Science of Computer Programming 87 (2014) 62–94

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Improving workflow modularity using a concern-specific layer on top of Unify

Niels Joncheere [*,1], Sebastian Günther [1,2], Ragnhild Van Der Straeten, Viviane Jonckers

Vrije Universiteit Brussel, Software Languages Lab, Pleinlaan 2, 1050 Brussels, Belgium

H I G H L I G H T S

- We present the Unify framework for uniform modularization of workflow concerns.
- We present and exemplify the framework's base language and connector mechanism.
- We present two concern-specific languages (CSLs) built on top of the framework.
- We propose a methodology for building CSLs on top of the framework.
- We validate the expressiveness, performance and scalability of our approach.

A R T I C L E   I N F O

Article history:
Received 21 July 2012
Received in revised form 4 November 2013
Accepted 5 November 2013
Available online 12 November 2013

Keywords:
Concern-specific languages
Domain-specific languages
Modularization
Separation of concerns
Workflow languages

A B S T R A C T

Workflows are a popular means of automating processes in many domains, ranging from high-level business process modeling to lower-level web service orchestration. However, state-of-the-art workflow languages offer a limited set of modularization mechanisms. This results in monolithic workflow specifications, in which different *concerns* are scattered across the workflow and tangled with one another. This hinders the design, evolution, and reusability of workflows expressed in these languages. We address this problem through the Unify framework. This framework enables uniform modularization of workflows by supporting the specification of all workflow concerns – including *crosscutting* ones – in isolation of each other. These independently specified workflow concerns are connected to each other using workflow-specific *connectors*. In order to further facilitate the development of workflows, we enable the definition of *concern-specific languages* (CSLs) on top of the Unify framework. A CSL facilitates the expression of a family of workflow concerns by offering abstractions that map well to the concerns' domain. Thus, domain experts can add concerns to a workflow using concern-specific language constructs. We exemplify the specification of a workflow in Unify, and show the definition and application of two concern-specific languages built on top of Unify.

1. Introduction

Workflow management systems [1] have become a popular means of automating processes in many domains (e.g., e-commerce [2], healthcare [3], bioinformatics [4]). A workflow is defined to be *the automation of a business process, in whole*

[*] Corresponding author.
E-mail addresses: njonchee@soft.vub.ac.be (N. Joncheere), sgunther@soft.vub.ac.be (S. Günther), rvdstrae@soft.vub.ac.be (R. Van Der Straeten), vejoncke@soft.vub.ac.be (V. Jonckers).
[1] Funded by Innoviris, the Brussels Institute for Research and Innovation, through the VariBru project.
[2] Present address: Senacor Technologies AG, Litfaßplatz 2, 10178 Berlin, Germany.

0167-6423/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.scico.2013.11.003

or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules [5]. A workflow is created by dividing a process into different activities, and by specifying the ordering in which these activities need to be performed. This ordering is called the *control flow perspective* [6], as it describes how control flows between the activities. Typically, control can be split into several branches and joined at a later time. This allows specifying parallelism and choice. On top of the control flow perspective, the *data perspective* [7] describes the data that is used and generated by the activities. Common workflow languages include WS-BPEL [8], BPMN [9], and YAWL [10]. Because workflows are well suited to representing processes, e.g., in a visual form, they are often used for communicating with *domain experts*. However, domain experts typically do not develop workflows by themselves, as specific technical knowledge is required for augmenting high-level process descriptions with low-level implementation details. *Workflow developers* fulfill this latter role and thus perform most of the workflow development.

Separation of concerns [11] is a general software engineering principle that refers to the ability to identify, encapsulate, and manipulate only those parts of software that are relevant to a particular concept, goal, or purpose. These parts, called *concerns*, are the primary motivation for organizing and decomposing software into manageable and comprehensible modules [12]. Realistic workflows consist of several concerns, which are connected in order to achieve the desired behavior. For example, the e-commerce workflow used throughout this paper consists of the order handling, payment, shipping, and reporting concerns, among others. However, if all of a workflow's concerns need to be specified in a single, monolithic workflow specification, it will be hard to add, maintain, remove or reuse these concerns. Although most workflow languages allow decomposing workflows into sub-workflows, this mechanism is typically aimed at grouping activities instead of facilitating the independent evolution and reuse of concerns. Moreover, the sub-workflow mechanism only provides for procedural decomposition of a workflow, which suffers from the "tyranny of the dominant decomposition" as identified by Tarr et al. [13]: concerns that do not align with the dominant, procedural decomposition end up scattered across the workflow and tangled with one another, and are thus called *crosscutting concerns* [14]. These problems have been discussed in related work by ourselves [15] and others [16–18], where they are mainly tackled using *aspect-oriented programming* for workflows. Nevertheless, the problems are not yet fully addressed by the proposed solutions.

The goal of our current solution is to facilitate independent evolution and reuse of *all* workflow concerns, i.e., not merely crosscutting concerns. This can be accomplished by improving the modularization mechanisms offered by the workflow language. We propose an approach called Unify which provides a set of workflow-specific modularization mechanisms that can be readily employed by a range of existing workflow languages. Unify facilitates specifying workflow concerns as separate modules. These modules are then composed using versatile connectors, which specify how the concerns are connected. The main contributions of Unify are the following [19]:

1. Existing research on modularization of workflow concerns is aimed at only modularizing crosscutting concerns [15,17,18], or at only modularizing one particular kind of concern, such as monitoring [20]. Unify, on the other hand, aims to provide a uniform approach for modularizing all workflow concerns.
2. Existing aspect-oriented approaches for workflows are fairly straightforward applications of general aspect-oriented principles: they only support the basic concern connection patterns (*before*, *after*, and *around*; cf. Section 2) that were identified in general aspect-oriented research and which are essentially sequential. Thus, they do not sufficiently consider the prevalence of other control flow patterns such as parallelism and choice in the context of workflows. Unify improves on this by allowing workflow concerns to connect to each other in more workflow-specific ways, i.e., the connector mechanism supports a number of dedicated concern connection patterns that recognize the specific characteristics of the workflow paradigm, and which are not supported by other approaches.
3. Unify is designed to be applicable to a range of concrete workflow languages. This is accomplished by defining its connector mechanism in terms of a general, extensible base language meta-model.
4. Unify defines a clear semantics for its modularization mechanism, which facilitates the application of existing workflow verification techniques. This semantics has been previously published elsewhere [21,19], so we will not consider this topic further in the current paper.
5. The Unify implementation can either be used as a separate workflow engine, or as a preprocessor that is compatible with existing tool chains.

In this paper, we extend the contributions of the Unify framework further by considering the perspective of domain experts. While the above contributions mainly benefit workflow developers, we wish to involve the domain experts in the development of their workflows. In order to achieve this, we add notations and abstractions on top of Unify that define *concern-specific languages* (CSLs) [22,23], which are a specific kind of *domain-specific languages* (DSLs) [24–26]. DSLs hide implementation details in order to facilitate their use by domain experts. A CSL is a DSL of which the domain is the specification of a specific family of concerns. CSLs offer, as their name implies, dedicated constructs for defining families of concerns. Thus, the CSLs defined on top of Unify hide much of the complexity of the underlying base language and connector mechanism.

A description of the Unify framework has been published previously [19]. The current paper extends the previously published paper by describing our data perspective (cf. Section 4.1.2), describing the latest version of our connector mechanism (which introduces support for *fragment* joinpoints as well as the notion of *internal connectors*; cf. Section 4.2), and describing how concern-specific languages can be built on top of Unify (cf. Section 5). We also provide a validation of our approach