Science of Computer Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

# Process-aware web programming with Jolie

Fabrizio Montesi [1]

*Department of Mathematics and Computer Science, University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark*

ARTICLE INFO

*Article history:*
Received 14 October 2014
Received in revised form 3 November 2015
Accepted 4 May 2016
Available online xxxx

*Keywords:*
Business processes
Programming languages
Sessions
Web services

ABSTRACT

We extend the Jolie programming language to capture the native modelling of process-aware web information systems, i.e., web information systems based upon the execution of business processes. Our main contribution is to offer a unifying approach for the programming of distributed architectures on the web, which can capture web servers, stateful process execution, and the composition of services via mediation. We discuss applications of this approach through a series of examples that cover, e.g., static content serving, multiparty sessions, and the evolution of web systems. Finally, we present a performance evaluation that includes a comparison of Jolie-based web systems to other frameworks and a measurement of its scalability.



## 1. Introduction

A Process-Aware Information System (PAIS) is an information system based upon the execution of business processes. These systems are required in many application scenarios, from inter-process communication to automated business integration [1]. Processes are typically expressed as structures that determine the order in which communications should be performed in a system. These structures can be complex and of different kinds; a systematic account can be found at [2]. For this reason, many formal methods [3–6], tools [7–11], and standards [12–14] have been developed to provide languages for the definition, verification, and execution of processes. In these works, compositionality plays a key role to make the development of processes manageable. For example, in approaches based on process calculi, complex process structures are obtained by composing simpler ones through the usage of standard composition operators such as sequence, choice, and parallel (see, e.g., [15]). Other approaches follow similar ideas using graphical formal models, e.g., Petri Nets [16,17].

In the last two decades, web applications have become increasingly process-aware. Web processes – i.e., processes inside of a web information system – are usually implemented server-side on top of *sessions*, which track incoming messages related to the same conversation. Sessions are supported by a shared memory space that lives through different client invocations. Differently from the aforementioned approaches for designing processes, the major languages and platforms for developing web applications (e.g., PHP, Ruby on Rails, and Java EE) do not support the explicit programming of process structures. As a workaround, programmers have to simulate processes using bookkeeping variables in the shared memory space of sessions. For example, consider a process in a Research Information Service (RIS) where a user has to authenticate through a `login` operation before accessing another operation, say `addPub`, for registering a publication. This would be implemented by defining the `login` and the `addPub` operations separately. The code for `login` would update a bookkeeping variable in the session state and the implementation for `addPub` would check that variable when it is invoked by the user. Although this approach is widely adopted, it is also error-prone: since processes can assume quite complex

*E-mail address:* fmontesi@imada.sdu.dk.
[1] Tel.: +45 65507171.

http://dx.doi.org/10.1016/j.scico.2016.05.002
0167-6423/

structures, simulating them through bookkeeping variables soon becomes cumbersome. Consequently, the produced code may be poorly readable and hard to maintain.

The limitations described above can be avoided by adopting a multi-layered architecture. For example, it is possible to stratify an application by employing: a web server technology (e.g., Apache Tomcat) for serving content to web browsers; a web scripting language (e.g., PHP) for programmable request processing; a process-oriented language (e.g., WS-BPEL [12]) for modelling the application processes; and, finally, mediation technologies such as proxies and ESB [18] for integrating the web application within larger systems. Such an architecture would offer a good separation of concerns. However, the resulting system would be highly heterogeneous, requiring a specific know-how for handling each part. Thus, it would be hard to maintain and potentially prone to breakage in case of modifications.

The aim of this paper is to simplify the programming of process-aware web information systems. We build our results on top of Jolie, a general-purpose service-oriented programming language that can handle both the structured modelling of processes and their integration within larger distributed systems [11,19]. Jolie is briefly introduced in § 2.

### 1.1. Contributions

Our main contribution is the extension of Jolie to obtain a unifying technology for the programming of processes, web technologies (web servers and scripting) and mediation services (e.g., proxies), which facilitates the development of heterogeneous information systems that involve the Web. We then investigate the applicability of this framework, showing that our extended version of Jolie captures the different components of web systems and their integration using a homogeneous set of concepts. We proceed as described below.

*Web processes* We extend the Jolie interpreter to support the HTTP protocol, enabling processes written in Jolie to send and receive HTTP messages (§ 3). The integration is *seamless*, meaning that the processes defined in Jolie remain abstract from the underlying data formats used in the Web: data structures in Jolie are transparently transformed to HTTP message payloads and vice versa (§ 3.1). These transformations can be configured using *port parameters*, an extension of the Jolie language that we develop to allow processes to map information from HTTP headers to application data and vice versa (§ 3.3). Parameters support mobility: they can be transparently transmitted from service registries to clients, allowing clients to be transparently configured at runtime with the right binding information (§ 3.4, Example 3.3).

*Web servers as processes* We develop a web server, called Leonardo (§ 4), using our approach. The web server is given as a simple process that (i) receives the name of the resource a client wants to access, then (ii) reads the content of such resource, and (iii) sends the content back to the client. Leonardo is an example of the fact that, in our framework, a web server is not a separate technology but it is instead a simple case of process. We also show how to extend Leonardo to handle simple CRUD operations over HTTP.

*Sessions* We combine our HTTP extension for Jolie with message correlation, a mechanism used in service-oriented technologies to route incoming messages to their respective processes running inside of a service [12,11]. We first show that this combination is adequate wrt existing practice: it enables Jolie processes to use the standard methodology of tracking client–server web sessions using unique session identifiers (§ 5.1). Then, we generalise such methodology to program multiparty sessions, i.e., structured conversations among a process and multiple external participants [20] (§ 5.2).

*Architectural programming* We present how to obtain separation of concerns in a web architecture implemented with our approach, by combining HTTP with aggregation, a Jolie primitive for programming the structure of service networks [21, 22,11] (§ 6). We demonstrate the usefulness of this combination by implementing a multi-layered system that integrates different components. We also discuss how to deal with the evolution of software architectures obtained with our approach (§ 6.2).

*RESTful services* We discuss how to develop web systems based on the REST style [23] with our framework, using URI templates to bridge resource-oriented interactions to processes (§ 7). The standard separation of concerns between routing and business logic can be achieved by developing a routing service that routes REST requests to other services, which required developing a reflection library for the Jolie language (§ 7.3). We then show how structured processes can be combined with our REST router to obtain RESTful multiparty sessions (§ 7.4). Since, in these scenarios, requests typically have to be validated both at client- and server-side, we provide an integration between Jolie and JavaScript to be able to run the same validation code, achieving the same de-duplication benefits as in frameworks based on JavaScript (§ 7.5).

*Performance* A performance evaluation of our approach is given (§ 8). This evaluation covers two main aspects. First, our data shows that our solution has comparable performance to that of other existing frameworks in the execution of basic tasks, such as offering static files or templated web pages. Second, we analyse the scalability of our approach wrt the number of services involved in the computation of responses to clients.

Please cite this article in press as: F. Montesi, Process-aware web programming with Jolie, Sci. Comput. Program. (2016), http://dx.doi.org/10.1016/j.scico.2016.05.002