Chapter 4
Structural Services: A New Approach to Enterprise Integration

Jose Delgado
Lisbon University, Portugal

ABSTRACT

This chapter compares the Service-Oriented Architecture (SOA) and Representational State Transfer (REST) architectural styles and contends that both have advantages and limitations for enterprise integration. SOA, based on behavior, has a lower modeling semantic gap for complex applications but lacks support for structured resources common in lower-grained applications. REST is based on structure and hypermedia but has a higher semantic gap in complex applications and, as this chapter contends, does not entail a lower resource coupling than SOA. A new architectural style, Structural Services, is proposed to get the best of both worlds, while reducing coupling with structural interoperability based on the concepts of compliance and conformance. Unlike REST, resources are able to offer a variable set of operations, and unlike SOA, services are allowed to have structure and use hypermedia. A distributed service programming language is briefly described to illustrate how this architectural style can be instantiated.

INTRODUCTION

Enterprise integration is a very complex issue, spanning from lower levels, such as data and service interoperability, to the highest levels, including strategy and governance alignment. The latter are mostly dealt with tacitly or at the documentation level, in coordination with the enterprise architectures. The former typically resort to technologies based on architectural styles such as SOA (Erl, 2008) and REST (Fielding, 2000), and constitute the focus of this chapter, although the others are tackled as well, in less detail.

SOA is guided by behavior and REST by state. SOA defines which resource types are used and establishes the set of operations provided by each, whereas REST starts with a state diagram, without relevant concern about distinguishing which state belongs to which resource.

REST is more flexible, in the sense that, if the server changes the links it sends in the responses, the client will follow this change automatically by using the new links. This however, is not as general as it may seem, since the client must be able to understand the structure of the responses. It is not merely a question of following all the

DOI: 10.4018/978-1-4666-6473-9.ch004

links in a response. This is why REST favors standardized media types, to reduce the coupling between the client and the server. However, instead of reducing it, it merely transfers it from runtime to design time, when media types are defined. SOA is more static, since the service contracts are defined beforehand and there is no provision for resource structure, but benefits from the support provided by tools in detecting errors. This is the classical tradeoff that a programmer has to face when choosing between a dynamic or static typed language.

In REST, there is an interesting side effect of the fact that the client uses a response to perform the next state transition. Only the links corresponding to allowed transitions are there, in line with the *affordances* approach (Amundsen, 2012). This is equivalent to dynamically adjusting the interface available to the client, according to the current state of the application. This is good protection mechanism, already used in human computer interfaces, with menus that become enabled and disabled, according to context. In SOA, this is not easy to achieve. Once a link to a service is available, all its operations can be invoked.

The flexibility of REST comes at a price, since it lies at a much lower level than SOA and the architecture of the problem does not map as easily onto state diagrams as onto resource class diagrams and processes. Nevertheless, it is simpler than SOA, requiring only basic Web technologies, and scales much better, but only as long as the application complies with the constraints imposed by REST. Complex functionality and bidirectional interactions (requests from server to client) are not a good match for REST.

This means that the ranges of types of applications that are more adequate for SOA and REST are not the same. REST is a better match for Web style applications (many clients, stateless interactions), which provide a simpler interface. This is why all the major Internet application providers, including cloud computing, now have REST interfaces. SOA is a better match for functionally-complex, enterprise-level distributed applications, in which REST exhibits a higher semantic gap (Ehrig, 2007) between the problem and solution spaces.

Ideally, we would like:

- To combine the structural properties of REST's resources and links with the richness of generic service interfaces;
- To reduce the semantic gap of REST, without losing its dynamic structural characteristics;
- To reduce the coupling between the provider and consumer of a service (increasing its range of applicability and adaptability), without losing its contract-based support.

Unfortunately, the technologies currently used to implement SOA and REST provide no hint or support on how to do this. These include data description languages such as Extensible Markup Language (XML) and JavaScript Object Notation (JSON), protocols such as Hypertext Transfer Protocol (HTTP) and SOAP (formerly an acronym for Simple Object Access Protocol), and service description languages, such as Web Services Description Language (WSDL).

This is a direct consequence of the fact that these technologies evolved from the original Web of Documents, made for stateless browsing and with text as the dominant media type. That web was made for people, not for enterprise integration. Today, the world is rather different. The web is now the Web of Services, in which the goal is to provide functionality, not just documents. There are now more computers connected than people, with binary data formats (computer data, images, video, and so on) as the norm rather than the exception.

Yet, the evolution has been to map the abstraction of Web of Services onto the Web of Documents, with the major revolution of XML and its schemas. The document abstraction has