Chapter 2

Lowering Coupling in Distributed Applications With Compliance and Conformance

José Carlos Martins Delgado

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

ABSTRACT

The interaction of applications in distributed system raises an integration problem that application-developing methods need to solve, even if the initial specifications change, which is actually the normal case. Current integration technologies, such as Web Services and RESTful APIs, solve the interoperability problem but usually entail more coupling than required by the interacting applications, since they share data schemas between applications, even if they do not actually exercise all the features of those schemas. The fundamental problem of application integration is therefore how to provide at most the minimum coupling possible while ensuring at least the minimum interoperability requirements. This chapter proposes compliance and conformance as the concepts to achieve this goal by sharing only the subset of the features of the data schema that applications actually use, with the goal of supporting a new architectural style, structural services, which seeks to combine the advantages of both SOA and REST.

DOI: 10.4018/978-1-7998-2142-7.ch002
Lowering Coupling in Distributed Applications With Compliance and Conformance

INTRODUCTION

A complex software system typically involves many interacting modules, with many decisions to take and many tradeoffs to consider, not only in each module but also in the ways in which the various modules interact. Object-oriented software design tries to minimize the semantic gap (Sikos, 2017) between a problem specification and the architecture of the software application that deals with that problem, by providing a close correspondence between the problem entities and the corresponding software modules (classes).

Ideally, classes should not have dependencies on others, avoiding constraints on one another and exhibiting completely independent lifecycles. This would allow separate development of each class and elimination of software design and programming inefficiencies due to interaction between the specifications of classes, which usually cause iterations in requirements for other classes and consequent changes.

However, classes do need to interact and to cooperate, to fulfill collectively the goals of the system. Therefore, a fundamental tenet in software design is to reduce class coupling as much as possible (Bidve, & Sarasu, 2016) without hindering the interaction capabilities necessary to support the required class interoperability.

Decoupling also translates into higher changeability (a change in a class is less likely to have a significant impact in other classes), higher adaptability (less constraints require less effort to adapt to changes in other classes), higher reusability (a class with less requirements and constraints has an increased applicability range) and higher reliability in a distributed context (a smaller set of requirements simplify the task of finding an alternative in case of failure).

Although decoupling constitutes a fairly obvious goal, as a means to reduce dependencies and constraints, tuning it to the right degree in practice is not an easy task. 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. This means that the main goal is to ensure that each interacting class 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 (Hendrickson, 2014).

Historically, interoperability has been the main goal in Web-based distributed systems, whereas decoupling has been one of the top concerns in software engineering, when developing an application, along with other metrics such as cohesion.

Software development methods emphasize decoupling, changeability and agility, which means structuring classes of an application so that a change somewhere affects the remaining classes as little as possible and an application developer can deal with it easily and in a short time. Interoperability between classes of a local
Related Content

Formal Specification and Implementation of Priority Queue using Stream Functions

A Novel Approach for Ontology-Based Feature Vector Generation for Web Text Document Classification
[www.igi-global.com/article/a-novel-approach-for-ontology-based-feature-vector-generation-for-web-text-document-classification/191205?camid=4v1a](www.igi-global.com/article/a-novel-approach-for-ontology-based-feature-vector-generation-for-web-text-document-classification/191205?camid=4v1a)
Knowledge Extraction from a Computational Consumer Model Based on Questionnaire Data Observed in Retail Service
www.igi-global.com/article/knowledge-extraction-computational-consumer-model/44685?camid=4v1a

Integration of Libre Software Applications to Create a Collaborative Work Platform for Researchers at GET
www.igi-global.com/chapter/integration-libre-software-applications-create/29547?camid=4v1a