Improving the Integration of Distributed Applications

2

José Carlos Martins Delgado
https://orcid.org/0000-0002-2536-4906
Instituto Superior Técnico, Universidade de Lisboa, Portugal

INTRODUCTION

Complex software systems typically involve many interacting applications, with many decisions to take and many tradeoffs to evaluate, not only in each application but also in the ways in which the various applications interact. Ideally, applications should be completely decoupled, with no constraints on one another and completely independent lifecycles. This would allow separate development of each application and elimination of design and programming inefficiencies due to interaction between the specifications of applications, which usually cause iterations in requirements for other applications and consequent changes.

However, applications do need to interact and to cooperate, to collectively fulfill the goals of the system. Therefore, a fundamental tenet in software design is to reduce application coupling as much as possible (Bidve, & Sarasu, 2016) without hindering the interaction capabilities necessary to support the required application interoperability. Application decoupling has also the advantage of improving:

- **Changeability**: A change in one application is less likely to have a significant impact in other applications.
- **Adaptability**: Fewer constraints require less effort to adapt to changes in other applications.
- **Reusability**: An application with fewer requirements and constraints has an increased applicability range.
- **Reliability**: A smaller set of requirements simplifies the task of finding an alternative application in case of failure.

In general, the fundamental problem of application design, in terms of interaction, is how to provide (at most) the minimum coupling possible while ensuring (at least) the minimum interoperability requirements. Therefore, the main goal is to ensure that each interacting application knows just enough about the others to be able to interoperate with them but no more than that, to avoid unnecessary dependencies and constraints. This is an instance of the principle of least knowledge (Hendricksen, 2014).

The two most used application integration approaches, Service-Oriented Architecture (SOA) (Erl, Merson, & Stoffers, 2017) and Representational State Transfer (REST) (Fielding, Taylor, Erenkrantz, Gorlick, Whitehead, Khare, & Oreizy, 2017) hardly comply with the principle of least knowledge. They achieve interoperability but do not solve the coupling problem, since they require, in practice, that the schemas used by the interacting applications are the same.

SOA is good at modeling distributed systems based on the service paradigm (an extension the object-oriented paradigm for distributed applications) but involves rather complex and static service specifications and entails sharing schemas between client and server, which is a heavy form of coupling. Changing the

DOI: 10.4018/978-1-7998-3479-3.ch017

interaction between Web Services is not a trivial task. REST is much simpler, justifying its increasing popularity, but is rather low level and not the best match for general-purpose, behavior-oriented distributed applications. It also entails a high level of coupling, since it requires that both interacting applications share the same media type specification.

This article revisits the application integration problem with an open mind, without *a priori* restrictions from specific technologies, such as Web Services for SOA and RESTful APIs for REST. The only assumption is that there are applications that need to interact, by using messages. The idea is to base the interoperability mechanism on the concepts of *compliance* (Czepa, Tran, Zdun, Kim, Weiss, & Ruhsam, 2017) and *conformance* (Carmona, van Dongen, Solti, & Weidlich, 2018), which allow partial interoperability, rather than on sharing data schemas. This makes all the difference. Unlike Web Services, there are no declared schemas that a client is forced to use in their entirety. Unlike RESTful applications, the client and server do not have to agree on a specific data type. Each resource (dataset, message or distributed application) has its own schema, stemming from its service (interface) definition. Checking for interoperability between two resources (for example, between a message and the parameter required by an operation) is done structurally, component by component, recursively until primitive resources are found.

The rest of the article is structured as follows. It starts by establishing a simple interoperability framework, with the various abstraction levels at which interoperability must be established. After defining the fundamental problem of application integration, the two most common solutions, SOA and REST, are analyzed and compared, in particular in what coupling is concerned. Compliance and conformance are then presented and proposed as a solution to reduce coupling to the bare minimum required by the interacting applications.

BACKGROUND

Modern applications, designed and managed in a distributed way, need to interact and collaborate with an increasing scale, since systems are becoming more complex and diversified. Digital-based technologies, such as cloud computing (Varghese, & Buyya, 2018), mobile computing (Page, & Thorsteinsson, 2018), and the Internet of Things (Paul, & Saraswathi, 2017) became ubiquitous and disruptive. In the recent 4.0 trend (Dornberger, 2018), collaboration means generating and exchanging more and more data, either at business, personal, or sensor levels. The fourth industrial revolution, commonly known as Industry 4.0 (Liao, Deschamps, Loures, & Ramos, 2017), in which agile reconfigurability of the production supply chain is a fundamental objective, is just an example.

All this raises the application integration problem to a completely new level, in which conventional integration technologies expose their limitations and require 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. To interact, applications must be interoperable, i.e., able to meaningfully operate together.

The ISO/IEC/IEEE 24765 standard (ISO, 2010) defines *interoperability* (Agostinho, Ducq, Zacharewicz, Sarraipa, Lampathaki, Poler, & Jardim-Goncalves, 2016) as "the ability of two or more systems or components to exchange information and to use the information that has been exchanged". Therefore, merely exchanging information is not enough. Interacting applications must also be able to understand it and to react according to each other's expectations.

A related problem is *coupling* (Bidve, & Sarasu, 2016), which provides an indication of how much applications are intertwined and depend on each other. Some degree of coupling is unavoidable, since