Future Generation Computer Systems 55 (2016) 147–162

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

HyperFlow: A model of computation, programming approach and enactment engine for complex distributed workflows

Bartosz Balis

AGH University of Science and Technology, Department of Computer Science, Krakow, Poland

HIGHLIGHTS

- A model of computation and system for scientific workflows, HyperFlow, is proposed.
- HyperFlow aims at high development productivity of skilled programmers.
- The HyperFlow Model of Computation combines simplicity with high expressiveness.
- Complex workflow patterns can be implemented using a simple syntax.
- HyperFlow enables a fully distributed and decentralized workflow execution.

ARTICLE INFO

Article history:
Received 30 March 2015
Received in revised form
13 August 2015
Accepted 28 August 2015
Available online 26 September 2015

Keywords:
Scientific workflows
Process networks
Workflow programming
Workflow patterns
Workflow enactment

ABSTRACT

This paper presents HyperFlow: a model of computation, programming approach and enactment engine for scientific workflows. Workflow programming in HyperFlow combines a simple declarative description of the workflow structure with low-level implementation of workflow activities in a mainstream scripting language. The aim of this approach is to increase the programming productivity of workflow developers who are skilled programmers and desire a programming experience similar to the one offered by a mature programming ecosystem. Combining a declarative description with low-level programming enables elimination of shim nodes from the workflow graph, considerably simplifying workflow implementations. The workflow description is based on a formal model of computation (Process Networks) and is characterized by a simple and concise syntax, utilizing just three key abstractions—processes, signals and functions. Yet it is sufficient for expressing complex workflow patterns in a simple way. The adopted model of computation implemented in the HyperFlow workflow engine enables fully distributed and decentralized workflow enactment. The paper describes HyperFlow from the perspective of its workflow programming capabilities, the adopted model of computation, as well as the enactment engine, in particular its distributed workflow enactment capability. The provenance model and logging features are also presented. Several workflow examples derived from other workflow systems and reimplemented in HyperFlow are extensively discussed.

1. Introduction

Workflows have been widely adopted as a programming model for scientific applications. Structuring a scientific application as a workflow provides a convenient level of abstraction for expressing a scientific problem, shifts the complexity of parallel processing and its optimization to a workflow management system and facilitates provenance tracking and reproducibility [1,2].

Workflow programming is a crucial step in the scientific workflow lifecycle in which the workflow developer composes a workflow graph, i.e. specifies the workflow activities and their dependencies. The second aspect of workflow programming, which – contrary to graph composition – has not received sufficient attention in the literature, is the programming of workflow activities themselves. The majority of existing workflow development systems are aimed at domain scientists and attempt to hide the complexity related to low-level programming [1]. Consequently, the prevalent approach in such systems is that the user is given a predefined 'palette' of components which can be composed into a workflow. Adding new components to the palette is not straightforward, requires using proprietary mechanisms of a given workflow system, and cannot usually be done by the workflow developer (domain scientist).

This approach, typically based on visual programming, is perfectly valid for domain scientists with limited programming

E-mail address: balis@agh.edu.pl.

http://dx.doi.org/10.1016/j.future.2015.08.015
0167-739X/

skills and can be particularly effective for systems focusing on a particular discipline where workflows are relatively small and many reusable workflow components are already available, a representative example being the Galaxy workbench [3]. However, in many cases workflow development needs to involve experienced programmers. For example, workflows can be too large and complex to be composed by the scientists who instead interact with specialized science gateways where workflow instances are automatically generated [4]. A similar case occurs when workflow components need to be developed from scratch. In such cases, a workflow programming approach is desired which provides the programming capabilities similar to those of a mainstream programming language without forfeiting the advantages of a formal workflow model.

This paper presents HyperFlow: a model of computation, programming approach and enactment engine for complex distributed workflows. The novel contribution of HyperFlow concerns three areas: workflow modeling, programming, and execution.

1. Simple and expressive model of computation. HyperFlow provides a model of computation for workflows which combines simplicity with broad expressiveness. The model is inspired by Process Networks and is based on only three abstractions: processes, signals, and functions. Despite its simplicity, the model enables advanced programming capabilities and a rich set of workflow patterns that can be expressed with it, including patterns for processing of data collections, patterns for parallel processing, and various data and control flow patterns.

2. Innovative workflow programming approach. In HyperFlow, a workflow implementation involves a simple declarative description of the workflow graph, expressed in a JSON syntax, and implementations of workflow activities as JavaScript/`Node.js` functions. This allows workflow developers to preserve all the advantages of a graph model of parallel computation while leveraging benefits of low-level programming in a mainstream general-purpose language and its runtime environment. These benefits include the simplification of workflow implementations (e.g. by handling glue code in the low-level implementations of workflow activities which enables elimination of shim nodes), and gaining access to a mature programming ecosystem with many developers, reusable software packages and rich learning resources.

3. Lightweight workflow runtime environment. Finally, HyperFlow provides a workflow enactment engine, implemented as a lightweight `Node.js` [5] application. The engine implements the HyperFlow model of computation end enacts the workflow, but the actual execution of workflow activities only requires the `Node.js` runtime environment which makes their implementation reusable outside the HyperFlow context. The workflow execution model of the HyperFlow engine is inherently distributed and decentralized. The engine provides a REST API which enables remote workflow management and delivers the capability to execute workflows in a fully distributed and decentralized manner where multiple engine instances cooperatively enact a workflow, without centralized control.[1]

This paper is a substantial extension of the previously published conference paper [6]. Section 2 overviews related work. Section 3 extensively describes the HyperFlow workflow model, programming approach and various programming capabilities, including the implementation of representative complex workflow patterns. Section 4 discusses the model of computation implemented in HyperFlow as well as its provenance model. In Section 5, the HyperFlow enactment engine is presented. Section 6 studies example workflows implemented in HyperFlow. Finally, Section 7 concludes the paper.

2. Related work

Two main approaches to workflow programming in existing workflow systems are the ones based on, respectively, visual or textual graph composition, and custom-designed dataflow scripting languages. In the former approach, the developer composes the workflow by selecting workflow activities from the available palette of components, configuring them, and connecting their inputs and outputs. The components in the palette can represent particular domain-specific procedures related to a specific discipline (life sciences, astronomy, Earth sciences, etc.), but they can also provide control flow constructs (e.g. loops and conditional statements), data manipulation operations (e.g. string processing, array operations and data transformations), or other useful high-level functions (HTTP or Web Service client, file system operations, math functions, etc.). Many workflow systems follow this general approach while using different names for the workflow components, for example *activities* in ASKALON [7], *actors* in Kepler [8], *services* in Taverna [9], and *tools* in Triana [10].

Adding new components to the existing palette is usually possible but it requires using proprietary mechanisms of a given workflow system. For example, in ASKALON workflows are composed of abstract activities which are mapped to concrete executable deployments. Consequently, all activities (even simple ones such as those handling data conversions) need to be semantically described and registered in an activity registry. In Kepler, new actors can be programmed in Java by implementing a specific interface in order to conform to the general actor lifecycle. For example, in [11] a set of new actors needed to be implemented in order to support submission and monitoring of jobs from Kepler workflows to a cloud infrastructure. In CLAVIRE [12], software packages need to be described in detail using a domain-specific language *EasyPackage* in order to be invokable from a workflow. In WS-VLAM, workflow activities are implemented as python modules that need to adhere to a specific interface [13]. Such mechanisms, while attempting to unify the way diverse software components are invoked from a workflow, ultimately remain proprietary solutions that the developers are forced to use, resulting in extra software development effort, decreased interoperability, and problems related to software maintenance.

In Taverna, workflows are developed by composing components (called 'services') in a graphical editor. One of the supported service types is a *beanshell* service which encapsulates an arbitrary piece of Java code. Workflow implementations are typically broken down into two types of services:

- reusable *domain services* with stable interfaces; in practice these are implemented as Web services ("Taverna workflows essentially specify Web service compositions" [14]);
- non-reusable *shim*[2] services with ad hoc interfaces that act as adaptors/connectors between "domain" services and are implemented on the basis of beanshell services.

Wrapping glue code as explicit nodes in the workflow graph has a number of disadvantages. Shim nodes introduce noise which obscures workflow presentation and its provenance trace, because the user is not interested in trivial transformations in the scientific pipeline. A solution presented in [15] proposes special annotations provided by the user or the developer in order to distinguish interesting and uninteresting workflow activities. Obviously such

[1] The HyperFlow engine is available as open-source software at https://github.com/dice-cyfronet/hyperflow.

[2] Shim node is a workflow activity which does not perform a scientific procedure, but serves as a connector/glue between other domain-specific workflow activities.