Future Generation Computer Systems 26 (2010) 245–256

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Three fundamental dimensions of scientific workflow interoperability: Model of computation, language, and execution environment

Erik Elmroth, Francisco Hernández *, Johan Tordsson

Department of Computing Science, Umeå University, SE-901 87 Umeå, Sweden
HPC2N, Umeå University, SE-901 87 Umeå, Sweden

ARTICLE INFO

Article history:
Received 12 March 2009
Received in revised form
28 July 2009
Accepted 12 August 2009
Available online 21 August 2009

Keywords:
Scientific workflows
Workflow interoperability
Workflow languages
Model of computation
Grid interoperability

ABSTRACT

We investigate interoperability aspects of scientific workflow systems and argue that the *workflow execution environment*, the *model of computation (MoC)*, and the *workflow language* form three dimensions that must be considered depending on the type of interoperability sought: at the *activity*, *sub-workflow*, or *workflow* levels. With a focus on the problems that affect interoperability, we illustrate how these issues are tackled by current scientific workflows as well as how similar problems have been addressed in related areas. Our long-term objective is to achieve (*logical*) interoperability between workflow systems operating under different MoCs, using distinct language features, and sharing activities running on different execution environments.

© 2009 Elsevier B.V. All rights reserved.

1. Introduction

To date, scientific workflow systems offer rich capabilities for designing, sharing, executing, monitoring, and overall managing of workflows. The increasing use of these systems correlates with the simplicity of the workflow paradigm that provides a clear-cut abstraction for coordinating stand-alone activities. With this paradigm, scientists are able to concentrate on their research at the problem domain level without requiring deep knowledge of programming languages, operating systems, arcane use of libraries, or hardware infrastructure. In addition, the ease by which scientists can describe experiments, share descriptions and results with colleagues, as well as automate the recording of vast amounts of data, for example, provenance information and other data relevant for reproducing experiments, have made the workflow paradigm the fundamental instrument for current and future scientific collaboration.

Currently, there are many sophisticated environments for creating and managing scientific workflows that have also started to incorporate capabilities for using powerful grid resources. Although similar in many respects, including domains of interest and offered capabilities, existing workflow systems are not yet interoperable. Rather than discussing if workflow systems are completely interoperable or not at all, here we argue that the methods and techniques required to make systems interoperable depend on the type of interoperability sought.

Consequently, our main contribution is the introduction of three dimensions: *workflow execution environment*, *model of computation (MoC)*, and *workflow language*, that classify the problems that must be addressed depending on whether interoperability is sought at the activity, sub-workflow, or workflow levels. Our analysis of the dimensions leverages research from the areas of theory of computation, compiler optimization, and (visual) programming languages. We investigate differences in the execution environments for local workflows, and those executing on remote (grid) resources. We also study the implications of selecting an MoC, including the repercussions of choosing between the *control-driven* and *data-driven* styles of representing workflows, as well as methods for converting between both representations. Another contribution is the analysis of language aspects relevant for scientific workflows, including iterations and conditions, as well as the consequences of having a type system associated to the workflow language.

The rest of the paper is organized as follows. Section 2 investigates the reasons why interoperability in workflows is desired. From this motivation the interoperability problem is sub-divided into three distinct levels that are better addressed by focusing on the three proposed dimensions. At the end of this section we explain and discuss why those three dimensions must be considered. Section 3 describes the execution environment dimension

* Corresponding author at: Department of Computing Science, Umeå University, SE-901 87 Umeå, Sweden. Tel.: +46 0 90786 99 13.
E-mail addresses: elmroth@cs.umu.se (E. Elmroth), hernandf@cs.umu.se (F. Hernández), tordsson@cs.umu.se (J. Tordsson).

0167-739X/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2009.08.011

with particular focus on differences between local and remote (grid) environments. The differences in the respective MoCs commonly employed for local and grid workflows as well as a study of control-driven and data-driven workflows are presented in Section 4. Section 5 focuses on workflow language related issues including language constructs such as conditions and iterations in light of their programming languages counterparts. Finally, in Section 6, we present our conclusions, followed by acknowledgments and a list of references.

2. Workflow interoperability

There are many scientific workflow systems currently in use; see, e.g., [1–10]. Several of these have been developed successfully within interdisciplinary collaborations between domain scientists, the *end-users*, and computer scientists, the *workflow engineers*. Some of these systems target a particular scientific domain (e.g., [5]) while others cover a range of fields (e.g., [1–4,6,8,10]). Furthermore, some of these systems have been designed to optimize the use of grid resources [1,3,6,8,10], while others are more apt for desktop interfaces and dedicated to single users [2,4,5].

The existence of such a wide range of workflow systems is comparable to the large number of programming languages available. In both cases solutions can be general purpose or tailored for specific domains, and the choice of one over the other depends not only on the problem at hand but also on personal preferences. Moreover, it is neither possible to have one solution suitable for all problems and preferred by all users, nor likely for a new solution to emerge and replace all existing ones. Yet, unlike programming languages in which interoperability is achieved at the binary or byte code level and calls for source-to-source interoperability have long faded away, achieving interoperability between workflow systems is a venture of high priority.

For this work, we consider a workflow to be a direct graph where the nodes of the graph represent *workflow activities* and the *links* represent interaction between activities. Workflow activities are the execution units in workflows, and can be either atomic entities or sub-workflows composed of other activities. Thus, a workflow can be structured as a hierarchical graph composed of sub-workflows and activities. *Ports* serve as a mechanism to communicate between activities. Input ports provide the input to an activity whereas output ports store the output generated by the activity. Output ports are connected to input ports to specify the direction in which the interaction between activities is carried out. We can also consider activities to be functions whose domains are given by the cross product of the input ports and whose ranges are given by the cross product of the output ports. A *workflow engine* is a software module that selects and executes activities, specified by a workflow description, a process commonly known as *enactment*. A *workflow language* is used to encode the workflow descriptions that are read by the engine during enactment. In this work we focus exclusively on interoperability amongst workflow engines.

Table 1 presents a summary of the characteristics of workflow engines and workflow activities depending on the location where they execute. A workflow engine can execute in a local machine or it can do so at a remote resource. The remote case is useful when processing long-running activities, as client tools can reconnect to the engine for monitoring and managing purposes without requiring permanent connections. When executing locally, the engine is used by a single user and it executes in the user's desktop machine, whereas the engine is accessible by many users when executing remotely. In the latter case, the engine is commonly exposed as a permanently available service.

Workflow activities can be executed in the machine where the engine runs, which need not be the user's desktop machine, or on remote resources. When activities execute in the same machine as the engine they are often tightly coupled to the engine, and the combination is offered as a coherent environment. In this case, activities are used by a single user whereas these tend to be re-usable by many users when deployed at remote resources. Remote execution can be achieved either by employing standardized interfaces (e.g., using Web services) or by employing grid middleware.[1] Thanks to their typed interfaces, Web services enables the use of remote activities in a manner very similar to the one employed when activities execute locally. This is not the case for activities that access grid middleware directly. These activities are standalone command line applications configured by environment variables or command line arguments and are executed batch style. Notice that the command line application execution style offered by grid middleware can also be exposed as a Web service [11]. We treat workflows accessing services and those coordinating local activities equally in most of this work. However, we pay special attention to workflows using grid middleware and we refer to them as *grid workflows*.

Table 1
Summary of characteristics of workflow engines and workflow activities depending on the location in which they run.

	Engine	Activities
Local (personal use)	User's desktop machine.	Machine where engine runs.
Remote (multiple users)	Engine located in server.	Accessed via service interfaces or (grid) middleware.

In the remainder of this section we first present some motivations for workflow interoperability. From these motivations we identify that interoperability can be sought at the activity, sub-workflow, or complete workflow level. We finally argue that interoperability, at those three levels, can be better addressed if the problem is tackled from three orthogonal dimensions.

2.1. A case for interoperability

It has recently been suggested [12] that end-users are not really pressing for interoperability among workflow systems. The rationale behind this suggestion is that many contemporary systems are developed in tight coordination between end-users and workflow engineers. Hence, instead of using other systems that already offer the desired functionalities, new features are added when needed. This rationale leads to duplication of efforts and misutilizes resources that could otherwise be employed in more productive endeavors. It is furthermore mainly beneficial for researchers who are involved in this development loop. Yet, it is also the case that users may not be interested in *full interoperability* between workflow systems.[2]

Instead, a less ambitious degree of interoperability is often sought. For example, several workflow systems have been developed for operation within a specific scientific domain; thus, it is expected that users from different scientific fields, or in some cases from different research groups, use different workflow systems. The magnitude of current scientific problems has provoked the formation of large interdisciplinary collaborations in which scientists from different fields contribute to a final solution. It is very difficult to modify systems just to enable these collaborations. Furthermore, users are more comfortable working in environments familiar to them, and adapting to a different system may

[1] In practice any middleware of RPC mechanism, but for our purpose we consider only grid middleware.

[2] Full interoperability is commonly defined as the ability for clients (be they human users or software components) to *seamlessly* use the full functionality provided by the different interoperable systems in a totally transparent manner [13].

Download English Version:

https://daneshyari.com/en/article/426249

Download Persian Version:

https://daneshyari.com/article/426249

Daneshyari.com