The Demand for Consistent Web-Based Workflow Editors

Sandra Gesing*
University of Notre Dame, Ctr for Research Computing & University of Edinburgh, Informatics Forum
111 Information Technology Center
Notre Dame, IN 46556, US
sandra.gesing@nd.edu

Malcolm Atkinson, Iraklis Klampanos, Michelle Galea
University of Edinburgh, Informatics Forum
Edinburgh EH8 9AB, UK
{malcolm.atkinson, iraklis.klampanos, michelle.galea}@ed.ac.uk

Michael R. Berthold
Universität Konstanz
FB Informatik & Informationswissenschaft
Box 712
78457 Konstanz, Germany
Michael.Berthold@uni-konstanz.de

Roberto Barbera, Diego Scardaci
Italian National Institute of Nuclear Physics
95123 Catania, Italy
{roberto.barbera, diego.scardaci}@ct.infn.it

Gabor Terstyanszky, Tamas Kiss
University of Westminster
115 New Cavendish Street, London W1W 6UW, UK
terstyg@wmin.ac.uk, t.kiss@westminster.ac.uk

Peter Kacsuk
Laboratory of Parallel and Distributed Systems
MTA SZTAKI
Kende Street 13-17, 1111
Budapest, Hungary
kacsuk@sztaki.hu

ABSTRACT

This paper identifies the high value to researchers in many disciplines of having web-based graphical editors for scientific workflows and draws attention to two technological transitions: good quality editors can now run in a browser and workflow enactment systems are emerging that manage multiple workflow languages and support multi-lingual workflows. We contend that this provides a unique opportunity to introduce multi-lingual graphical workflow editors which in turn would yield substantial benefits: workflow users would find it easier to share and combine methods encoded in multiple workflow languages, the common framework would stimulate conceptual convergence and increased workflow component sharing, and the many workflow communities could share a substantial part of the effort of delivering good quality graphical workflow editors in browsers. The paper examines whether such a common framework is feasible and presents an initial design for a web-based editor, tested with a preliminary prototype. It is not a *fait accompli* but rather an urgent rallying cry to explore collaboratively a generic web-based framework before investing in many divergent individual implementations.

Keywords

H.5.3 Group and Organization Interfaces—Information Systems, web-based workflow editors, workflow composition, workflow languages and concepts, workflow interoperability

*Corresponding author

WORKS13 November 17, 2013, Denver, CO, USA.

ACM 978-1-4503-2502-8/13/11 ...$15.00
http://dx.doi.org/10.1145/2534248.2534260.

1. INTRODUCTION

In this paper we argue the case for a community effort to define, develop and support a *web-based generic* workflow editing system for scientific and data-intensive applications. This is timely because of emerging technological trends:

1. recent advances in W3C standards mean that it is now feasible to provide easily accessible good quality graphical editors in browsers [10]; and
2. systems that are capable of handling and enacting workflows written in multiple workflow languages are now available [50].

The investment will be worthwhile because of three main reasons. Firstly, there is a growing use of workflows, particularly in research, as a means of making methods repeatable, enabling their incremental improvement, and allowing methods to be shared, re-used, repurposed or validated. Secondly, there is also a proliferation of workflow languages, as a result of contemporaneous research, targeting various communities and different enactment models. Lastly, there is not yet a standard underpinning scientific workflow languages in sight, which could be processed via a wide range of different workflow systems. The development of such a standard and its integration in workflow systems would necessitate a vast amount of work on each supporting workflow system. Consequently, researchers will benefit from adapting and combining methods that are encoded in different workflow languages; we call the combined workflow a 'multi-lingual meta workflow'. Projects, such as SHIWA and ER-Flow, deliver multi-lingual meta-workflow enactment [50] (currently >10 workflow systems with submission to many Distributed Computing Infrastructures (DCIs)). The authors represent the ER-Flow, KNIME, and Dispel workflow systems and the Catania Science Gateway Framework and have user communities that require web-based editors.

Many of today's workflow systems use editors implemented as applications that have to be installed on users' workstations. This inhibits uptake as potential users may be reluctant or unable to install and manage such software, and rarely do it for more than one workflow language. Conversely, there is rapid uptake of facilities made available via browsers, where the explicit installation of software and many security issues are avoided. Developing good quality editors that run in the full gamut of popular browsers on a range of devices is a major undertaking. The following benefits will be obtained by developing a generic system, capable of handling and authoring a range of workflow languages.

1. Users will be better able to transfer skills between workflow languages, and to edit multi-lingual meta workflows within the one editor.
2. The common framework will encourage convergence of concepts, leading to greater ability to share workflows, components and libraries and methods.
3. Development costs of versatile and effective web-based editors will be amortised across workflow communities increasing the sustainable quality of editing systems.

A critical mass of implementation and adoption is necessary to achieve these benefits. That depends on a framework that accommodates a sufficient spectrum of workflow languages and has sufficient commonality that the consistency and amortisation materialises. We believe that by drawing on previous work we can initiate such a framework. The matter is urgent as we know that many workflow language developers are beginning work on web-based editors specific to their language. Once those are well developed the opportunity for amortising development will be lost. Once they are in common use it will be harder to introduce an alternative as users will have become accustomed to their particularities. This is therefore an urgent rallying cry for collaboration across research communities to drive an open-source project that will rapidly achieve that critical mass.

The framework adopts the model-view-controller (MVC) pattern for interactive systems. Here the *model* captures the properties of each workflow language, each community's mechanisms for sharing, such as access to registries of services and data, and the details of each workflow instance. The properties of a language will be specified once per language by a specialist in that workflow language, and the sharing mechanisms will be shaped and pre-populated at that stage. The *view* provides a manipulable visualisation of the model, e.g., of a particular workflow instance that is being created, edited or submitted. The *controller* contains parameters that govern the transformations between the model and the view. In part, it is set by workflow language experts as they install their language in the framework, so that the familiar look-and-feel encourages users to adopt the web-based system. In part, it is set by user preferences, e.g., determining which aspects of a workflow are visible, how nesting, scale and complexity are managed, as well as conventional control of sharing, colour, authorship, etc.

The web-based generic editor will support a number of capabilities for typical users, including import and export of workflows; copying, creation, editing, saving and discard of mono-lingual and multi-lingual workflows; management of a workflow's lifecycle; and interaction with registries for sharing and use of predefined libraries of components. The scope of the generic workflow editor should be carefully limited to achieve the 80 : 20 trade-off, 80% of benefits for 20% of the effort and complexity. For example:

1. quirky details in existing editors cannot be replicated;
2. the editor will not 'understand' a workflow's semantics – only selected composition constraints; and
3. hence, translation between workflow languages will not be attempted.

There is an additional caveat—the interfaces with enactment services and security still require analysis. However, a minimal *submit-run-collect-results* model will need to be supported immediately and more sophisticated incremental monitored enactments will be needed for debugging and dynamic control.

Section 2 illustrates the driving technological trends and the current breadth of workflow use and implementation. Section 3 introduces the framework which is capable of supporting the required generic functionality. We concentrate mainly on the model but show how the view and controller can be derived from this in a straightforward way. Section 4 presents work-in-progress on `GeWWE`, a prototype of the proposed generic GUI framework, and illustrates how this generic framework can be used for several workflow languages. We conclude with a summary of contributions and our vision as to how the workflow community can convert this idea into a widely used generic workflow editing system. The appendices and a web-page, `bit.ly/WBWFE`, provide readers with more detail including the full schema and invite discussion of the way forward.

2. BACKGROUND

This section introduces the terminology used in the paper, describes a workflow lifecycle, from inception to execution, and highlights our focus—that of workflow definition, and the tools used for that purpose. It also touches on the increasing requirement for interoperability between workflows and workflow systems, and the pivotal role that registries of workflows and workflow components play in this.

2.1 Workflow concepts and lifecycle

A *scientific workflow* is a set of interrelated computational and data-handling tasks designed to achieve a specific goal. It is often used to automate processes which are frequently executed, or to formalise and standardise processes. A workflow may be used to define and run computational experiments or to conduct recurrent processes on observational, experimental and simulation data. Scripting languages and graphical notation may be used to represent the tasks in a workflow, and the dependencies between them—this is discussed in more detail in Section 2.2.

A *Workflow Management System (WfMS)* is a software system that facilitates the management of workflows from their initial definition to their enactment. A WfMS enables the exploration and analysis of scientific data by enabling the quick (re-)design of experiments defined as workflows; and, by providing easy selection and integration in a workflow of the required resources – data, algorithms and computation.

The major components of a WfMS correspond to the different phases of a workflow lifecycle depicted in Figure 1. Different WfMSs implement the four phases of a workflow lifecycle listed below with varying degrees of sophistication.

Workflow composition – the specification of the tasks and

their interrelationships that denote the workflow, using graphical or text-based editors. If generic elements or partial specifications have been used, the result is called an '*abstract*' workflow. '*Components*', the building blocks of workflows, are made available (stored and described) through '*registries*' or '*repositories*' (interchangeable terms). Workflows may be used as conventional components, as composite building blocks for multi-lingual meta workflows, or as an initial template for a new workflow.

Resource mapping – the (sometimes automated) identification of specific versions of tasks and allocation of resources to those tasks to be used during enactment, resulting in a '*concrete*' workflow. '*Resources*' include data sources, data storage and computational facilities, such as a laptop, desktop, local cluster, grids, clouds and high-performance computing (HPC) centres. They also include application code and web services. They are listed in '*registries*' and may be selected during workflow composition.

Execution and monitoring – of a concrete workflow. This comprises deployment of the relevant code and data onto computational resources, their activation, and the collection of results. Successful completion of one subset of tasks may trigger further deployment and activation of tasks that were awaiting their results. Streaming enactment models deploy all of the tasks for concurrent activation. The software organising this is often called a '*workflow execution engine*'.

Provenance capture – from each stage of the workflow lifecycle enables provenance information to be made available to aid workflow and component re-use, to optimise resource allocation, job scheduling and exception handling.

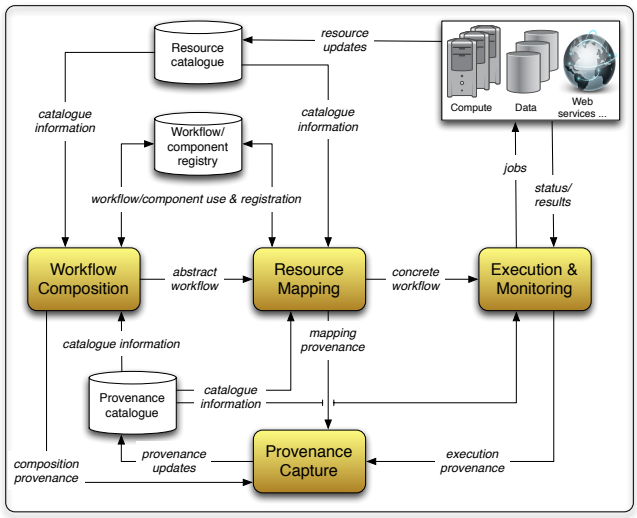


Figure 1: Workflow lifecycle adapted from [17].

Our work on web-based generic workflow editors currently focusses on workflow composition and resource mapping.

2.2 Workflow composition and languages

WfMSs often provide a graphical user interface to help users define their workflows by composing graphs, with nodes in a graph generally representing the tasks in a workflow, and edges between nodes representing dependencies between tasks.

The graphical notation used by an editor varies from system to system, from relatively simple directed acyclic graphs (DAGs), to more expressive notations, such as Petri nets and UML activity diagrams, that can depict choice, iteration and concurrency in a workflow. Taverna's workbench [31], for instance, offers the facility to design a workflow in the form of a DAG, Grid-Flow's [28] editor uses Petri nets, and Askalon [19] uses UML activity diagrams. KNIME supports DAG-like workflows with the possibility of partial execution to test as you build [11].

The languages used to capture these graphical representations, or used directly by a user via a textual editor to define a workflow, also vary. Many are XML-based, and include Taverna's t2flow, Pegasus' [37] DAX, and Askalon's AWDL; Grid-Flow's GFDL, used to describe data, programs and processes, is a declarative language that borrows heavily from SQL, while Meandre's [2] scripting ZigZag language is modelled on Python.

These workflow descriptions are often further compiled into another language, one used to communicate with the underlying execution engine for deployment of a workflow. For instance, ZigZag files are compiled to `.mau` (Meandre archive unit) files which are executed by a Meandre engine, and Pegasus' DAX files are transformed into `.dag` (DAGMan) files for execution on a Condor DAGMan [14] engine. Removing one layer of communication is Dispel [4,42], a language used for both describing an initial workflow defined by a user, and for communicating with an execution engine. KNIME directly interprets its XML representation.

It is not always the case that a WfMS offers both appropriate textual/graphical editors, and a suitable execution environment for the problem in hand. Hence, there have been a few attempts to mix-and-match between systems' composition tools and execution engines. For instance, Pegasus allows the creation of a DAX file via direct XML definition, or DAX generation APIs in Java, Perl and Python (a simple Pegasus GUI exists but is generally used for system demo purposes only). This approach is not necessarily the easiest one, however, for creating workflows consisting of hundreds of computations to be executed over distributed environments; Wings' [23] graphical interface has therefore been used to compose such workflows, and then pass them for distributed execution onto a Pegasus engine, or Apache OODT, a software architecture for providing access to distributed resources [52]. Similarly, Triana's graphical environment [49] was extended for the GriPhyN and Open-Science Grid projects to allow the generation of Pegasus execution files [48]; and, Apache Airavata's [41] graphical client, called XBaya, can also be used to export defined workflows into other languages such as Taverna's t2flow and Pegasus' Condor DAGMan.

In general and increasingly, the proliferation of different WfMSs with their proprietary workflow languages and graphical and textual editors, coupled with a lack of standardisation of scientific workflow models, presents a challenge to the sharing of knowledge and experience in the form of workflows. A generic web-based workflow editor catering for different workflow languages and for multi-lingual meta-workflow editing, with the versatility to facilitate existing working practices, is a step towards increased sharing. This effect will be increased if the collaboration in developing a generic editor leads to consistency in the ways in which workflow languages and registries are accessed.

2.3 Workflow interoperability

The challenge of an ever growing diversity of workflow languages has long been recognised, see for example [16, 22]. A range of models motivate fundamental differences, e.g., differentiating between master-worker models where the workflow execution engine is the focus and controller – P-GRADE/ gUse [20] for example, versus peer-to-peer models which achieve orchestration as an emergent behaviour – Dispel for example. Other diversity emerges from implementation strategies, e.g., those that assume a common file infrastructure from those that do not. There is also significant variation in the level of abstraction, the support for the four phases of Figure 1 and linguistic style. The greatest cause of diversity is that many of the workflow languages started in a particular discipline and with communities that already adopted particular data services and programming languages. Those systems often develop a large investment in primitive components for their community and a significant collection of workflows supporting current working practices. Analysis of many existing workflow systems, in science and business, has yielded 40 control-flow patterns, 40 data patterns and 43 resource patterns [53]. A generic editor cannot directly cover such diversity and we therefore introduce a *category* level in the conceptual model that permits each workflow language to specify the patterns it supports— see Section 3.

There will remain strong forces for multiple workflow languages for the foreseeable future. The infeasibility of moving communities onto new technologies because of their intellectual investment collides with the high-cost of rebuilding the libraries of components and workflows. This comes to the fore when experienced people move communities or when researchers combine methods from different communities.

To address this confrontation Elmroth et al. identified common structures in workflow systems that could underpin integration or translation across workflow languages [18]. They identified three dimensions of variation: workflow-execution environment, model of computation, and workflow language, and three levels at which these should be considered: activity, sub-workflow and workflow, thereby partitioning the overall challenge into manageable parts. The SHIWA project investigated two strategies [35, 40] for workflow integration:

1. develop a catalogue of the functional elements of workflow languages and translate via this between languages;
2. provide a common management and enactment environment for a set of workflow languages and provide for enactments that use more than one.

The latter strategy led to an effective system that today supports more than ten workflow languages with the help of the ER-Flow project [50]. A workflow engine for multiple languages could sit behind a multi-lingual graphical workflow editor and the ER-Flow project is involved via the authors.

2.4 Workflow sharing and registries

Workflow-based systems/infrastructures require repositories or registries from which to retrieve components of interest for execution, modification or study. Depending on the requirements around which different systems are designed, such repositories may contain workflow building components, workflows, information about resources, required data, etc. In most cases, a workflow system, and from the user's perspective its editor, is expected to interact with multiple repositories or registries, seamlessly.

There are a number of workflow systems for e-Science, each with their design decisions and target audiences. Here, we briefly outline a few examples of their repositories. A web-based platform for importing and sharing workflows as well as derivative or related digital research objects is myExperiment [47]. As a sharing platform, myExperiment also relies on external repositories, most notably BioCatalogue [12] and recently others, through the myGrid collaboration [24]. Even though not dictated by design, myExperiment is best integrated with the Taverna workflow system [31], which in turn integrates processing elements or processes as web services. BioCatalogue is a catalogue of bioinformatics-related web services, which can be used when composing Taverna (or other, e.g., KNIME) workflows and can be viewed and interacted with from workflow editors.

Two repositories representing different approaches to workflow management that cover different stages of a workflow's lifecycle are SHIWA [50] and Wf4Ever [9]. SHIWA addresses the challenge of workflow interoperability, supporting workflows from different environments. Users make use of SHIWA services, including a central repository of workflows created by different WfMSs, to compose and execute meta-workflows through the SHIWA portal. Wf4Ever on the other hand focuses on workflow preservation and digital-experiment reproducibility. Wf4Ever exposes APIs allowing access to the digital object store, a generic store including workflows and related services, such as recommendations and workflow transformation.

SCI-BUS [35] offers a repository with a user-interface-oriented approach. The repository presents gUSE workflows wrapped in portlets deployable in portal frameworks based on the JSR168/JSR286 [1, 45] standard. Internally, the lifecycle of a workflow is managed via a portlet. Its users are supported with user interfaces tailored to its application.

The repositories and systems discussed above take a higher-level view of workflow composition, as registrable components are generally assumed to be readily enactable in their own right, within their respective contexts. In contrast to that, there are workflow repositories which expose individual workflow components which, while they cannot be run on their own, they are used for composing enactable workflows at a finer granularity. An example of such a registry is the VERCE Dispel registry [38], which is designed to contain Dispel language [4] components, predominantly data-stream processing elements, as well as complete Dispel workflows currently targeted at seismology. Kepler has a large repertoire of available components, which it calls 'actors', covering both file-based tasks and stream-based processing [7]. Another system which follows a similar, fine-grained approach is KNIME, an open-source, enterprise-oriented workflow-based workbench addressing data analysis and transformation. It offers over 2,000 re-usable workflows and workflow fragments in its enterprise-extension repository, as well as allowing sharing via myExperiment and the use of web services, such as those in BioCatalogue.

The multitude of approaches, briefly introduced above, demonstrates that the design of generic tools should take into account the different requirements addressed regarding component granularity and computing resources as well as the "non-functional" attribution for use and modification,

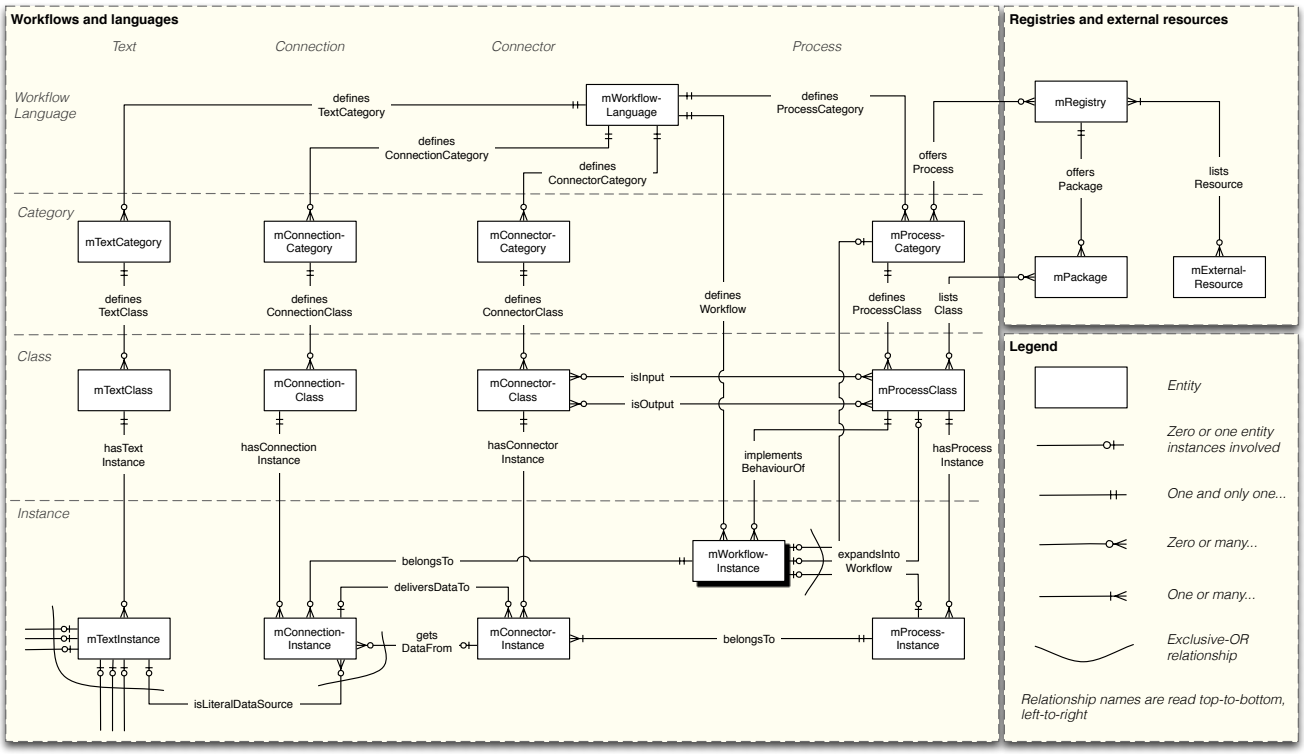


Figure 2: Entity relationship diagram for the model perspective of the generic workflow editor.

permissions and provenance. However, the ability to describe components and workflows in libraries has developed a repertoire of models for workflows – see for example [15].

3. A SINGLE DATA MODEL FOR MULTIPLE WORKFLOW CONCEPTS

Many diverse WfMSs are in use in numerous communities. The conceptual and technological diversity has led to numerous workflow languages varying in syntax and semantics. The main challenge for a generic web-based workflow editor has been to design a *single* model that is sufficiently comprehensive for integrating the diversity of workflow languages, concepts, components and representations without being overly complex. We identify three user roles and note that each role contributes information predominately to particular parts of the model that we introduce below.

Workflow language importer will be an expert in that language who configures basic entities and is responsible for providing the language properties (levels *Workflow Language* and *Category* of Figure 2), setting up the standard look and feel (i.e., mapping categories and classes to visual forms in the *controller*) and establishing the default list of registries thereby pre-populating the *Class* level and establishing an initial set of `mWorkflowInstance`s.

Workflow creator is an expert in a particular workflow language and some domain of use of that language. They are enabled to process all steps for the whole lifecycle of editing a workflow including altering graphical representations, thereby producing from scratch or from prior templates, new `mWorkflowInstance`s and all their associated *Instance* level information, and optionally new composite classes, that may also be submitted to a registry for reuse. They may also set some specific *controller* properties for their products.

Workflow editor is a typical user from a domain applying the workflows. They will view and parameterise `mWorkflowInstance`s, and then submit them. They may set viewing preferences in the *controller* and store versions of their submitted workflows and results *a la* VisTrails [8, 39].

The design of the workflow editor follows the MVC pattern. To make explicit the roles of entities in the model, entities belonging to the *model* (M) perspective are prefixed with `m`. Figure 2 represents the *model* (M) perspective of the *logical* information needed by the workflow editor. A number of relationships with `mTextInstance` have been suppressed for clarity. These allow the representation of names, annotation, comments, parameters, and scripts, to be specified and controlled. There are two main parts to the model perspective: firstly, *workflows and languages*, and secondly, *registries and external resources*. *Workflows and languages* include the entities for defining the logical flow of a workflow and distinct workflow instances, *registries and external resources* provide information about available workflow components and constructs, and accessible computation and data resources. Where applicable, workflow constructs may be bundled as packages that would be loaded as pallets of usable icons in the editor's *view* perspective. The logical distinction between the two parts derives from the provenance of the data stored in the entities. Users are enabled to compose workflows from configured workflow languages with information provided in the workflow editor. In con-

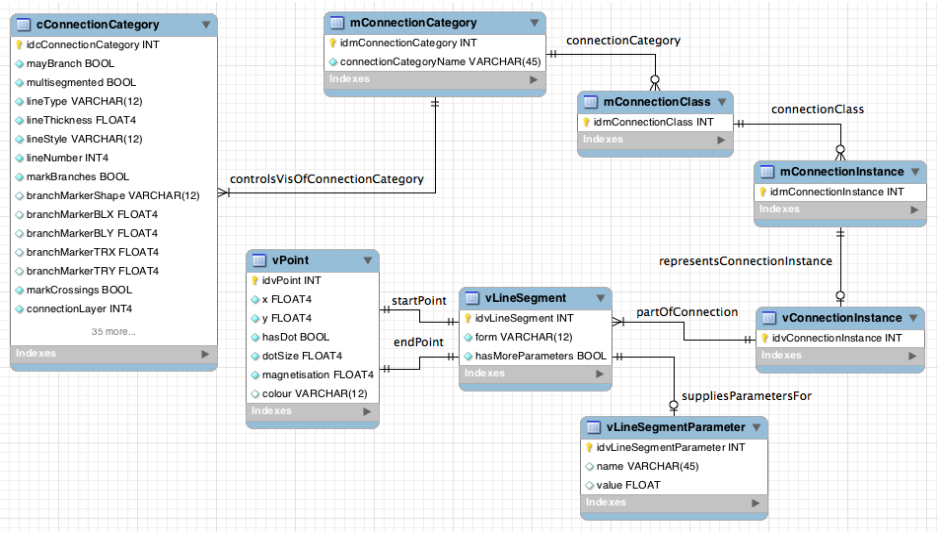


Figure 3: Entity relationship fragment illustrating model, view and controller perspectives.

trast, *registries and external resources* contain information, which is typically set up by an expert with knowledge about addresses and interfaces of systems outside of the workflow editor. Information in these registries is accumulated from many sessions by many users working either via the web-based generic editor or via today's tools. The goal is to offer a generic interface for diverse registries—eventually this should be underpinned by a standard registry API. Thus, users are enabled to conveniently connect to and use multiple registries.

The conceptual model for the entities of *workflows and languages* has four layers. In the first, a *workflow language* is introduced with, in the second layer, all *categories* of components that can be used in potential workflows, for each language. `mProcessCategory`, for example, defines the types of processes dependent on the workflow language, e.g. a `Job` in gUSE or a `PrimitivePE`, `CompositePE` or `FunctionalPE` (where `PE` denotes Processing Element) in Dispel. In the third layer, the *classes* of the components for each category are represented while the fourth layer contains the specific *instances* of all of the classes in any workflow instance that is being edited. Fortunately, as we are only editing the graph, not interpreting it, the model does not have to capture all of the semantics of the various workflow languages; it only needs to discriminate nodes and edges that need to be treated differently in each installed language. It needs to differentiate nodes or edges when they need to be rendered differently or when the available editing actions are different. For example, a processing node that is primitive does not have the possibility of being opened to show its expansion, whereas a composite node (or meta node) can be opened to show its expansion in any of its available forms – this enables recursive composition of workflows to be viewed and edited.

There is not space to fully describe elements in the *view* (V) and *controller* (C) perspectives – readers are referred to `bit.ly/WBWFE`. The *view* entities are required to represent only the model entities at the *class* and *instance* levels that have been or are being viewed. They are mostly homomorphic with the *model* perspective, with the corresponding name except with prefix `v`. A good example of where their form is more complex is in the representation of connections, as shown in Figure 3. Here, a set of not-necessarily-connected line segments (so that curved paths, manhattan paths, data-distribution trees, iteration and parallelisation may all be visualised) denote a `mConnectionInstance`. By default, the generation of the visual form is controlled by a *controller* entity higher in the conceptual tree, here a `cConnectionCategory`, that specifies the rules for all connections in that category. But the actual values in the *view* perspective may be set during upload from another representation or by a user manipulating the auto-generated form. Each `v` entity needs to reference its corresponding `m` entity and retain user-set or uploaded viewing information, such as size, icons and instance positions, connection routes, colouring, shading, fonts and line styles.

The elements of the *controller* (C) perspective have prefix `c` and for the most part, the corresponding names. They are only needed for the *category* level, as they describe the mappings for all entities consistent with the category to which they are linked. They may be created at a lower level to describe exceptional behaviour. They describe a two-way mapping between the model and view perspectives and what a user is permitted to change. For example, they specify the actions on hover, click and double-click, and the set of enabled operations shown on right click for their referenced entities. Their language-specific values are set by a workflow language importer, and their other values are set as user preferences. The handling of meta elements is a good example. They might highlight all of their input and output paths and immediately connected elements on click, might show a succinct description on hover, and might expand to expose their internal implementation on double-click, and offer a full repertoire of available operations on right click. Such potential behaviour will be described in a corresponding *controller* entity. More information about the entities can be found in Appendix A. The functions supported by the editor will include:

1. import from, and export to, the currently supported representations of a given workflow language;
2. creation, authoring and saving instances of workflows;

3. composition and amendment of potentially multi-lingual meta workflows;
4. management of each workflow's lifecycle, from abstract to concrete forms, enactment and discard;
5. interaction with repositories used for sharing; and
6. use of packages of predefined components and sub-workflows.

The prototype GeWWE (Generic Web-based Workflow Editor) in its current form is focused on the lifecycle of the editing process of workflows (see Fig. 4). Here the stages of workflows' lifecycles are more fine-grained than just *abstract* and *concrete*. An *abstract* workflow closely matches its visual representation. The successive stages of a *concrete* workflow are *logical flow*, *instantiated* workflow and *ready-to-process* workflow. In some workflow languages some of these stages may be elided by automated completion.

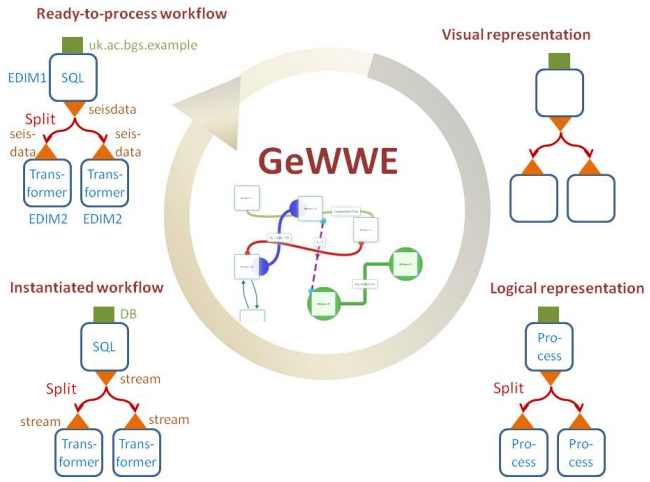


Figure 4: Lifecycle of editing a workflow.

The *visual representation* – GeWWE is a graphical editor, which allows users to select graphical nodes, the processes, and graphical connections with diverse forms and characteristics. The logical flow in the workflow is rudimentary and defined via the graph.

The *logical flow* – processes are associated with each other via connections between connectors. Connectors represent the input and output of a process and each connector is bound to one process instance whereas a process may be bound to several connectors.

The *instantiated workflow* – is one whose processes, connectors and connections are populated with full definitions.

The *ready-to-process workflow* – if an instantiated workflow includes complete descriptions of all of the exact processes, inputs, outputs and external resources to be used, then the workflow is in a *ready-to-process* state.

The result of a fully performed lifecycle is a ready-to-process workflow, which can be exported to a submission interface of a suitable workflow engine. The submission and the monitoring of a workflow is part of the submission interface. The latter is not yet part of GeWWE but envisaged for further development. The lifecycle of editing a workflow can also be started by importing into GeWWE a workflow at any stage of completion including ready-to-run, which can then be changed by a user or adapted for different inputs, outputs or external resources. GeWWE preserves the source of a workflow in the chosen workflow language as if the workflow has been edited via the original WfMS. Users are able to insert and edit for each node the source of the process, related connectors and connections.

4. STATUS AND EXAMPLES

A first prototype of the workflow editor has been developed as proof-of-concept. GeWWE is web-based and applies the VAADIN [27] framework deployed in a servlet container, e.g., Apache Tomcat [51]. VAADIN allows us to build server-side and client-side web applications in combination very efficiently. It supports Google Web Toolkit (GWT) [29] libraries and translates Java to JavaScript on the client side. To build workflow graphs with drag-and-drop mechanisms, we chose the powerful JavaScript library jsPlumb [34]. GeWWE has been developed as a combination of a server-side and client-side web software with the data used in the workflow editor being stored in a MySQL database [30]. The data is managed on the server-side using Hibernate [33]. Thus, the choice of the underlying database is flexible and another relational database could be used.

4.1 Example workflow language installations

In its current state, GeWWE rudimentarily supports three workflow languages: Dispel, gUSE [35] in XML format, and Galaxy [25] workflows in JSON format. We use simple examples and only discuss the editor issues; as the normal power and tutorial guides for the languages still apply. As the first step to demonstrate the model's generic applicability, these three workflow languages, which we already work with, were chosen for several reasons:

1. they are used in diverse communities,
2. the registry interface can be tested with Dispel, and
3. they exhibit significantly different enactment models, levels of abstraction and syntactic forms.
   - Dispel and Galaxy support users with a kind of toolbox. Dispel offers pre-configured processing elements in three distinct categories and Galaxy offers pre-configured tools.
   - gUSE and Galaxy support DAG-based workflows and interpret each process as a single job, whereas Dispel activates the whole graph and runs the tasks concurrently with data streaming between the processes.
   - gUSE and Galaxy support the import and export of workflow instances, whereas Dispel denotes workflows in textual form designed for both human and machine comprehension.

The following figures illustrate examples for each workflow system chosen to be sufficiently simple for exposition here.

4.2 Status

Figure 8 illustrates the basic layout of GeWWE. Users can switch between the main groups of functionality via the menubar. The options include opening, saving and deleting workflows. On the left side, workflow languages, registries, process classes and resources can be selected. Icons denoting different process classes can be chosen and inserted via drag-and-drop onto the right side and connected with each other by drawing arcs between their displayed connectors.

In the near future, the visualisation of connectors and the

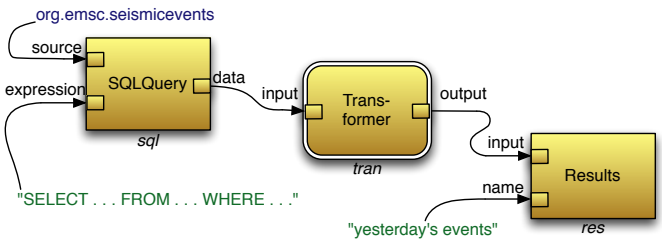


Figure 5: A data-streaming Dispel workflow—first executing an SQL expression, transforming the query results as they arrive and streaming results to the user—see Appendix B.

Figure 6: A gUSE workflow downloading a PDB file, splitting the stored complex into its receptor and ligand parts and performing a docking step.

options for structured text will be implemented. It will then be possible to edit processes, their parameters, and their inputs and outputs. Extended export and import features will be developed. The aim is not only to create and accept files in a pre-configured workflow language but also to check them with a suitable parser (that has been registered as a handler for the language) for syntactic correctness. These parsers will also be applied for checking manually inserted structured text to interpret it as a workflow or process. When a process is a composite or meta node, users will be able to double click on its icon and see an expansion representing its internal composition—this applies recursively and expanded nodes may be collapsed to their summary icon. GeWWE allows users to connect any components loaded from registries in a workflow—this includes, in principle, components from different languages. Eventually verification of the compatibility of connected components and other composition rules should be incorporated in the editor to give users early warning of potential errors and prevent the construction of compositions for which there is as yet no enactment system. Dispel, for example, is stream oriented. To connect a Dispel process with a gUSE or Galaxy process, the output of Dispel has to be inserted into a file to serve as input for a gUSE or Galaxy process. Vice versa, an output file has to be handed over as a data stream to a Dispel process.

Whereas we concentrate in the first step on editing workflows and importing and exporting them from and to suitable workflow management systems, we intend to add an interface for directly receiving workflows from WfMSs and submitting workflows to WfMSs. The assumption is that the WfMSs support an external interface, e.g. via web services.

In addition to the features of the workflow editor, an authentication and authorisation concept has to be integrated into the editor; at least sufficient to satisfy requirements for using resources and registries. On the one hand there are the different user roles (workflow language importer, workflow creator, workflow editor), on the other hand the integration

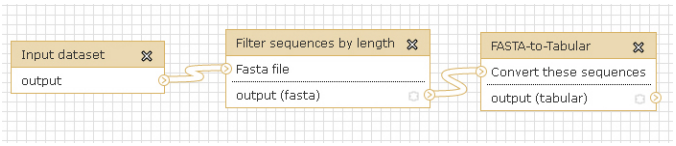


Figure 7: A Galaxy workflow for reading an input dataset in FASTA format, filtering the sequences by length and converting the output to a simple text file formatted for the import into Excel.

of external systems, such as registries, data and compute resources, relies on diverse security technologies. There has been no security standard evolved for cloud computing so far but most of the established grid middlewares and grid file systems rely on X.509-based certificates [32] or Shibboleth [43] for authentication. Batch systems again are mostly using role-based login-and-password mechanisms. Thus, the workflow editor has to offer an easily extendable interface for different security concepts.

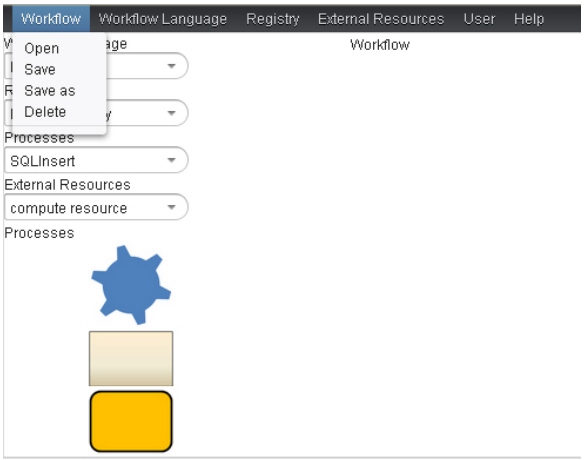
It will frequently be the case that the web-based editing will be accessed as a portlet in a science gateway. For example, it might be used as part of one of the 19 science gateways supported by SCI-BUS, in which case the editor would need to exploit and work well with the existing user identification and authentication system. Another example of such a requirement emerges with the Catania Science Gateway Framework (CSGF) [3, 6], which already complies with the EGI.eu [44] portal policies [36] using Shibboleth and SAML [46] on top of a JSAGA [13, 26] platform for organising execution on multiple kinds of DCIs. The editor would need to be capable of functioning in such a predefined context.

5. CONCLUSION AND OUTLOOK

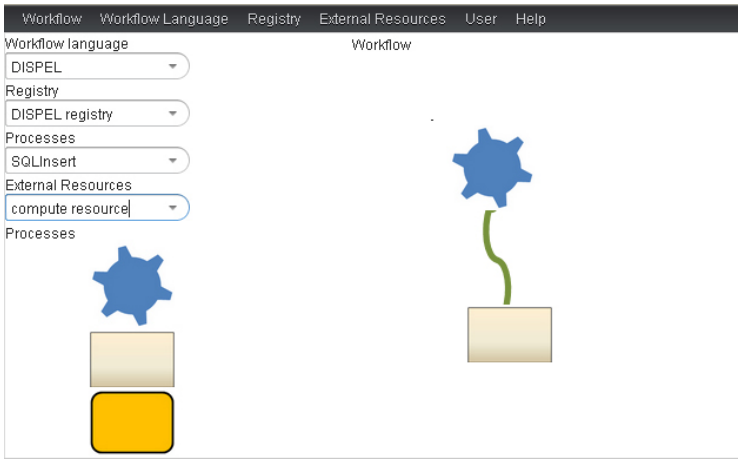
This paper was triggered by our own need for a web-based (i.e., run in any browser) graphical editor for the workflow languages we use. That in turn led to a survey of colleagues with other workflow languages, where we found a similar need existed. In many cases, they had work underway to build such an editor or they were planning such work. For example, KNIME currently requires a downloaded client for workflow editing; its web-based interactive workflow execution portal would benefit from a versatile workflow editor.

As we commenced our design we also noted that a full implementation would take substantial effort to build and maintain, and we spotted that commonality between solutions would have significant advantages: consistency for end users, a potential for convergence encouraging greater reuse of components and methods, and widely shared effort in development and maintenance making sophisticated web-based workflow editing more feasible and sustainable. The urgent need for action was evident—one-off solutions were already under development.

This posed the question, "Is a common framework for web-based workflow editing feasible and will it realise these benefits?". Drawing on previous experience of characterising workflows and their components in the ADMIRE [5] and SHIWA projects, and partitioning the design using the MVC pattern, we were able to propose a common logical model which we believe has the capacity to handle editing of all workflows, meeting their diversity and their communities'



(a) The menubar with drop-down menus for the main modes of editor operation.



(b) On the left a menu setting preferences for a Dispel workflow and for selected pallets of classes of components and on the right space showing a simple workflow.

Figure 8: The basic layout of GeWWE.

preferences. We partition the model into a mapping appropriate for a group of users that identifies resources and components they may use and a larger logical domain covering the workflows themselves. The latter is structured into four layers for power and comprehensibility. They progressively introduce a *workflow language*, the *categories* of component, connection, connector and text each language discriminates, introduces the specific *classes* of each of these that users may actually copy into a workflow they are editing, and then a final layer of *instances* which contain all the specific information about those copies. It is also structured into four logical columns: *processes*, which contain algorithms, *text* which names, denotes parameters or representations in other languages, *connections* that carry data from one process to another and also represent dependencies and timing constraints, and *connectors* that characterise the interfaces with processes. We argue that this is logically simple and sufficient for the required diversity. By arranging for the *view* and *controller* perspectives to be homomorphic with parts of the *model* perspective, minimal additional complexity is introduced.

We set out to test the hypothesis that this model was sufficient using GeWWE, a prototype implementation of the envisaged common framework. As yet we are not able to demonstrate the accommodation of workflow language *styles* and of interaction with enactment services. Otherwise, the model has stood up well to initial tests. We anticipate that the generic editor will be used in the context of many science gateways, for example the VERCE (seismology), EFFORT (rock physics and volcanology) [21], SCI-BUS and CSGF science gateways. As indicated above, there is a challenge to fit in these contexts, as they become more complex and as they adopt different security practices.

We have deliberately exposed these ideas to criticism and comment as early as possible. This means we have less evidence, but it also means we are not trying to sell a particular solution to the form and implementation of web-based workflow editors. Instead, we put forward the model as an open invitation for others to discuss:

1. whether the proposed common framework would be worthwhile, and
2. whether the model is appropriate for a large spectrum of workflows and their operational environments?

We hope there will be an emerging consensus on the former question, and that a community discussion will then drive refinement and further testing of the model and of the framework's implicit architecture. The following steps should be undertaken.

1. Complete and accurate mappings of several workflow languages into the model.
2. Development of a reasonably complete repertoire of the envisaged editing functions.
3. Analysis of submission and monitoring interfaces.
4. Consideration of modelling language-specific composition rules to give early error warnings.
5. Investigation of how best to support flexibly security without imposing it on all users.
6. Review of the model and framework in the light of the above five steps.

We believe that this will be best achieved by an open international collaboration developing an open-source software product, and would be delighted to hear from any reader interested in contributing – see bit.ly/WBWFE.

Acknowledgment

The initial research was funded by the NeSC Research Platform grant EP/F057695/1 from the UK EPSRC. Thanks also go to Dr Paul Martin, Data-Intensive Research Group, University of Edinburgh, Dr Dave Snelling, Fujitsu Labs Europe and Dr Liew Chee Sun, University of Malaya for their insightful and valuable comments.

6. REFERENCES

[1] A. Abdelnur and S. Hepper. JSR 168: Portlet Specification. http://www.jcp.org/en/jsr/detail?id=168, 2003.

[2] B. Ács, X. Llorà, L. Auvil, B. Capitanu, D. Tcheng, M. Haberman, L. Dong, T. Wentling, and M. Welge.

A general approach to data-intensive computing using the Meandre component-based framework. In *Proceedings of the 1st International Workshop on Workflow Approaches to New Data-centric Science*, WANDS '10, pages 8:1–8:12, New York, NY, USA, 2010. ACM.

[3] V. Ardizzone, R. Barbera, A. Calanducci, M. Fargetta, E. Ingrà, I. Porro, G. L. Rocca, S. Monforte, R. Ricceri, R. Rotondo, D. Scardaci, and A. Schenone. The decide science gateway. *J. Grid Comput.*, 10(4):689–707, 2012.

[4] M. Atkinson. Data-Intensive Thinking with DISPEL. In *THE DATA BONANZA: Improving Knowledge Discovery for Science, Engineering and Business*, chapter 4, pages 61–122. John Wiley & Sons Inc., 2013.

[5] M. P. Atkinson, C. S. Liew, M. Galea, P. Martin, A. Krause, A. Mouat, O. Corcho, and D. Snelling. Data-intensive architecture for scientific knowledge discovery. *Distributed and Parallel Databases*, 30:307–324, 2012.

[6] R. Barbera and *et al.* Catania Science Gateway Framework. http://www.catania-science-gateways.it, 2013.

[7] D. Barseghian, I. Altintas, M. B. Jones, D. Crawl, N. Potter, J. Gallagher, P. Cornillon, M. Schildhauer, E. T. Borer, E. W. Seabloom, and P. R. Hosseini. Workflows and extensions to the Kepler scientific workflow system to support environmental sensor data access and analysis. *Ecological Informatics*, 5:42–50, 2010.

[8] L. Bavoil, S. P. Callahan, P. J. Crossno, J. Freire, and H. T. Vo. VisTrails: Enabling interactive multiple-view visualizations. In *In IEEE Visualization 2005*, pages 135–142, 2005.

[9] K. Belhajjame, O. Corcho, D. Garijo, J. Zhao, P. Missier, D. Newman, R. Palma, S. Bechhofer, E. García, G.-P. J. Manuel, G. Klyne, K. Page, M. Roos, J. E. Ruiz, S. Soiland-Reyes, L. Verdes-Montenegro, D. D. Roure, and C. Goble. Workflow-centric research objects: First class citizens in scholarly discourse. In *Proceedings of the Second International Conference on the Future of Scholarly Communication and Scientific Publishing*, 2012.

[10] R. Berjon, S. Faulkner, T. Leithead, E. Navarra, E. O'Connor, S. Pfieffer, and Hickson, I. (Eds). HTML 5.1: A vocabulary and associated APIs for HTML and XHTML. Technical report, W3C, 2013.

[11] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, K. Thiel, and B. Wiswedel. KNIME - the konstanz information miner. *SIGKDD Explorations*, 11(1), 2009.

[12] J. Bhagat, F. Tanoh, E. Nzuobontane, T. Laurent, J. Orlowski, M. Roos, K. Wolstencroft, S. Aleksejevs, R. Stevens, S. Pettifer, R. Lopez, and C. Goble. BioCatalogue: a universal catalogue of web services for the life sciences. *Nucleic Acids Research*, 2010.

[13] CCIN2P3. JSAGA. http://grid.in2p3.fr/jsaga, 2013.

[14] Condor Team. Condor DAGMan manual. Technical report, University of Wisconsin-Madison, 2008.

[15] O. Corcho. Sharing and Reuse in Knowledge Discovery. In *THE DATA BONANZA: Improving Knowledge Discovery for Science, Engineering and Business*, chapter 8, pages 181–192. John Wiley & Sons Inc., 2013.

[16] V. Curcin and M. Ghanem. Scientific workflow systems - can one size fit all? In *Cairo International Biomedical Engineering Conference*, CIBEC '08, pages 1–9, December 2008.

[17] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, May 2009.

[18] E. Elmroth, F. Hernández, and J. Tordsson. Three fundamental dimensions of scientific workflow interoperability: Model of computation, language, and execution environment. *Future Generation Computer Systems*, 26(2):245–256, February 2010.

[19] T. Fahringer, R. Prodan, R. Duan, J. Hofer, F. Nadeem, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, and M. Wieczorek. ASKALON: A Development and Grid Computing Environment for Scientific Workflows. In I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, editors, *Workflows for e-Science*, pages 450–471. Springer London, 2007.

[20] Z. Farkas and P. Kacsuk. P-GRADE portal: A generic workflow system to support user communities. *Future Generation Computer Systems*, 27(5):454–465, 2011.

[21] R. Filgueira et al. EFFORT (Exploring Failure Forecasting in Real Time). http://effort.is.ed.ac.uk:8080/, 2013.

[22] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers. Examining the challenges of scientific workflows. *Computer*, 40(12):24–32, December 2007.

[23] Y. Gil, V. Ratnakar, E. Deelman, G. Mehta, and J. Kim. Wings for Pegasus: Creating large-scale scientific applications using semantic representations of computational workflows. In *Proceedings of the Nineteenth Conference on Innovative Applications of Artificial Intelligence*, IAAI '07, pages 1767–1774. AAAI Press, July 2007.

[24] C. Goble et al. myGrid. http://www.mygrid.org.uk, 2013.

[25] J. Goecks, A. Nekrutenko, J. Taylor, and G. Team. Galaxy: A comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8):R86, 2010.

[26] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, A. Merzky, J. Shalf, and C. Smith. A Simple API for Grid Applications (SAGA). Technical Report GFD.90, Open Grid Forum, 2011.

[27] M. Grönroos. *Book of Vaadin*. Oy IT Mill Ltd, 2010.

[28] Z. Guan, F. Hernandez, P. Bangalore, J. Gray, A. Skjellum, V. Velusamy, and Y. Liu. Grid-Flow: a Grid-enabled scientific workflow system with a Petri-net-based interface. *Concurrency and Computation: Practice and Experience*, 18:1115–1140, 2006.

[29] M. Hahn. The Google Web Toolkit: a deeper look and Extensions for GWT.

http://www.dark-bit.de/wp-content/uploads/2009/07/paper_marcel_hahn_final.pdf, 2008.
[30] S. Hinz et al. MySQL. http://dev.mysql.com, 2013.
[31] D. Hull, K. Wolstencroft, R. Stevens, C. A. Goble, M. R. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34:729–732, 2006.
[32] I. T. U. (ITU). ITU-T Recommendation X.509. http://www.itu.int/rec/T-REC-X.509/en, 1988.
[33] JBoss Community. Hibernate. http://www.hibernate.org/, 2013.
[34] JBoss Community. jsPlumb. http://jsplumbtoolkit.com/doc/home, 2013.
[35] P. Kacsuk, Z. Farkas, M. Kozlovszky, G. Hermann, Á. Balaskó, K. Karóczkai, and I. Marton. WS-PGRADE/gUSE Generic DCI Gateway Framework for a Large Variety of User Communities. *J. Grid Comput.*, 10(4):601–630, 2012.
[36] D. Kelsey. EGI-InSPIRE VO Portal Policy. https://documents.egi.eu/public/ShowDocument?docid=80, 2010.
[37] J. Kim, E. Deelman, Y. Gil, G. Mehta, and V. Ratnakar. Provenance trails in the Wings/Pegasus system. *Concurrency and Computation: Practice and Experience*, 20(5):587–597, April 2008.
[38] I. Klampanos. Supporting Collaborative Scientific Workflow Development: The Dispel Information Registry. http://research.nesc.ac.uk/files/Registry-OSDC13.pdf, 2013.
[39] D. Koop, C. E. Scheidegger, S. P. Callahan, J. Freire, and C. T. Silva. VisComplete: automating suggestions for visualization pipelines. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1691–1698, 2008.
[40] D. Krefting, T. Glatard, V. Korkhov, J. Montagnat, and S. Olabarriaga. Enabling Grid Interoperability at Workflow Level. In *Proceedings of Grid Workflow Workshop 2011*, volume 826. CEUR Workshop Proceedings, 2012.
[41] S. Marru, L. Gunathilake, C. Herath, P. Tangchaisin, M. E. Pierce, C. Mattmann, R. Singh, T. Gunarathne, E. Chinthaka, R. Gardler, A. Slominski, A. Douma, S. Perera, and S. Weerawarana. Apache airavata: a framework for distributed applications and computational workflows. In *SC-GCE*, pages 21–28, 2011.
[42] P. Martin and G. Yaikhom. Definition of the DISPEL Language. In *THE DATA BONANZA: Improving Knowledge Discovery for Science, Engineering and Business*, Parallel and Distributed Computing, series editor Albert Y. Zomaya, chapter 10, pages 203–236. John Wiley & Sons Inc., 2013.
[43] R. L. Morgan, S. Cantor, S. Carmody, W. Hoehn, and K. Klingenstein. Federated Security: The Shibboleth Approach. *EDUCAUSE Quarterly*, 27(4):12–17, 2004.
[44] S. Newhouse et al. European Grid Infrastructure. http://www.egi.eu, 2013.
[45] M. Nicklous and S. Hepper. JSR 286: Portlet Specification 2.0. http://www.jcp.org/en/jsr/detail?id=286, 2008.
[46] OASIS. Security Assertion Mark-up Language. http://saml.xml.org, 2013.
[47] D. D. Roure, C. Goble, and R. Stevens. The design and realisation of the ^{my}Experiment Virtual Research Environment for social sharing of workflows. *Future Generation Computer Systems*, 25(5):561–567, 2009.
[48] M. Rynge. Pegasus 4.2 on the Open Science Grid. http://pegasus.isi.edu/presentations/2013/Pegasus-4.2-OSG-2013.pdf.
[49] I. Taylor, M. Shields, I. Wang, and A. Harrison. Visual Grid Workflow in Triana. *Journal of Grid Computing*, 3:153–169, 2005. 10.1007/s10723-005-9007-3.
[50] G. Terstyanszky, T. Kukla, T. Kiss, P. Kacsuk, Ákos Balaskó, and Z. Farkas. Enabling Scientific Workflow Sharing through Coarse-Grained Interoperability. *Journal of Future Generation Computing Systems*, submitted 2013 (under review).
[51] The Apache Software Foundation. Apache Tomcat. http://tomcat.apache.org/tomcat-6.0-doc/.
[52] The Apache Software Foundation. Apache OODT. http://oodt.apache.org, 2013.
[53] W. van der Aalst and A. ter Hofstede. Workflow Patterns. http://www.workflowpatterns.com, 2013.

APPENDIX

A. MODELS TO REPRESENT WORKFLOWS

The table gives brief definitions of the logical entities in the *model* perspective (as shown in Figure 2) to which all workflows are mapped, hence each has prefix `m`. The related *view* and *controller* logical models are straightforwardly related as described in Section 3 and bit.ly/WBWFE.

Table: Roles of entities in the model perspective (Figure 2)

Entity	Describes
Registries and external resources	
`mRegistry`	External descriptions of computational resources, data sources, libraries, workflow components, tools, and web services.
`mExternalResource`	Available compute and data resources.
`mPackage`	Collections of components.
Workflows and languages	
Workflow Languages	
`mWorkflowLanguage`	Each workflow language installed.
Category	
`mTextCategory`	Major roles for text, e.g., plain, script, structured, XML.
`mConnectionCategory`	How data are passed, e.g., as parameters, files, streams, and control flow, e.g., split, join pair, or condition.
`mConnectorCategory`	Types of input and output to a process.
`mProcessCategory`	Categories of process, e.g., application, inline function, web service or stream processor.
Class	
`mTextClass`	A role for text, e.g., class name, instance identifier, input parameters, description, annotation.
`mConnectionClass`	A specific pattern of data transport and flow control, e.g., deliver output file to destinations and start them.
`mConnectorClass`	A specific form of connection termination on a process boundary, e.g., parameter input or data-stream output.
`mProcessClass`	Behaviour and algorithm that this process applies, e.g., `DBQuery`, `Merge`.

Entity	Describes
Instance	
mTextInstance	Acts in exactly one of ≈ 20 roles, including: {naming, identifying, describing or annotating} a {process, connection, connector or workflow} or providing a parameter or script.
mConnectionInstance	An instance of a connection class, with a given connector or plain text as source and ≥ 1 destination connectors.
mConnectorInstance	A particular connector on a process instance at an end of a connection.
mWorkflowInstance	The whole workflow on which the editor is acting or a sub-workflow corresponding to an expansion of a composite or meta-node process.
mProcessInstance	An instance of a process class.

B. EXPANSION OF DISPEL WORKFLOW

We now illustrate the use of the model entities for the Dispel example in Figure 5. The textual representation in the Dispel language follows.

```
1  package paper.seismology {          //set context
2    use dispel.db.SQLQuery;           //import SQLQuery
3    use paper.Transform;              //import Transform
4    use dispel.lang.Results;          //import Results
5
6    SQLQuery sql = new SQLQuery;    // new instance
7    Transform tran = new Transform; // new instance
8    Results res = new Results;      // new instance
9
10   sql.data => tran.input;      // data flow sql to tran
11   tran.output => res.input;    // data flow tran to res
12   |- "org.emsc.seismicevents" -|
13       => sql.source;           //URI of data resource
14   |- "SELECT ... FROM ... WHERE ..."
15       -| => sql.expression;    // supply the query
16   |- "yesterday's events" -|
17       => res.name;             //name results
18
19   submit res;                  // submit for enactment
20 }
```

We then tabulate the top-level entities that would be used to denote that workflow with abbreviated content and omitting all relationships. The instances of entities are ordered in terms of the logical layers of Figure 2 except that forward references have been avoided when possible. The *Id* is automatically created by the system and reflects the order of the creation of the entities.

Instances of model entities representing Dispel example

Id	Model Entity	content / role
m001	mWorkflowLanguage	"Dispel"
m002	mProcessCategory	primitivePE / implementation hidden – may be in any language
m003	mProcessCategory	compositePE / implementation, inner workflow may be opened
m004	mTextCategory	Dispel workflow / whole or part workflows
m005	mTextCategory	Identifier / identifies anything
m006	mTextCategory	String / denotes parameters
m007	mConnectionCategory	Stream / one connection type
m008	mConnectorCategory	Input / any input data stream
m009	mConnectorCategory	Output / output data stream
m010	mProcessClass	SQLQuery /defines an m002, description and mechanism
m011	mProcessClass	Transformer / defines an m003 description and expansion
m012	mProcessClass	Result /defines an m002, description and mechanism
m013	mTextClass	Dispel script instances m004 / denotes a workflow
m014	mTextClass	Identifier instances m005 / names
m015	mTextClass	Literal instances m006 / used for stream literals
m016	mConnectionClass	Stream instances m007 / streams
m017	mConnectorClass	Input instances m008 / for input
m018	mConnectorClass	Output instances m009 / output
m019	mProcessInstance	sql instance of m010 / does query, streams result
m020	mProcessInstance	tran instance of m011 / transforms data as it arrives
m021	mProcessInstance	res instance of m012 / sends stream to user
m022	mWorkflowInstance	Workflow / whole of example
m023	mWorkflowInstance	Workflow / expansion of m011
m024	mTextInstance	Instance of m013 / expansion of m022
m025	mTextInstance	Instance of m013 / expansion of m023
m026	mConnectorInstance	Instance of m017 / input to sql
m027	mConnectorInstance	Instance of m017 / input to sql
m028	mConnectorInstance	Instance of m018 / output from sql
m029	mConnectorInstance	Instance of m017 / input to tran
m030	mConnectorInstance	Instance of m018 / output from tran
m031	mConnectorInstance	Instance of m017 / input to res
m032	mConnectorInstance	Instance of m017 / input to res
m033	mTextInstance	"source", Instance of m014 /identifies m026
m034	mTextInstance	"expression", Instance of m014 /identifies m027
m035	mTextInstance	"data", Instance of m014 /identifies m028
m036	mTextInstance	"input", Instance of m014 /identifies m029 and m031
m037	mTextInstance	"output", Instance of m014 /identifies m030
m038	mTextInstance	"name", Instance of m014 /identifies m032
m039	mTextInstance	"org.emsc.seismicevents", Instance of m015 /stream literal
m040	mTextInstance	"SELECT . . . FROM . . . WHERE . . .", Instance of m015 /stream literal
m041	mTextInstance	"yesterday's events", Instance of m015 /stream literal
m042	mConnectionInstance	Instance of m016 / from m039 to m026
m043	mConnectionInstance	Instance of m016 / from m040 to m027
m044	mConnectionInstance	Instance of m016 / from m028 to m029
m045	mConnectionInstance	Instance of m016 / from m030 to m031
m046	mConnectionInstance	Instance of m016 / from m041 to m032
m047	mTextInstance	"SQLQuery", Instance of m014 /PE identifier m019
m048	mTextInstance	"sql", Instance of m014 /instance identifier m019
m049	mTextInstance	"Transformer", Instance of m014 /PE identifier m020
m050	mTextInstance	"tran", Instance of m014 /instance identifier m020
m051	mTextInstance	"Results", Instance of m014 /PE identifier m021
m052	mTextInstance	"res", Instance of m014 /instance identifier m021