type and values) and semantics. The other class of interoperation issues refers to differences between the policies of two applications. For instance, differences in the offered/expected quality of service, e.g., response time, price, etc. Resolution of mismatches of this type may require negotiation and making agreements between applications.

Presentation layer. The integration in this layer refers to constructing applications by integrating components at the graphical user interface (UI) level [5]. UI-level integration fosters integration at a higher level of abstraction, where graphical representations of components are composed to build a new application. Integration issues at this layer include definition of a language and model for representation of components so that the integration is facilitated [5].

2.3 Integration Technologies before SOA

In this section, we give a brief overview of main integration technologies prior to Web services, and other realization of SOA.

2.3.1 Data integration

Data integration has been subject of research for many years most notably in the context of databases [13, 26]. The goal of data integration systems is to build applications by integrating heterogeneous data sources. Data integration systems have three elements: *source schema*, *mediated (target) schema*, and *the mapping* between them. Source schema refer to the data model of data sources to be integrated, mediated schema is the view of the integrated system from the existing data sources, and the mapping provide mechanisms for transforming queries and data from the integrated systems to those of data sources. A closely related area in this context is the *schema mapping* that aims at providing automated assistance for mapping schema definition of one data source into another [23]. These provide techniques for identifying syntactic, structural and semantic heterogeneities between schemas.

Note that in data integration systems, little cooperation from the component applications is needed, as one can always tap into the applications databases, e.g., by the means of SQL queries. The drawback of this approach is that it requires a significant effort to understand the data models and to maintain the mediated schema in the wake of changes in the data sources. In data integration systems, the integration is achieved through building new applications through composition of data sources at the data layer, and integration at the communication layer is achieved through tight coupling with integrated data sources.

2.3.2 Business logic integration

The integration of applications at the business logic level has been thoroughly studied in the last thirty years giving rise to technologies such as remote procedure calls (PRCs), object brokers (such as DCOM and CORBA), message brokers, electronic data interchange (EDI) and also standard specifications such as RosettaNet [2]. We can broadly categorize exiting solutions into RPC-based and message-oriented approaches.

Examples of RPC-based approaches include DCOM, Java RMI and CORBA, which enable calling operations on remote interfaces and so to integrate applications. They provide mechanisms for communication level integration, as well, but leave the data-level integration to approaches in data integration systems. On the other hand, message-oriented approaches such as EDI and RosettaNet target integration at the business process (business protocol) level through standardization. EDI provides VPN network and associated protocols for integration at the communication layer, and proposes to address data level issues through offering standardized business document formats. RosettaNet mainly provides specifications for integration at the business protocol level between applications. Finally, message-oriented middleware, suited for EAI scenarios, fosters integration through establishing a shared communication medium between parties, and the development of adapters (see [2, 16, 12] for a comparative study of these approaches).

3 Service Oriented Architecture

Service Oriented Architecture (SOA) is an architectural style that provides guidelines on how services are described, discovered and used [2, 21]. The purpose of this architecture is to address the requirements of application development for distributed information systems, which are loosely-coupled and potentially heterogeneous. In SOA, software applications are packaged as "services". Services are defined to be standards-based, platform- and protocol-independent to address interactions in heterogeneous environments. In the following, we give on overview of main realization of SOA and compare them in the context of proposed integration layers.

3.1 SOA Realization Technologies

Currently, there are four main approaches in SOA that provide specifications and standards for interoperation among services: *the WS-* family*, *ebXML*[1], *semantic Web services*[2], and *REpresentational State Ttansfer (REST)-ful services*.

The WS in WS-* family stands for "Web Services". Web services have become the preferred implementation technology for realizing the SOA paradigm. Web services rely, conceptually, on SOA, and, technologically, on open standard specifications and protocols. WS-* specifications are a group of standards

[1] http://www.ebXML.org
[2] http://www.daml.org/services

mainly proposed by industrial software vendors that develop specifications in an incremental and modular manner: specifications are introduced in a bottom-up fashion where the basic building blocks are simple, horizontal specifications. The specifications stack is gradually extended, with specifications at a higher level of abstractions built on top of more foundational ones.

ebXML (Electronic Business XML) is a joint initiative of the United Nations (UN/CEFACT) and OASIS[3] as a global electronic business standard. ebXML provides a framework for business-to-business integration and introduces a suite of specifications that enable businesses to locate their partners and conduct business based on a collaborative business process. It takes a top-down approach by allowing collaborations between partners to come up with a mutually negotiated agreement at a higher level, i.e., business process and contracts, and then working down towards how to exchange concrete messages. The technical architecture of ebXML provides a set of specifications for following fundamental components: (i) business process specification schema (BPSS) provides a framework to support execution of business collaborations consisting of business transactions, (ii) messaging services and security (ebMS), (iii) collaboration protocol profile and agreements (CPP/A), and (v) core components.

The semantic web services (SWS) aims to provide Web services with a rich semantic description of capabilities and contents in unambiguous and computer-interpretable languages to improve the quality and robustness of activities in the lifecycle of Web services including service discovery and invocation, automated composition, negotiation and contracting, enactment, monitoring and recovery [15, 14, 3]. Current efforts in this area can be organized into two categories, both of which assume using of a shared ontology between trading partners: (i) bringing semantic to Web services by defining and using semantic Web markup languages such as OWL-S [15], or WSMO [3], and (ii) incorporating semantic information by annotating messages and operations (supported by ontologies) of Web service specifications such as WSDL using their extensibility points and offering specifications such as WSDL-S [14].

The key component of the first category is using a language for the description of Web services. OWL-S (formerly known as DAML-S) is an OWL-based ontology for Web services in this category. OWL-S consists of three interrelated subontologies, known as the *serviceProfile*, *serviceModel*, and *serviceGrounding*. The serviceProfile expresses what a service does in terms of functional and non-functional properties, and its role is similar to CPP in ebXML-based approach. The serviceModel describes how a service works in terms of the workflow and possible execution paths of the service. The serviceGrounding maps the abstract constructs of the process model onto concrete specifications of message format and protocols. WSMO is another proposal in this area, which focuses more on providing a framework for developing semantic Web services. For comparison of OWL-S and WSMO refer to [22]. Here, we only discuss WSDL-S and OWL-S approach, which is currently more mature, compared to WSMO.

[3] http://www.oasis-open.org

REST is an architectural style that identifies how resources in a network, and specially World Wide Web, are defined, addressed and can be accessed [8]. In this architecture, applications in the network are modeled as a set of resources, which are uniquely addressable using a URI. Each resource also supports a constrained set of well-defined operations, and a constraint set of content types, e.g., XML, HTML, CSV, text, etc. REST adopts HTTP protocol for communication between resources, and therefore, the core operations of REST, i.e., GET, POST, PUT, and DELETE are those of HTTP. REST promotes a client-server, stateless and layered architecture. Due to its simplicity, which makes it scalable to the Internet, it has been adopted for implementing services (such services are called RESTful services). A client application that interacts with a resource (service) should know its URI address and can request to execute one of the core REST operations on the resource. The client should also know the data format of the output data. Therefore, the client developer has to read the documentation of the service to make it work with the service, or the data format may be shipped along with the message content.

The main differences between RESTful services with WS-*, which using SOAP on top of HTTP to provide additional functionalities, include: (i) RESTful services rely on a small set of domain-independent operations (e.g., GET to retrieve a representation of resources and PUT to update resources), while in SOAP-based services operations are defined in a domain-specific manner (e.g., a procurement service may offer a createPurchaseOrder operation), and (ii) REST works based on currently used Web standards such as HTTP and SSL. However, SOAP-based approach proposes a suite of extensible specifications to enable advanced functionalities such as reliable messaging, message-level and federated security and coordination, etc. RESTful services have been widely adopted for offering a significant number of simple services over the Internet, and they have gained remarkable popularity among developers. For instance, in Amazon Web services, the usage of their RESTful services far exceeds using its SOAP-based Web services although developers have to read textual description of RESFTful services to understand how to develop clients to interact with them.

It should be noted that another key distinction between above approaches is that WS-* and ebXML approaches are industry initiatives, while SWS is mainly promoted by academia. REST architectural style is proposed in the academic environment, but it has been favored by industry.

3.2 Analysis of SOA approaches using integration layers

Figure 3 compares the above four approaches of SOA realizations in various integration layers. As it can be seen WS-* family of approaches, semantic Web services, and ebXML target integration at the business logic level. On the other hand, RESTful services mainly intend to simplify integration at the data layer, and recently they have been also used for integration at the presentation layer [5]. There has been an extensive study on comparison of standardization efforts in WS-* family, semantic Web services and ebXML (the reader is referred to [12, 24, 18]). In the following, we only focus on emerging technologies in this area,

and in particular, *data services*, which are RESTful services and target the data integration layer, *service component architecture (SCA)* for integration in the business logic layer, and finally *mashups* for the integration at the presentation layer.

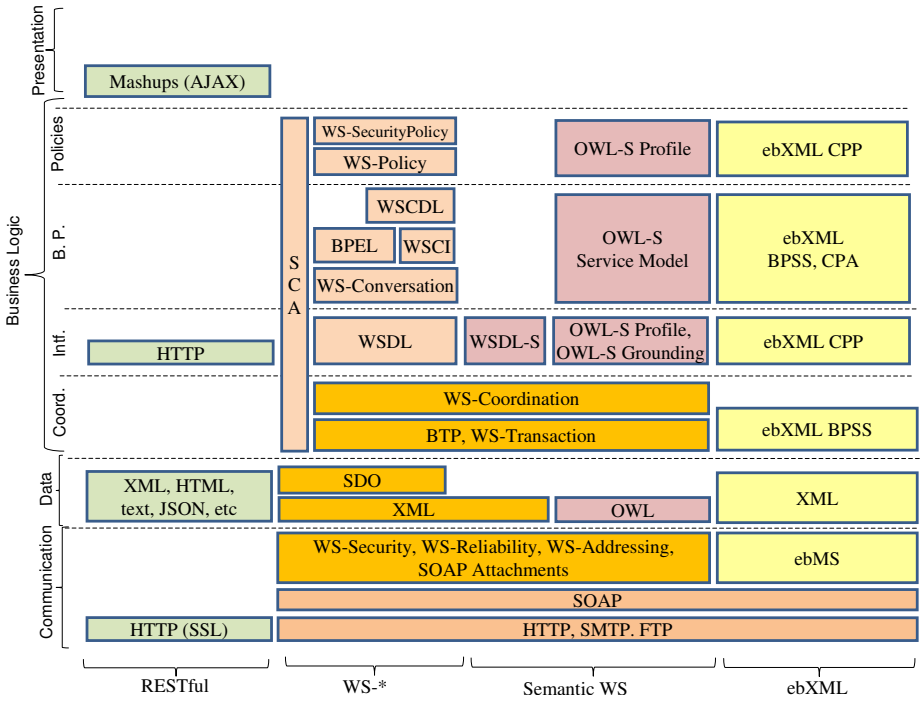

Fig. 3. The comparison of SOA realization approaches in the context of proposed integration layers

3.2.1 Data level integration: Data Services and SDO

Over the last few years, there has been an enormous increase in the number of distributed data sources, which are needed to be accessed over networks, and more notably over the Internet. The concept of "service" in SOA has provided a proper abstraction for wrapping and offering application over the Internet. This abstraction has been adopted to expose data sources with different types over the Internet. The term *data services*, coined by Microsoft[4], is used to refer to such services [4, 1]. Data services provide solutions for integration at the data layer. Data services can be used to provide virtual, aggregated views of data in multiple data sources. Hence, a data service provides data mediation, integration and also an abstraction for the underlying data sources. They simplify the data

[4] Astoria project, http://astoria.mslivelabs.com

access, integration and manipulation. Data services also can be used to expose data integration systems as services.

To implement data services, REST architecture is adopted, due to its simplicity, so that they can be accessed over HTTP and identified using a URI. The data in a data service is represented using an abstract model, called *entity data model*, which is an extended form of entity-relationship model. A data service can be configured to return the data in several formats including XML, JSON[5], RDF+XML, text, etc. It supports HTTP GET method for accessing data and HTTP methods such as PUT, POST or DELETE to manipulate data through the data service.

Integration at the data layer is also needed in business logic layer integration approaches. In such scenarios, data has to be exchanged between services and non-services applications (e.g., Java programs) and other data sources. XML has been adopted as the data format by WS-* family, which is intended for integration at the business logic layer (see Figure 3). To facilitate data exchange between both services and non-services and data sources using a single format a generic data format called *service data objects*(SDO)[6] is introduced. SDO offers more than a data format. Indeed, it provides a data programming architecture and a set of APIs for accessing and manipulation of data.

SDO architecture consists of three components: *data objects*, *data graph*, and *data access service*. Data objects contain a set of named properties that contain data elements or refer to another data objects. There are also data object APIs to access and manipulate the data. Data graphs are in fact envelops for data objects, which are transported between partner applications. Data graphs keep the track of changes in data objects by partner applications. Data graphs can be constructed from data sources, e.g., XML files, relational databases, EJBs, and services, e.g., Web services, and adapters (implementing data mediation and transformations). Finally, data access services are the software components that populate data graphs from data sources and services, and manipulate the data graph based on manipulation of data sources.

"Data access services" in SDO play a similar role to that of "data services" above. However, SDO provide a more rich data representation, exchange and manipulation approach made for data integration over heterogeneous data sources in an enterprise. SDO approach targets data integration with business logic level integration solutions. SDO offers a programming platform for data integration, and hence should be used by integration developers. However, data services target data integration for end users or non-expert users. SDO is currently widely supported in the implementation tools, and its specification has been sent to OASIS for standardization.

3.2.2 Business logic level integration: SCA

Web services (WS-*) approach mainly targets integration at the business logic

[5] http://www.json.org
[6] http://www.osoa.org/display/Main/Service+Data+Objects+Home

level (see Figure 3). The standardization in Web services simplifies interoperation at the business logic level from basic coordination layer to policies and non-functional properties. However, besides standardization, realizing SOA requires programming models, methodologies and tools to enable interoperation and application integration. In addition, in an enterprise not only Web services, but also existing functionalities that are not Web services have to be integrated in building integrated applications. As the concept of service promotes reuse, an important aspect is to provide a framework to support users in composing services and other non-services functionalities, which can be exposed as services.

To fulfill the above requirements, a group of software vendors including BEA, IBM, Oracle and SAP led an initiative represented by a set of specifications, called *service component architecture (SCA)*[7]. The intention of SCA is to simplify the creation and integration of business applications using SOA paradigm. In this architecture, an application is seen as a set of components (services), which implement (service) interfaces. It provides abstractions and methodologies for component construction, component composition (assembly) and deployment. The framework is intended to be neutral to component implementations (it supports as component implementation languages such as Java, BPEL, PHP, C++, .NET, etc). The principle that this architecture follows is to separate the business logic implementation from the data exchange between components.

SCA offers a service assembly models that is a framework for the composition of components into bigger ones, which can be deployed to the server together, or into systems that can be deployed separately. An SCA component (simple or composite) can be exposed as a service, and can consume other external service components. The communication between components is modeled using *wires*. SDO data representation format (see Section 3.2.1) is developed to be used for transportation of data on wires between components in SCA. Currently, SCA has been submitted for standardization to OASIS.

Comparing SCA with other standard specifications (e.g., in WS-* family of standards), it is not intended to address integration from one specific aspect. However, it provides an architecture, programming models and abstractions to support development of large scale systems using SOA across different layers in the business logic level (see Figure 3). It builds on and exploits the offerings of SOA and in particular Web services, and take the idea of "software as service" one step ahead by enabling to expose business logic functionalities (developed in different programming languages) as services that can communicate, be reused and composed with other services.

3.2.3 Presentation level integration: Mashups

The integration problem at the communication, data, and business logic level has been extensively studied, as discussed in previous sections. However, little work has been done to facilitate integration at the presentation level. Since development of user interfaces (UI) is one of the most time-consuming parts of application development, testing and maintenance, the reuse of UI components is

[7] http://www.osoa.org/display/Main/Service+Component+Architecture+Home

as important as reuse of business logic [5]. Recently, the concept of Web mashups and related technologies have been introduced, which take the first step in this direction.

Web mashups are Websites or Web applications that combines content and presentations from more than one source into an integrated experience [17]. Mashups are developed by compositing data, business logic (APIs) and UI of existing applications or services. The difference with traditional integration approaches is that Web mashups also integrate UI components. Nowadays, mashups are implemented using AJAX (Asysnchronous Javascript + XML) [9], but it is not necessary to be implemented using it. AJAX follows the REST architecture and aim to allow client side browser based applications to provide a rich and responsive interface at the same level of desktop applications. It enables to send requests to Web servers and services and receive responses without blocking.

The common principle in mashups is to quickly compose an application from existing (REST, Javascript, RSS/Atom, and SOAP) services. Mashup applications usually combine services for unexpected usages. The main feature of mashup, from an integration point of view, is that they allow for integration at the user interface (presentation) level. Mashup can be also considered as an ad-hoc approach for composition of existing content and services for building situational applications (typically short-lived, and just-in-time solutions), which is to the interest of many end users. To support development of mashup applications, numerous tools and frameworks have emerged recently to assist developers and end users. Examples of these tools and frameworks are Yahoo Pipes[8], Google Mashup Editor[9], Microsoft Popfly[10], and Intel Mash Maker[11].

Recently, a broader term, i.e., Web 2.0 [20] has been introduced to refer to all user-centric creation, access and sharing of information and presentation components on the Web. Web 2.0 has transformed the way that end users are using the Web. In Web 2.0, users collaborate and share information in new ways such as social networking and wikis. Web 2.0 consists of a set of principles and practices that makes the existing Web technologies more people centric. The common principles of Web 2.0 include: (i) looking at the web as a platform that allows extending the concept of service to any piece of data, software or application that is exposed on the Web, (ii) using the collective intelligence (collaboration) to create, share, compose and refine applications. This mandates offering lightweight programming models, and rich user interface. AJAX and mashups aim to provide such programming language models and user interfaces.

The composition and integration in the mashups are mainly based on data flow (e.g., a series operations performed on the data flowing from one component to another in Yahoo Pipes), and synchronization is based on events (e.g., in using Javascripts and receiving response) rather than ordered invocation of services, which is the main approach in business logic level integration (e.g., WS-* family).

[8] http://pipes.yahoo.com
[9] http://editor.googlemashups.com
[10] http://www.popfly.ms
[11] http://mashmaker.intel.com

It should be noted that the mashups are about simplicity, usability and ease of access, and that unlike WS-* approach or data integration approaches (e.g., ETL) this simplicity has the upper hand over completeness of features or full extensibility.

4 Conclusions and Future Directions

As reviewed in this chapter, available approaches for realization of SOA remarkably simplify integration at the communication, data, and business logic levels. This is achieved by proposing frameworks, abstractions and standardization efforts that increase the opportunities for homogeneities and unification of communication protocols and data format for data exchange. However, the integration at the end user (presentation) level has not yet received the required attention.

We believe the end users are the focus of next wave of research and development work in the various approaches in SOA both in RESTful services and mashups, and also in WS-* family of specifications. As also can be seen in Figure 3, the presentation level integration for the RESTful services does not still provide a full-fledged approach for integration, and there is no counterpart efforts in WS-* approaches. One possible future direction could be also to adopt the "service" concept as an abstraction for integration at the presentation level, so that presentation components (and GUIs) are offered with published interfaces that can be easily integrated and composed. Examples of initiative in this directions is Google Map APIs [12]. However, further end user level support is needed as currently, such practices involves lots of low level scripting and coding, which may not be convenient for end users.

The end-user driven trend in integration has also been witnessed by introduction of new concepts such as *process of me* by Gartner [10], and *Internet service bus* [7]. Gartner report states that we should redefine processes in an enterprise, and put the focus on people so that individuals have understanding and control of processes that they are involved in them. The process of me includes integration of end user tools such as instant messaging, spreadsheets, threaded discussions and management of real-time events with business process applications, and other Internet technologies based on Web 2.0. Therefore, a major enabler step for approaches that offer business logic integration approaches (e.g., WS-* family) to fill this gap, and to support individuals (employees) by integration of ad-hoc personal and collaboration tools with processes supported by traditional business applications. Realizing the concept of process of me requires framework and tool support to allow users to define their own views of the process execution in the enterprise with preferred end users-oriented tools.

With a similar spirit, Internet service bus proposal takes the end user involvement to the next level by promoting the ideas of creating end user Web applications on the Web and using the Web as an execution platform for end

[12] code.google.com/apis/maps

user applications and other software and services. This idea can be seen as taking what SCA (and in general enterprise service bus) provides for professional integration developers in composition of services and application and offering them for end users. In such an environment end users should be supported in the process of finding existing services and integrating them.

It should be noted that while SOA, and the abstraction of "service" simplifies significantly the integration at various level, there is still the need for bridges, mediators, adapters and mismatch resolution frameworks (e.g., data mediators, business protocol adapters, and policies resolution frameworks). In fact, SOA, and in particular standardization in SOA, reduces the opportunities of heterogeneities. However, at the higher levels of abstractions (e.g., business-level interfaces, business protocols, and policies), WS-* family offers languages to define the service interface, business protocol and policies. There have been considerable research and development efforts in identifying and classifying mismatches between such service specifications, and their resolution (e.g., see [19, 6, 25]). However, these approaches still involve many manual steps by the developers. Specially, there is a need for automated approaches for (simple) data mediation between various formats at the end user side, when building mashup applications, and in spreadsheet environments, which are most popular tools for data integration and manipulation.

Acknowledgement

Authors would like to thank the anonymous reviewers for their valuable feedbacks on the earlier draft of the chapter.

References

1. A. Adya and et al. Anatomy of the ADO.NET entity framework. In *SIGMOD*, 2007.
2. G. Alonso, F. Casati, H. A. Kuno, and V. Machiraju. *Web Services - Concepts, Architectures and Applications*. Springer, 2004.
3. C. Bussler, D. Fensel, and A. Maedche. A conceptual architecture for semantic web enabled web services. *SIGMOD Rec.*, 31(4):24–29, 2002.
4. M. Carey. Data delivery in a service-oriented world: the bea aqualogic data services platform. In *SIGMOD*, 2006.
5. F. Daniel, J. Yu, B. Benatallah, F. Casati, M. Matera, and R. Saint-Paul. Understanding ui integration: A survey of problems, technologies, and opportunities. *IEEE Internet Computing*, 11(3):59–66, 2007.
6. M. Dumas, M. Spork, and K. Wang. Adapt or perish: Algebra and visual notation for service interface adaptation. In *In Proc. of BPM'06*, pages 65–80, 2006.
7. D. F. Ferguson, D. Pilarinos, and J. Shewchuk, editors. *The Internet Service Bus*. Microsft, `msdn2.microsoft.com/en-us/library/bb906065.aspx`, May 2006.
8. R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, USA, 2000.
9. J. J. Garrett, editor. *Ajax: A New Approach to Web Applications*. `http://www.adaptivepath.com/ideas/essays/archives/000385.php`, February 2005.

10. Y. Genovese, J. Comport, and S. Hayward, editors. *Person-to-Process Interaction Emerges as the 'Process of Me'.* Gartner, `http://www.gartner.com/DisplayDocument?ref=g_search&id=492389`, May 2006.
11. A. Y. Halevy and et al. Enterprise information integration: successes, challenges and controversies. In *SIGMOD Conference*, pages 778–787, 2005.
12. D. J. Kim, M. Agrawal, B. Jayaraman, and H. R. Rao. A comparison of b2b e-service solutions. *Commun. ACM*, 46(12):317–324, 2003.
13. M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.
14. K. Li, K. Verma, R. Mulye, R. Rabbani, J. A. Miller, and A. P. Sheth. Designing semantic web processes: The WSDL-S approach. In *Semantic Web Services, Processes and Applications*, pages 161–193. Springer, 2006.
15. D. L. Martin, M. Paolucci, S. A. McIlraith, M. H. Burstein, and et al. Bringing semantics to web services: The owl-s approach. In *Proc. of 1st Int'l Workshop Semantic Web Services and Web Process Composition (SWSWPC'2004)*, pages 26–42, 2004.
16. B. Medjahed, B. Benatallah, A. Bouguettaya, A. H. H. Ngu, and A. K. Elmagarmid. Business-to-business interactions: issues and enabling technologies. *The VLDB J.*, 12(1):59–85, 2003.
17. D. Merrill, editor. *Mashups: The new breed of Web app.* `http://www.ibm.com/developerworks/library/x-mashups.html`, April 2006.
18. H. R. M. Nezhad, B. Benatallah, F. Casati, and F. Toumani. Web services interoperability specifications. *IEEE Internet Computing*, 39(5):24–32, 2006.
19. H. R. M. Nezhad, B. Benatallah, A. Martens, F. Curbera, and F. Casati. Semi-automated adaptation of service interactions. In *In Proc. of WWW'07*, pages 993–1002, 2007.
20. T. O'Reilly, editor. *What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software.* `http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html`, September 2005.
21. M. P. Papazoglou and W.-J. van den Heuvel. Service oriented architectures: approaches, technologies and research issues. *VLDB J.*, 16(3):389–415, 2007.
22. A. Polleres and R. Lara, editors. *A Conceptual Comparison between WSMO and OWL-S.* `www.wsmo.org/2004/d4/d4.1/v0.1/`, 2005.
23. E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.
24. M. Turner, D. Budgen, and P. Brereton. Turning software into a service. *IEEE Computer*, 36(10):38–44, 2003.
25. E. Wohlstadter, S. Tai, T. Mikalsen, I. Rouvellou, and P. Devanbu. Glueqos: Middleware to sweeten quality-of-service policy interactions. In *In Proc. of ICSE'04*, pages 189–199, 2004.
26. P. Ziegler and K. R. Dittrich. Three decades of data integration - all problems solved? In *IFIP Congress Topical Sessions*, pages 3–12, 2004.