The Journal of Systems and Software 111 (2016) 89–104

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

On the design of a maintainable software development kit to implement integration solutions

Rafael Z. Frantz [a,*], Rafael Corchuelo [b], Fabricia Roos-Frantz [a]

[a] Department of Exact Sciences and Engineering, Unijuí University, Rua do Comércio, 3000, Ijuí 98700-000, RS, Brazil
[b] ETSI Informática, University of Seville, Avda. Reina Mercedes, s/n, Sevilla 41012, Spain

ARTICLE INFO

Article history:
Received 5 July 2013
Revised 14 July 2015
Accepted 25 August 2015
Available online 16 September 2015

Keywords:
Enterprise Application Integration
Integration framework

ABSTRACT

Companies typically rely on applications purchased from third parties or developed at home to support their business activities. It is not uncommon that these applications were not designed taking integration into account. Enterprise Application Integration provides methodologies and tools to design and implement integration solutions. Camel, Spring Integration, and Mule range amongst the most popular open-source tools that provide support to implement integration solutions. The adaptive maintenance of a software tool is very important for companies that need to reuse existing tools to build their own. We have analysed 25 maintainability measures on Camel, Spring Integration, and Mule. We have conducted a statistical analysis to confirm the results obtained with the maintainability measures, and it follows that these tools may have problems regarding maintenance. These problems increase the costs of the adaptation process. This motivated us to work on a new proposal that has been carefully designed in order to reduce maintainability efforts. Guaraná SDK is the software tool that we provide to implement integration solutions. We have also computed the maintainability measures regarding Guaraná SDK and the results suggest that maintaining it is easier than maintaining the others. Furthermore, we have conducted an industrial experience to demonstrate the application of our proposal in industry.

1. Introduction

Companies rely on applications to support their business activities. Frequently, these applications are legacy systems, packages purchased from third parties, or developed at home to solve a particular problem. This usually results in heterogeneous software ecosystems, which are composed of applications that were not usually designed taking integration into account. Integration is necessary, chiefly because it allows to reuse two or more applications to support new business processes, or because the current business processes have to be optimised by interacting with other applications within the software ecosystem. Enterprise Application Integration provides methodologies and tools to design and implement integration solutions. The goal of an Enterprise Application Integration solution is to keep a number of applications' data in synchrony or to develop new functionality on top of them, so that applications do not have to be changed and are not disturbed by the integration solution (Hohpe and Woolf, 2003).

In the last years, several tools have emerged to support the design and implementation of integration solutions. Hohpe and Woolf (2003) documented many patterns found in the development of integration solutions. These patterns basically aim to support three core concepts, namely: pipes, filters, and resource adapters. Camel, Spring Integration, and Mule range amongst the most popular open-source tools that provide support for some of these integration patterns. Camel provides a fluent API (Fowler, 2010) that software engineers can use programmatically or by means of a graphical editor. In both cases, the integration solution is implemented using a Java, Scala, or XML Spring-based configuration files. Spring Integration was built on top of the Spring Framework container, and provides a command-query API (Fowler, 2010). This tool can be used programmatically or by means of a graphical editor. Integration solutions are implemented using either Java code or an XML Spring-based configuration file. The architecture of Mule got inspiration from the concept of enterprise service bus. Software engineers count on a command-query API (Fowler, 2010) to use this tool programmatically, or a workbench to design and implement integration solutions using a graphical editor. Integration solutions are implemented using either Java code or an XML Spring-based configuration file. In earlier versions, Mule supported a limited range of integration patterns; version 3.0 resulted in a complete re-design whose focus was on supporting the majority

* Corresponding author. Tel.: +55 5533320200.
E-mail addresses: rzfrantz@unijui.edu.br (R.Z. Frantz), corchu@us.es (R. Corchuelo), frfrantz@unijui.edu.br (F. Roos-Frantz).

http://dx.doi.org/10.1016/j.jss.2015.08.044
0164-1212/

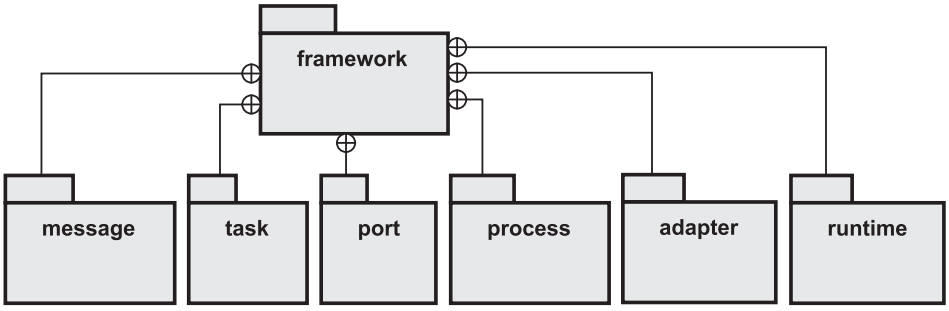

Fig. 1. Packages of which our framework is composed.

of integration patterns. As of the time of writing this article, Camel, Spring Integration, and Mule are at version 2.7.1, 2.0.3, and 3.1, respectively. In the rest of the article, we implicitly refer to these versions.

We are concerned with maintainability. According to IEEE (1990), maintenance can be classified as corrective, perfective, and adaptive. Corrective maintenance aims to repair software systems to eliminate faults that might cause them to deviate from their normal processing. Perfective maintenance aims to modify a software system, usually to improve the performance of current functionalities or even to improve the maintainability of the overall software system. Adaptive maintenance focuses on adapting a software system to use it in new execution environments or business processes.

In this article, we are interested in adaptive maintenance, which is very important for companies that need to reuse existing tools to build their own (Chen and Huang, 2009). Many companies rely on open-source tools that can be adapted to a specific context within their business domain. For example, a company that develops Enterprise Application Integration solutions may need tools that focus on specific contexts such as e-commerce, health systems, financial systems, and insurance systems to meet standards and recommendations like RosettaNet (2011), HL7 (2011), Swift (2011), and HIPAA (2011), respectively. Other authors have evaluated open-source tools from a performance point of view (García-Jiménez et al., 2010); we think that our work is complementary.

It is not new that the design and implementation of a software system has an impact on its maintenance costs (Epping and Lott, 1994; Jorgensen, 1995; Bergin and Keating, 2003; Schneidewind, 1987). International standards such as ISO 9126-1 (ISO/IEC, 2001) or more recent ISO 25010 (ISO/IEC, 2011) define quality models that help to understand what may have an impact on the maintainability of software systems. According to these standards, the maintainability of a software system can be influenced by the amount of effort to change the system (Changeability), the capability of a software to avoid collateral effects produced by changes on it (Stability), the ability to identify and diagnose failures (Analysability), and the effort to verify the software after changes (Testability). In both design and implementation, software engineers need to pay attention to readability, understandability, and complexity, since they are related to several subcharacteristics that characterise maintainability. Thus, the resulting models and source code must be easy to read and understand, because it is very common that the people who work on them shall not maintain them. The complexity of the algorithms should be kept low, not only for performance reasons, but because it makes it easier for a software engineer to follow their execution flows and debug them. Thus, to reduce the costs involved in the adaptation of a software system to a specific context, it is very important that the software system was designed taking into account issues that have a negative impact on maintenance.

How costly it is to maintain a tool depends on a variety of measurable properties. We have computed these measures on Camel, Spring Integration, and Mule, and the results do not seem promising enough. The focus of this article is only on the core implementation of these proposals, which have similar functionalities, since the core aim at providing support for the integration patterns documented by Hohpe and Woolf (2003). The results motivated us to work on a Software Development Kit (SDK) to which we refer to as Guaraná SDK.[1] The design decisions and the implementation of the core of Guaraná SDK had always maintainability in mind. The result is that its design provides better values for the maintainability measures regarding its core implementation, which suggests that its core is more maintainable and thus easier to adapt for a particular context than the core implementation of Camel, Spring Integration, or Mule. The core of our proposal also aims at providing support for the integration patterns. The core of Guaraná SDK is composed of two layers, namely: the framework and the toolkit. The former provides a number of classes and interfaces that provide the foundation to implement tasks, adapters, and workflows, as well as a Runtime System to which we deploy and run the integration solutions; the latter extends the framework to provide an implementation of tasks and adapters that is intended to be general purpose.

A six-page abstract regarding our results was presented in Frantz and Corchuelo (2012); in this article, we extend our preliminary paper as follows: we analyse 16 additional maintainability measures, we analyse an additional wide-spread open-source tool, Mule, we provide a statistical analysis based on Kolmorogov–Smirnov's test, Shapiro–Wilk's test, Iman–Davenport's test, and Bergmann–Hommel's test to confirm our intuitive conclusion from the results obtained with the maintainability measures, we provide a comprehensive description of each layer of Guaraná SDK, and we demonstrate our proposal by means of an industrial experience that has been developed in co-operation with a spin-off company. We have also developed a domain-specific language that is intended to facilitate designing integration solutions at a high level of abstraction (Frantz et al., 2011).

The rest of the article is organised as follows: Section 2 presents the framework layer of Guaraná SDK; Section 3 presents the toolkit layer; Section 4 presents the experimental study we conducted; Section 5 presents an industrial experience on which we have worked; finally, Section 6 reports on our main conclusions.

2. The framework layer

In this section, we describe the framework layer. Fig. 1 provides an overview of this layer by showing the six packages of which it is composed. In the following subsections we describe each package.

2.1. Messages

Messages are used to wrap the data that is manipulated in an integration solution. They are composed of a header, a body and one or more attachments, cf. Fig. 2.

[1] Guaraná technology is available at http://www.guaranasolutions.com.