Chapter 9
Improving Application Integration by Combining Services and Resources

José Carlos Martins Delgado
University of Lisbon, Portugal

ABSTRACT

The main application integration approaches, the service-oriented architecture (SOA) and representational state transfer (REST) architectural styles, are rather different in their modeling paradigm, forcing application developers to choose between one and the other. In addition, both introduce more application coupling than required, since data schemas need to be common, even if not all instantiations of those schemas are used. This chapter contends that it is possible to improve this scenario by conceiving a new architectural style, structural services, which combines services and resources to reduce the semantic gap with the applications, allowing to tune the application integration between pure service-based and pure resource-based, or an intermediate mix. Unlike REST, resources are not constrained to offer a fixed set of operations, and unlike SOA, services are allowed to have structure. In addition, compliance is used to reduce coupling to the bare minimum required by the actually used application features.

INTRODUCTION

The world is increasingly distributed and most real case scenarios involve interaction between distributed applications that need to cooperate to achieve common or complementary goals. Examples of such scenarios include:

- Enterprise-class applications (Romero, & Vernadat, 2016), deployed on either conventional or cloud computing platforms (Ritter, May, & Rinderle-Ma, 2017), most likely including hybrid clouds, integrating the enterprise's owned infrastructure with one or more public clouds.

DOI: 10.4018/978-1-5225-7271-8.ch009

- Mobile cloud computing (Abolfazli, Sanaei, Sanaei, Shojafar, & Gani, 2016), particularly given the ever-increasing pervasiveness of smartphones and tablets that created a surge in the BYOD (Bring Your Own Device) tendency (Weeger, Wang, & Gewald, 2016).
- The Internet of Things (Botta, de Donato, Persico, & Pescapé, 2016), with an explosive development rate that raises the need to integrate software applications with the physical world, including sensor networks (Iyengar & Brooks, 2016). Al-Fuqaha, Guizani, Mohammadi, Aledhari, and Ayyash (2015) provide estimates that indicate that the number of Internet-capable, autonomous devices greatly outnumber human-operated devices, which means that the Internet is no longer dominated by human users, but rather by small computer-based devices that require technologies adequate to them, rather than to full-fledged servers.

The world is also increasingly dependent on computers, generating and exchanging more and more data, either at business, personal, or sensor levels. This raises the integration problem to a completely new level, in which conventional integration technologies (such as HTTP, XML, JSON, Web Services, and RESTful APIs) expose their limitations. These technologies were conceived initially for human interaction, with text as the main format and subsecond time scales, not for heavy-duty, machine-level binary data exchange. These new integration problems need new solutions.

Integration (Panetto & Whitman, 2016) can be broadly defined as the act of instantiating a given method to design or adapt two or more systems, so that they cooperate and accomplish one or more common goals. What these words really mean depends largely on the domain to which the systems belong, although there is a pervasive, underlying notion that these systems are active and reacting upon stimuli sent by others, in order to accomplish higher-level goals than those achievable by each single system.

To interact, applications must be interoperable, i.e., able to meaningfully operate together. *Interoperability* (Agostinho, Ducq, Zacharewicz, Sarraipa, Lampathaki, Poler, & Jardim-Goncalves, 2016) is a characteristic that relates systems with this ability and is defined by the 24765 standard (ISO, 2010) as the ability of two or more systems or components to exchange information and to use the information that has been exchanged. This means that merely exchanging information is not enough. Interacting systems must also be able to understand it and to react according to each other's expectations.

Interoperability is distinct from integration. Interoperability is a necessary but not sufficient condition for integration, which must realize the potential provided by interoperability. This is an inherently hard problem, since system interaction occurs at several levels of detail, from very low level (physical communication) to very high level (such as the purpose of the interacting parties to engage in an interaction).

Another problem is *coupling* (Bidve, & Sarasu, 2016), which provides an indication of how much applications are intertwined.

Interoperability and coupling are two facets of the same problem, application integration, and reflect two unfortunately conflicting goals:

- **Interoperability:** Applications need to interact to accomplish collaboration. This necessarily entails some form of mutual knowledge and understanding, but creates dependencies that may hamper the evolution (changes) of these applications.
- **Decoupling:** Applications should not have dependencies on others, in order to be able to evolve freely and dynamically. Unfortunately, independent applications do not understand each other and are not able to interact, which means that some form of coupling is unavoidable to achieve interoperability.