Rich Services: The Integration Piece of the SOA Puzzle

Matthew Arrott, Barry Demchak, Vina Ermagan, Claudiu Farcas, Emilia Farcas, Ingolf H. Krüger, Massimiliano Menarini

University of California, San Diego – Calit2
{marrott, bdemchak, vermagan, cfarcas, efarcas, ikrueger, mmenarini}@ucsd.edu

Abstract

One of the key challenges to successful systems-of-systems integration using Web services technologies is how to address crosscutting architectural concerns such as policy management, governance, and authentication, while still maintaining the lightweight implementation and deployment flavor that distinguishes Web services from earlier attempts at providing interoperable enterprise systems.

To address this challenge, this article introduces the notion of a Rich Service, an extension of the standard service notion, based on an architectural pattern that allows hierarchical decomposition of system architecture according to separate concerns. Rich Services enable the capture of different system aspects and their interactions. By leveraging emerging Enterprise Service Bus technologies, Rich Services also enable a direct transition from a logical to a deployed service-oriented architecture (SOA). This results in immediate benefits not only in SOA design, implementation, deployment, and quality assurance, but also in the traceability of architectural requirements to an SOA implementation.

1. Introduction

Web service technologies have greatly facilitated rapid data and application integration by means of open standards for information exchange, service discovery, and binding across the Internet. The reduction of the integration challenge to the technological issues came at the cost of addressing the full complexity arising from the composition of business processes and associated applications from multiple enterprises with requirements such as security, policy, and governance. Reintegrating Web services with these and other crosscutting business concerns (such as manageability, scalability, and dependability), while maintaining the lightweight flavor of Service Oriented Architectures (SOAs) is one of the key remaining challenges in design, deployment, and quality assurance.

WS: State of the Art. The Web services community has learned important lessons from attempts to establish CORBA as an all-encompassing middleware platform for business systems integration. The heavy weight of tightly integrating all these concerns into the CORBA infrastructure turned out to be difficult – to the point that the core goal of system interoperability was jeopardized both from a logical and technological perspective.

The introduction of Web services lead to a much needed separation of concerns: first, the core interoperability issue was addressed by using standard communication protocols (HTTP/SOAP), data marshaling (XML), and interface description (WSDL) technologies. Second, some of the crosscutting technological and business issues were addressed in a separate step (e.g., the need to discover and connect to services at run time lead to the creation of UDDI, and security concerns lead to the enhancement of SOAP via WS-Security [7]).

With these basics in place, attention turned to the other critically important aspects of system integration we have mentioned above. Much attention was paid to maintaining the lean technological core of Web services. For instance, service composition has been one of the most active development areas for Web service technologies in recent years. Two interesting directions in broadening the Web service concept are the semantic Web and related business workflows. Extensions to include semantic/ontology information (e.g., OWL-S [16]) aim to enrich the WS core technologies by means of meaningful and flexible runtime discovery, binding, and automatic composition of newly published services. Extensions toward service orchestration (e.g., BPEL [3]) and choreography (e.g., WSCL [13], WS-CDL [2]) focus on the coordination of Web services to support a business process.

Challenge: horizontal and vertical service integration. The move towards tailored Web service standards and technologies layered over a lean technological core resulted in a fragmentation of concerns. This fragmentation created the challenge of how to integrate the pieces of the puzzle back into a coherent picture suitable for enterprise-scale SOAs. We distinguish between “horizontal” and “vertical” service integration. Horizontal service integration refers to managing the interplay of services and the corresponding crosscutting concerns at the same logical or deployment level. Vertical service integration refers to the hierarchical decomposition of one service (and the crosscutting concerns pertaining to this service) into a set of sub-services such that their environment is shielded from the structural and behavioral complexity of the embedded sub-services and the form of their composition.
At the deployment level, horizontal service integration is increasingly being addressed by means of Enterprise Service Bus (ESB) technologies. ESBs combine three key elements to facilitate system integration: (a) the strengths of message-oriented middleware (MOM), (b) a flexible plugin architecture, which enables inline message processing (e.g., for policy enforcement or security transformations), and (c) a rich set of data adapters/connectors to facilitate rapid connections between emerging and legacy data sources, applications, and services. While this combination has shown promise as a facilitator of rapid systems-of-systems integration at the implementation/deployment level [24], the transition from a logical architecture to an ESB implementation is so far left largely unexplored. Furthermore, only ad-hoc support for vertical service integration is provided in the context of ESBs, which is another impediment to realizing the full scalability potential of SOAs in an enterprise setting.

Contribution. To conquer this challenge, we propose to enrich the well known Web service notion along the two major dimensions we have identified above. To solve the problem of horizontal service integration, we propose to embed services into an architectural blueprint that lifts the integration benefits (a) – (c) of ESBs from the deployment level to the level of logical architectures. To solve the problem of vertical service integration, we propose the introduction of a lightweight concept for hierarchical service composition, which extends the architectural blueprint across all hierarchical layers. We call services that exhibit these properties Rich Services. In essence, a Rich Service wraps “traditional” services, such as Web services, within an architectural framework for addressing horizontal and vertical integration concerns.

By design, the distance between the proposed Rich Service architecture blueprint and various deployment options, especially those relying on ESBs, is minimal. This results in immediate traceability between requirements captured and addressed at the level of logical architectures and their implementation.

As we will discuss in detail in Section 2, the Rich Service notion promotes the separation of concerns by allowing the decomposition of a problem according to various aspects, each in isolation. Following the ESB model, the Rich Service architecture blueprint leverages a flexible communication framework where information can be routed between plugins according to rules that can change at run-time. This key feature enables a seamless composition of Rich Services. Finally, the hierarchical nature of the architecture allows it to address complex problems, including systems-of-systems integration, by promoting the decomposition of systems. The result is a collection of services that implement complex hierarchical concerns, thereby aligning with the organizational topology of the stakeholders – concerns pertaining to a stakeholder are grouped and evaluated in the (hierarchical) context of the stakeholder.

Outline. The remainder of this paper is structured as follows. In Section 2, we introduce Rich Services both at the logical and deployment levels. In Section 3, an architectural case study clarifies how Rich Services are applied in a complex system with a variety of crosscutting concerns. In Section 4, we discuss open issues and future work on composition operations, the formal definition of Rich Services, and the related work. We present our conclusions and an outlook in Section 5.

2. Rich Services

The number and complexity of various business concerns (e.g., governance, security, and policy) that need to be addressed by Web services go beyond Web services’ capabilities. The demand for the integration of these concerns leads to the urge for a richer framework for Web services that is scalable to complex systems, is dynamic, and provides decoupling between various concerns.

In this section, we propose the architecture blueprint of Figure 1 as a comprehensive and flexible framework for the analysis, design, and deployment of large scale and complex Web service-based systems. The proposed architecture is considered both a logical architecture and a guide to a deployment architecture. The one-to-one mapping between the logical architecture and its deployment is an important feature, which contributes to the practicality of the approach. In the following, we will explore the proposed architecture from both logical and deployment perspectives.

2.1. Logical View

Figure 1 depicts a logical service-oriented architecture for complex Web service-based systems. This architecture is inspired by ESB architecture/implementations, such as Mule and Cape Clear. The main entities of the architecture are (a) the Service/Data Connector, which serves as the sole mechanism for interaction between the Rich Service and its environment, (b) the Messenger and the Router/Interceptor, which together form the communication infrastructure, and (c) the Rich Services connected to Messenger and Router/Interceptor, which encapsulate various application and infrastructure functionalities. In the following, we elaborate on each of these main entities and their role in the system.

To address the horizontal integration challenge, the logical architecture is organized around a message-based communication infrastructure having two main layers. The Messenger layer is responsible for message transmission between endpoints. By providing the means for asynchronous messaging, the Messenger supports decoupling of
Rich Services. The second layer, the Router/Interceptor, is in charge of intercepting messages placed on the Messenger, then routing them. The routing policies of the communication infrastructure are the heart of the Router/Interceptor layer. Leveraging the interceptor pattern readily facilitates dynamic behavior injection based on the interactions among Rich Services. This is useful for the injection of policies governing the integration of a set of horizontally decomposed services.

The main entity of the architecture is the notion of Rich Service. A Rich Service could be a simple functionality block such as a Web service, or it could be hierarchically decomposed.

We distinguish between Rich Application Services and Rich Infrastructure Services. Rich Application services interface directly with the Messenger; they initiate interactions. Rich Infrastructure Services interface directly with the Router/Interceptor; they monitor and govern the interactions among the application and other infrastructure services.

Policy enforcement, encryption, failure management, and authentication are typical examples of Rich Infrastructure Services. These are mainly intermediary services [17], and need direct access to the Router/Interceptor layer. The purpose of an encryption service, for instance, is to ensure that all messages transmitted over the communication medium are encrypted. A traditional service approach would require modifications to every service in the system in order to integrate the encryption mechanism, leading to scattered functionality. On the other hand, the proposed architecture introduces encryption as an intermediary service that can inform the Router/Interceptor layer to modify the routing tables to ensure that every message sent to the external network must first be processed by the encryption service. Only the communication infrastructure needs to be aware of the encryption service.

A Service/Data Connector is the means by which Rich Services are connected to the communication infrastructure. The Service/Data Connector encapsulates and hides the internal structure of the connected Rich Service, and exports only the description and interfaces that the connected Rich Service intends to provide and make visible externally. The communication infrastructure is only aware of the Service/Data Connector, and does not need to know any other information about the internal structure of the Rich Service. This helps tackle the vertical integration challenge introduced by systems-of-systems.

One important aspect of the proposed architecture, together with the notion of Rich Service, is that it is hierarchical. Every Rich Service S.i (as in Figure 1) can be decomposed into further Rich Services, say S.i.1 … S.i.k, connected via their Service/Data Connectors to S.i’s internal communication infrastructure. S.i’s internal Messenger is then connected via another Service/Data Connector to the communication infrastructure in the next higher level of the hierarchy. The Service/Data Connector in this case acts as a gateway, and is responsible for routing the messages entering the composite Rich Service to the appropri-
The need to change or adapt the implementation produced at runtime and provide their functionality of dynamic systems, where new services can seamlessly communicate. A second unique feature of the proposed architecture is that the use of the Router/Interceptor layer removes dependencies between services and their relative locations in the logical hierarchy. In this section, we explain how this logical architecture can be mapped rapidly to a deployment architecture.

2.2. Deployment of Rich Services

In this section, we present one possible instantiation of the architecture, leveraging existing WS-related technologies: the Mule ESB framework [8], the Java language, and the core Web services technologies identified in Section 1. A similar deployment mapping can easily be accomplished using alternative technologies.

As mentioned in Section 1, ESBs combine the strengths of message-oriented middleware, a flexible plugin architecture for processing messages to handle crosscutting concerns for a set of horizontally decomposed services, and a rich set of data adapters/connectors to facilitate rapid connections between emerging and legacy data sources, applications and services. Examples of ESB implementations include Architect’s Toolbox, Cape Clear’s ESB, Fiorano ESB, Sonic ESB, SpiritSoft’s Spiritwave, and CodeHaus’ Mule.

The close alignment of the logical architecture for the Rich Services introduced in Section 2.1 and the ESB deployment architecture yields a direct mapping from logical to deployment concepts.

Figure 2 shows an example of a concrete deployment of a Rich Service. Each of the nested Rich Services blocks could be decomposed using the same architecture with the same or different deployment technologies. In particular, the Mule transport layer corresponding to the Messenger element of the logical architecture can be leveraged to connect both components local to Mule and external components. External components are connected to the Mule framework using any subset of the Mule-supported transport technologies. For instance, they could be implemented as Web services and connected via the Mule SOAP transport provider. Furthermore, external services can be another implementation of Rich Services, as illus-
Mule leverages various message bus implementations [19] to create different communication channels that connect different sets of components. Mule allows the creation of two special types of plugins (i.e., routers and interceptors) which are allowed to modify how messages are dispatched and how components’ methods are called. We can use those two plugin types to implement Rich Infrastructure Services. Routers allow the Mule framework to receive messages from one channel and re-publish them according to arbitrary routing rules. We can use this machinery, for instance, in the implementation of the Service/Data Connector. In this case, the router in the connector enables relevant messages to be exchanged between particular elements of the Rich Service and the external environment. Interceptors can modify how the elaboration is carried out by the different components inside Mule, and can be used to address common concerns by modifying the behavior of a set of components. This bears similarity to the Aspect-Oriented Programming paradigm.

The Mule infrastructure can be leveraged to easily create simple services (i.e., services that are not further decomposed according to the Rich Service pattern) using plain Java objects. Such an object can be structured as a Mule Universal Message Object (UMO) so as to decouple the computation from delivery and transport mechanisms. Using the infrastructure provided by Mule, it is then possible to add data transformation (via transformer objects), and have UMOs participate in interactions with other components (using appropriate message receivers and dispatchers) to any of the supported channels. In this way it is possible to have UMO objects that interact, for example, with external web services, other UMO components, or the external Service/Data Connector without explicit knowledge of how and where the other component is implemented. This feature facilitates code reuse and supports separation of concerns, the core concept around which Rich Services are modeled.

The implementation of the Service/Data Connector is straightforward using the Mule framework. It can be implemented by having a router listen and publish relevant messages on an external channel. The router can enforce encapsulation policies for the Rich Service and expose just some of the internal components to the environment external to the Rich Service. Leveraging the many technologies supported by Mule as transport mechanisms, a connector can publish itself via JMS, HTTP, SOAP, etc. In particular, if different Rich Services share the same Java virtual machine (vm), the connection can be implemented using simple intra-vm method calls, immensely decreasing the communication overhead. Therefore, this implementation strategy allows the code to be highly independent of the physical deployment configuration, yet to leverage deployment configurations to optimize performance. The encapsulation and hierarchical decomposition of the Rich Service notion is also independent of where the service implementation is deployed, allowing the reuse of logical models when different deployment constraints have to be satisfied. Again, this feature promotes separation of concerns, thus reducing the complexity involved in developing and maintaining large distributed systems.

3. Case Study

We demonstrate the utility of the proposed architecture blueprint by using a case study from the domain of global ocean observatories, namely the federated Ocean Research Interactive Observatory Networks (ORION) program [25]. This case study is an elaboration of the ORION-CI conceptual architecture available at [25]. Clearly, here we can only scratch the surface of the complexity of building an architecture of the scale of ORION. However, it allows us to show how to decompose services both horizontally and vertically by using the Rich Services architecture blueprint, and also to demonstrate the direct mapping to state of the art Web services and ESB technologies.

A system satisfying the goals of ORION would support scientific discovery by providing eligible oceanographers ubiquitous access to instrument networks for sensing and actuation, computational resources, and modeling and simulation facilities, as well as means for distributed data storage and access. A traditional SOA approach would quickly reach its limits in the face of the challenges induced by the diverse requirements of supporting governance of the different authority domains, access policies, and concerns of the multiple stakeholders involved in such a complex system-of-systems. The complexity of the resulting cyber-infrastructure requires a decomposition methodology and an architecture that supports the deployment, operation, and distributed management of thousands of independently owned taskable resources of various types (e.g., sensors, sensor platforms, processes, numerical models and simulations) across a core infrastructure operated by independent stakeholders.
The Rich Service architectural pattern enables hierarchical structuring of the stakeholders’ logical roles into the cyber-infrastructure, and encapsulation of crosscutting concerns according to their individual policies. In addition, the concerns and authority domains of a stakeholder may be extended beyond the infrastructure under its direct control through business relations (e.g., contracts), such as owning an entity but having it managed by another stakeholder. Figure 3 represents several stakeholders as high-level Rich Services such as Global Scale Observatory, Modeling Facility, and Research Laboratory. Such a decomposition allows us to reason about their role in the cyber-infrastructure without dealing directly with their deployment models. The fractal nature of the architecture scales from the top-level view of the ORION system-of-systems down to the lower deployment levels in the various participating organizations. The figure displays multiple views of the system, illustrating specific roles and showing several stakeholders’ crosscutting concerns.

As an example of a use case playing out over this architecture, we consider an oceanographer accessing a remote ocean instrument. The oceanographer and instrument are part of different authority domains, each with its own set of requirements and policies. The oceanographer belongs to the Research Laboratory, for which the core concerns are the identification and authentication of the oceanographer, and the provisioning of the research facilities within a specific set of policies regarding the available ocean instruments, knowledge bases, and possible experiments at a given time.

The identification and the management of the remote Instrument also concerns another stakeholder, as the Instrument is located deeper in the hierarchy of the Global Scale Observatory, and within the local domain of control of a Regional Cabled Observatory near the seashore. Although the Instrument belongs to an Observatory Service associated with the Research Facility, it still has to obey the set of policies regarding the power usage, allowed research activities, timing of the activities, and the available mathematical processes for the resulting data of the encompassing Regional Cabled Observatory. These requirements are instances of additional Rich Services provided by the stakeholders of the program. For example, the mathematical processing services could be available in terms of CPU processing time on a supercomputer center governed by the Modeling Facility.

The Router/Interceptor layers and the Service/Data connectors of the Observatory and subsequent higher-level services enable the composition and collaboration of different concerns, and exposes the oceanographer to the capabilities of the Instrument further up in the hierarchy. Thus, they facilitate a seamless communication along the levels of hierarchy without the oceanographer or Instrument being aware of the fact that they are communicating with entities out of their authority domain.

A possible deployment plan might include classic Web services or a more general ESB-based technology, such as Mule. Thus, each embedded Rich Service in the path from the oceanographer to the Instrument might have its own set of Rich Infrastructure Services that would transparently alter the message flow to implement identification, authentication, accounting, or logging of the data/control messages of interest at that level.

At each level, one or more Rich Services might deal with the specific concerns and policies of a stakeholder. Thus, the overall role of a stakeholder in the cyber-infrastructure would be the reunion of all its roles on all levels of the architecture. Translated into a flat view, this
reunion is the root of the complexity that makes the development of such systems-of-systems difficult. The vertical integration capability of the proposed architecture solves this problem and allows the decoupling of the local, regional, coastal and global concerns of the stakeholders at all levels of the architecture, and allows a simplified management over the lifetime of the program.

4. Discussion and Related Work

Several approaches have been proposed to compose Web services in complex distributed applications [10]. A first approach to service composition entails explicitly describing the interactions between all entities, particularly in the various services. For example, BPEL[3] is a flow-composition language based on the orchestration model, where a central coordinator calls all the composed services. Another interaction-based approach, known as choreography or conversation model (e.g., [14], WSCL [13], WS-CDL [2]), takes a global view that mandates how each entity must actively participate in the conversation. Rich Services support these composition approaches – their encapsulation and hierarchy guide developers to focus on one hierarchical level at a time. Dynamic interaction-based frameworks such as E-flow [5] could also leverage the dynamic routing capabilities of our architecture to enable binding to services selected at run time.

A second approach to composition leverages rich ontologies that describe service characteristics. For example, the Semantic Web [6] community uses semantic annotations to reason about Web services by using languages such OWL-S [16]. Properties, preconditions, and effects of Web services are explicitly declared [15]; they specify the state change in the world produced by the execution of the service. Therefore, the composition of Semantic Web services can be generated automatically with techniques from AI planning [1]. QoS-based dynamic composition approaches extend WSDL by using ontologies to describe QoS requirements. This enables frameworks such as Star-WSCoP [4] to achieve semantic matches for services, and to provide an estimation of the QoS metrics of a composite service. Both the flexible architecture of Rich Services and the deployment strategy leveraging the ESB allow us to expose richer interfaces and reconfigure communication at run time, thereby enabling the use of ontology-based composition techniques.

Following the philosophy of separation of concerns, our architecture enables dynamic interception and routing of messages. Thus, it allows the integration of business rules and rule-based engines into a service composition, and supports approaches such as [11] and [12], which take an aspect-oriented view of this integration. In particular, [11] leverages an ESB to achieve the integration, which is similar to our suggested deployment infrastructure.

Recently, OASIS and W3C released two important documents capturing the essence of SOA and Web services. The OASIS SOA Reference Model [18] builds a common understanding of the core characteristics of a Service Oriented Architecture. Our work complements this effort by providing an architecture pattern that is a SOA particularly suitable to integrating complex distributed applications. The W3C Web Services Architecture [17] defines the core architectural and technological entities for applications based on Web services infrastructures and their relationships. We leverage this work at the logical architecture level by encapsulating the Web service entities in the hierarchical structure of a Rich Service, and in the deployment architecture level by configuring an ESB infrastructure that adopts standard WS technologies.

The idea of leveraging models to develop applications is not new. MDA [26], for example, promotes the use of models to document and engineer software. In [9] an approach to model-driven composition with UML, which is mapped to BPEL, is presented. Also, in the context of Web services MDA has found valuable application. For example, in [23] the authors promote the use of UML to develop Web services applications according to MDA. Similar to our approach, they promote the disentanglement of models from details related to deployment and technological details. Instead of focusing on a particular modeling language, our approach uses an architecture pattern to decouple those concerns. Furthermore, we also promote this decoupling in the deployment stage.

An architecture-based approach to WS integration is presented in [21]. It leverages middleware and provides components at the application and middleware/infrastructure levels. It also advocates a flexible communication infrastructure based on routers/switches. Our approach is more general by providing encapsulation and hierarchy to deal with complexity, and by providing a deployment mapping to ESB. We enable the decomposition of systems according to concerns by allowing modularization of crosscutting aspects in infrastructure services.

Work on WS composition has highlighted its tight coupling with interaction modeling. [27] explores the use of Message Sequence Charts (MSC) to define interactions. [22], for example, presents a tool that transforms MSC to BPEL specifications to allow WS composition. We leverage similar techniques to compose Rich Services. Toward this goal we have already experimented with the definition and composition of services based on MSC in [20]. In future work, we plan to leverage and extend existing interaction modeling techniques to compose Rich Services.

The elaboration of a formal model for service interfaces and their composition is beyond the scope of this paper. However, in [20] a formal foundation for service-oriented systems is laid out; the concepts of components and services as total and partial behaviors presented in this
work will be leveraged in future work to provide the formal underpinning for Rich Services.

5. Conclusions

The approach introduced in this paper improves the state of the art in Web services development by providing a flexible and comprehensive architectural framework that can be used as a basis for a methodological approach to complex Web service application development. The notion of Rich Service improves on the flat view of services conveyed by standard WSDL interfaces by providing a hierarchical structure for the service and an explicit encapsulation gateway. Using ESB and WS technologies for implementation of Rich Services promotes component reuse and component interoperability across application domains.

The Rich Service architecture blueprint improves understandability and maintainability while still retaining the advantages of a traditional SOA. It allows the modeling of crosscutting concerns and composition at the logical level, and a direct mapping to deployment by leveraging the achievements of existing, lightweight technologies.

Rich Services, as a logical architecture blueprint, provide structure to the architectural layout of integration solutions both horizontally and vertically. This facilitates the investigation of systematic design and refactoring steps towards SOAs – a rich topic for further research.

6. Acknowledgements

Our work was partially supported by the NSF within the projects “RESCUE” (award #0331690), “Reosphere” (award #0403433), and “ITR: Collaborative Research: Looking Ahead: Designing the Next Generation Cyber-infrastructure to Operate Interactive Ocean Observatories” (award OCE/GEO #0427924), as well as by funds from the California Institute for Telecommunications and Information Technology (Calit2). We are grateful to the anonymous reviewers for insightful comments.

7. References