

A Reference Architecture for Scientific Workflow Management Systems and the VIEW SOA Solution

Cui Lin, *Student Member, IEEE*, Shiyong Lu, *Member, IEEE*, Xubo Fei, Artem Chebotko, *Member, IEEE*, Darshan Pai, *Student Member, IEEE*, Zhaoqiang Lai, Farshad Fotouhi, *Member, IEEE*, and Jing Hua, *Member, IEEE*

Abstract—Scientific workflows have recently emerged as a new paradigm for scientists to formalize and structure complex and distributed scientific processes to enable and accelerate many scientific discoveries. In contrast to business workflows, which are typically control flow oriented, scientific workflows tend to be dataflow oriented, introducing a new set of requirements for system development. These requirements demand a new architectural design for scientific workflow management systems (SWFMSs). Although several SWFMSs have been developed that provide much experience for future research and development, a study from an architectural perspective is still missing. The main contributions of this paper are: 1) based on a comprehensive survey of the literature and identification of key requirements for SWFMSs, we propose the first reference architecture for SWFMSs; 2) according to the reference architecture, we further propose a service-oriented architecture for VIEW (a Visual sciEntific Workflow management system); 3) we implemented VIEW to validate the feasibility of the proposed architectures; and 4) we present a VIEW-based scientific workflow application system (SWFAS), called FiberFlow, to showcase the application of our VIEW system.

Index Terms—Reference architecture, scientific workflows, scientific workflow management system, SOA, VIEW.

1 INTRODUCTION

SCIENTIFIC workflows have recently emerged as a new paradigm for scientists to integrate, structure, and orchestrate a wide range of local and remote heterogeneous services and software tools into complex scientific processes to enable and accelerate many scientific discoveries [1]. A scientific workflow is the computerized facilitation or automation of a scientific process, in whole or part, which usually streamlines a collection of scientific tasks with data channels and dataflow constructs to automate data computation and analysis to enable and accelerate scientific discovery. A scientific workflow management system (SWFMS) is a system that completely defines, modifies, manages, monitors, and executes scientific workflows through the execution of scientific tasks whose execution order is driven by a computerized representation of the workflow logic. The design of a reference architecture at an appropriate level of abstraction that addresses architectural requirements for SWFMSs is critical and challenging.

The Workflow Management Coalition (WfMC) proposed a reference architecture for business workflows [2] in 1995. Since then, the reference architecture and its variants [3] have been widely adopted in the development of different

business workflow management systems (BWFMSs) [4], [5], [6], [7], [8]. However, these reference architectures are not suitable for SWFMSs as business workflows and scientific workflows have different goals. While the goal of business workflows is to reduce human resources (and other costs) and increase revenue, the goal of scientific workflows is to reduce both human and computation costs and accelerate the speed of turning large amounts of bits and bytes into knowledge and discovery. Moreover, business workflows are typically control flow oriented, while scientific workflows tend to be dataflow oriented, introducing a new set of requirements and challenges for system development, from the support of intensive user interaction and visualization, customizable and extensible GUI, reproducibility, high-end computing, to heterogeneous data, software tool, and service management. While several SWFMSs [9], [10], [11], [12], [13], [14] have been developed during the past few years, which provide much experience for future research and development, an architectural reference that can provide a high-level organization of subsystems and their interactions in an SWFMS is missing. The state of the art is still ad hoc in scientific workflow design, specification, development, execution, and provenance tracking, etc. First, each system uses a proprietary workflow language, whose semantics has not yet been fully investigated and formalized. Second, each system has either no explicit architectural design or the architecture is proprietary and restricted greatly by the legacy system that the SWFMS is built upon. For example, Kepler is built on the Ptolemy II system, and therefore, each new requirement that is needed by an SWFMS is based on extensions to the architecture of Ptolemy. Pegasus, on the other hand, is built upon Condor and Dagman by adding another workflow mapper on the

• C. Lin, S. Lu, X. Fei, D. Pai, Z. Lai, F. Fotouhi, and J. Hua are with the Department of Computer Science, Wayne State University, Detroit, MI 48202. E-mail: {cullin, shiyong, xubo, darshan, kevinlai, fotouhi, jinghua}@wayne.edu.

• A. Chebotko is with the Department of Computer Science, University of Texas—Pan American, Edinburg, TX 78539. E-mail: artem@wayne.edu.

Manuscript received 9 Nov. 2008; revised 15 Jan. 2009; accepted 28 Jan. 2009; published online 9 Feb. 2009

For information on obtaining reprints of this article, please send e-mail to: tsc@computer.org, and reference IEEECS Log Number TSC-2008-11-0098. Digital Object Identifier no. 10.1109/TSC.2009.4.

top of these two systems. Third, all these systems have different provenance models, not only in terms of what provenance information should be recorded, but also in terms of representation, storage, and querying models. We expect that the availability of such a reference architecture can provide a basis for comparison between different systems and a guidance for the architectural design of an SWFMS in a specific scientific domain.

To address this issue,

1. we propose the first reference architecture for SWFMSs based on a comprehensive survey of the literature and identification of key requirements;
2. according to the proposed reference architecture, we further propose a service-oriented architecture for the VIEW system. Leveraging SOA [15], VIEW consists of six loosely coupled service components, each of which corresponds to a functional component that is identified in the reference architecture, whose functionality is exposed as a Web service;
3. we implemented the VIEW system to validate the feasibility of the proposed architectures; and
4. we present a VIEW-based scientific workflow application system (SWFAS), called FiberFlow, to demonstrate the capabilities of VIEW in support of user-interaction-intensive, visualization-intensive, and compute-intensive scientific workflows in a heterogeneous and distributed computing environment.

The rest of the paper is organized as follows: Section 2 identifies seven key architectural requirements for SWFMSs. In response to these requirements, Section 3 proposes our reference architecture for SWFMSs. According to the reference architecture, Section 4 further proposes a service-oriented architecture for VIEW (a Visual sciEntific Workflow management system). Section 5 presents configuration management for VIEW subsystems, and Section 6 showcases the FiberFlow system using VIEW as its underlying SWFMS. After that, we evaluate five representative SWFMSs using the proposed reference architecture in Section 7, followed by related work in Section 8. Finally, Section 9 concludes the paper and comments on future work.

2 SEVEN KEY ARCHITECTURAL REQUIREMENTS

In addition to the general requirements of scalability, reliability, extensibility, availability, and security, what are the key architectural requirements for an SWFMS? Based on a comprehensive study of the workflow literature from an architectural perspective [16] and our own experience from the development of the VIEW system, we identify the following seven key architectural requirements for an SWFMS:

R1: User interface customizability and user interaction support. In scientific workflows, scientists are often the end users to design, modify, run, rerun, and monitor scientific workflows. User friendly graphical user interfaces are critical to increase the usability of an SWFMS. Domain-specific visualization capability is often needed to support the visualization of various workflow artifacts. The goal is to speed up the *exploratory process* of arriving at a proper workflow design with appropriate parameter values and input data sets that lead to sought-after scientific results. Therefore, a key architectural requirement

is the flexibility of customizing the user interface according to different science and engineering disciplines, scientific domains, or problems, or to an individual scientist's style, while reusing the same underlying workflow management framework. Customizing user interface should be localized and should not affect any other functional components of the system.

R2: Reproducibility support. Reproducibility is the fundamental principle of any science method. Scientific results produced from the execution of scientific workflows must be reproducible. Therefore, sufficient provenance information, including the derivation history of a data product, needs to be maintained in order to answer the following questions: What workflows or workflow steps are executed to produce this result? Which versions of softwares and OSs are used? What parameter values are used? What input data sets have contributed to this result? What scientists' interactions are involved in producing this result? With such information, a scientific result can be reproduced in the same system or in other peer systems when necessary. Therefore, a key functional component for an SWFMS is the management of provenance metadata, from collection, representation, storage, and querying, to visualization. Such a component is usually not required for a BWFMS.

R3: Heterogeneous and distributed services and software tools integration. Scientists often need to integrate and orchestrate a wide range of heterogeneous analytical and computational services and software tools into a scientific workflow for solving a complex scientific problem. Such services and software tools are usually written in various programming languages, invoked by different invocation mechanisms, and run in heterogeneous and distributed computing environments. Therefore, a key architectural requirement is to provide an abstraction of various services and software tools as *workflow tasks* (abr. task in this paper). Tasks not only keep scientists transparent to the heterogeneity and distribution of underlying task components, but also promote SWFMS extensibility so that the integration of future services and software tools whose interfaces and communication protocols are yet unknown does not affect other functional components of an SWFMS.

R4: Heterogeneous and distributed data product management. The execution of scientific workflows often consume and produce huge amounts of distributed data objects. These data objects can be of primitive or complex types, files in different sizes and formats, database tables, or data objects in other forms. Scientists are often overwhelmed and lost in the sea of heterogeneous and distributed data objects. Therefore, a key architectural requirement for an SWFMS is to provide an abstraction of these data objects as *data products*. Data products for SWFMSs include: 1) *workflow source data* that are registered into an SWFMS from external sources (produced by other systems, instruments, or experiments); 2) *workflow parameters* that are specified and tuned by users for each workflow run; 3) *workflow results* which consist of workflow intermediate and final results produced by workflow runs. Therefore, an SWFMS needs to support the efficient management of data products, including data product storage, archival, browsing, querying, access, movement, and visualization.

R5: High-end computing support. Today, many scientific problems need the support of high-end computing, such as

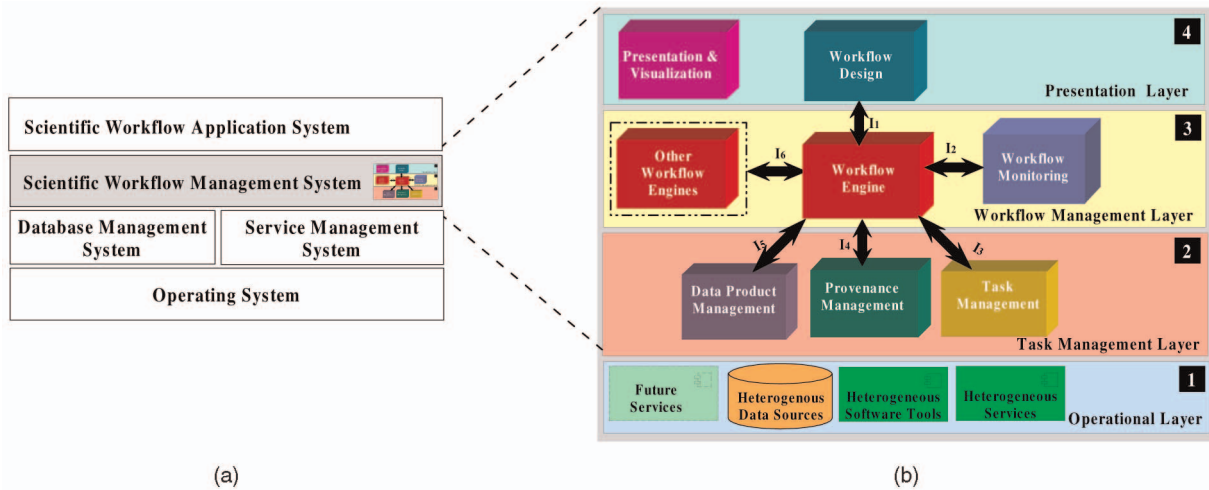


Fig. 1. (a) The position of an SWFMS within a software stack and (b) zoom-in view of the reference architecture for SWFMSs.

Grid computing and Cloud computing [17]. Given the fast advance of high-end computing technology, a key architectural requirement for an SWFMS is to separate the science-focused and technology-independent problem solving environment (PSE) from the underlying often fast advanced high-end computing infrastructure. In this way, domain scientists can focus on their science while utilizing the state-of-the-art computing technologies in a transparent fashion.

R6: Workflow monitoring and failure handling. The monitoring of the progress of the workflow execution is very important, particularly for long-running scientific workflows. Moreover, since scientific workflows are often designed and modified by scientists in an ad hoc fashion and can involve various distributed tasks that are accessed over network communications, many exceptions or failures can occur in an unforeseeable way. Finally, the complexity and scale of data analysis and computation in scientific workflows impose additional challenges on workflow monitoring and failure handling. Therefore, a key architectural requirement for an SWFMS is to provide the support for status and failure monitoring at various levels and the mechanism for catching, localizing, and handling failures automatically or with minimal human intervention.

R7: Interoperability. As more and more scientific research projects become collaborative in nature and involve multiple geographically distributed organizations, many scientific workflows are distributed and collaborative, consisting of multiple subworkflows, each of which might be managed by a different SWFMS. Therefore, a key architectural requirement for SWFMSs is to promote and facilitate the interoperability between different SWFMSs so that one SWFMS can take advantage of the software tool libraries and salient features provided by another SWFMS. The interoperability for SWFMSs lies in three levels: 1) *task-level interoperability*, which requires that various tasks and data products from different SWFMSs can interoperate one with another; 2) *workflow-level interoperability*, which requires that a scientific workflow in one SWFMS can be executed in or invoked by another SWFMS; and 3) *subsystem-level interoperability*, which requires that a subsystem in one SWFMS can be reused by or shared by different SWFMSs.

3 A REFERENCE ARCHITECTURE FOR SWFMSs

Although the reference architecture proposed by WfMC [2] has been widely used for BWFMSs, this reference architecture does not satisfy key requirements R1-R5 for SWFMSs identified in the previous section. In this section, we propose a reference architecture for SWFMSs. As shown in Fig. 1b, the reference architecture consists of four logical layers, seven major functional subsystems, and six interfaces. Fig. 1a shows a typical software stack of a scientific workflow application: on top of an operating system, a data management system and a service management are used by an SWFMS for data management and service management, respectively. An SWFAS is developed over an SWFMS by the introduction of additional domain-specific application data and functionalities.

3.1 Layers

The first layer is the *Operational Layer*, which consists of a wide range of heterogeneous and distributed data sources, software tools, services, and their operational environments, including high-end computing environments. The separation of the Operational Layer from other layers isolates data sources, software tools, services, and their associated high-end computing environments from the scope of an SWFMS, thus satisfying requirement R5.

The second layer is called the *Task Management Layer*. Tasks are the building blocks of scientific workflows. Tasks consume input data products and produce output data products. At the same time, provenance is captured automatically to record the derivation history of a data product, including original data sources, intermediate data products, and the steps that are applied to produce the data product. This layer abstracts underlying heterogeneous data into data products, services, and software tools into tasks, and provides efficient management for data products, tasks, and provenance metadata. Therefore, the Task Management Layer satisfies requirements R2, R3, and R4. Moreover, the separation of the Task Management Layer from the Operational Layer promotes the extensibility of the Operational Layer with new services and new high-end computing facilities, and localizes system evolution due to hardware or software advances to the interface between the Operational Layer and the Task Management Layer. The

task-level interoperability requirement (R7: level 1) should be addressed in this layer.

The third layer is the *Workflow Management Layer*, which is responsible for the execution and monitoring of scientific workflows. At this layer, the building blocks of a scientific workflow are the tasks provided by the underlying Task Management Layer. In this layer, an execution of a scientific workflow is called a *workflow run*, which consists of an coordinated execution of tasks, each of which is called a *task run*. Therefore, the Workflow Management Layer addresses requirements R6 and R7. The separation of the Workflow Management Layer from the Task Management Layer concerns two aspects as follows: 1) it isolates the choice of a workflow model from the choice of a task model, so changes to the workflow structure do not need to affect the structures of tasks and 2) it separates workflow scheduling from task execution, thus improves the performance and scalability of the whole system. The interoperability of workflows (requirement R7: level 2) has to be addressed by standardizing *workflow models*, *workflow run models*, and *workflow languages*.

The fourth layer is the *Presentation Layer*, which provides the functionality of workflow design and various user interfaces and visualizations for all assets of the whole system. The Presentation Layer has interfaces to each lower layer (not shown in the figure for simplicity). The separation of the Presentation Layer from other layers provides the flexibility of customizing the user interfaces of the system and promotes the reusability of the rest of system components for different scientific domains. Thus, this separation supports requirement R1. The interoperability of workflows (requirement R7: level 2) should be addressed by standardizing the workflow layout (e.g., look-and-feel) at this layer.

3.2 Subsystems

The seven major functional subsystems correspond to the key functionalities required for an SWFMS. Although the reference architecture allows the introduction of additional subsystems and their features in each layer, this paper only focuses on the major subsystems and their essential functionalities.

The *Workflow Design* subsystem is responsible for the design and modification of scientific workflows. Workflow Design produces workflow specifications represented in a workflow specification language that supports a particular workflow model. One can design and modify a scientific workflow using a standalone or Web-based workflow designer, which supports both graphical- and scripting-based design interfaces. The interoperability of workflows (requirement R7: level 2) should be addressed in this subsystem by the standardization of workflow languages.

The *Presentation and Visualization* subsystem is very important especially for data-intensive and visualization-intensive scientific workflows, in which the presentation of workflows and visualization of various data products and provenance metadata in multidimensions are the key to gain insights and knowledge from large amount of data and metadata. These two subsystems are located at the Presentation Layer to meet requirement R1. In this subsystem, the interoperability of workflows (requirement R7: level 2) should be addressed by the standardization of scientific workflow layout.

The *Workflow Engine* subsystem is at the heart of the whole system and is the subsystem that provides management and execution environments for workflow runs. The Workflow Engine creates and executes workflow runs according to a workflow run model, which defines the state transitions of each scientific workflow and its constituent task runs. The interoperability of workflows (requirement R7: level 2) should be addressed by the standardization of interfaces, workflow models, and workflow run models, so that a scientific workflow or its constituent subworkflows can be scheduled and executed in multiple Workflow Engines that are provided by various vendors. In SWFMSs, multiple Workflow Engine subsystems can be distributed, and each Workflow Engine can execute several workflows in parallel.

The *Workflow Monitoring* subsystem meets requirement R6 and is in charge of monitoring the status of workflow execution during workflow runtime and if failures occur, provides tools for failure handling [18].

The *Task Management* subsystem addresses heterogeneity and distribution issues (requirement R3) and provides management and execution environment for tasks, according to a *task model* and *task run model*, respectively. The interoperability of tasks between various workflow environments (requirement R7: level 1) can be addressed in this subsystem.

The *Provenance Management* subsystem meets requirement R2 and is mainly responsible for the management of scientific workflow provenance metadata, including their representation, storage, archival, searching, and visualization.

The *Data Product Management* subsystem meets requirement R4 and is mainly responsible for the management of heterogeneous data products. One key challenge for data product management is the heterogeneous and potentially distributed nature of data products, making efficient access and movement of data products an important research problem. The interoperability of data products between various workflow environments (requirement R7: level 1) can be addressed in this subsystem.

3.3 Interfaces

Each subsystem interacts with other subsystems by its interfaces. The interoperability between subsystems (requirement R7: level 3) in various SWFMSs should be addressed by standardizing the interfaces provided by each subsystem. In the reference architecture, six interfaces are explicitly defined, which show how the Workflow Engine interacts with other subsystems. The details of the interfaces between subsystems at the same layer are not shown in the figure for simplicity.

Interface I_1 provides a set of interfaces for the communications between Workflow Design subsystem and the Workflow Engine, so workflow specifications created by workflow design tools can be interpreted in the workflow execution environment. Interface I_2 provides a set of interfaces to report workflow run status from the Workflow Engine to the Workflow Monitor and to send back information from the Workflow Monitor to the Workflow Engine when dealing with exceptions, failure, and recovery. Interface I_3 provides a set of interfaces to deal with the communications between the Workflow Engine and the Task Management subsystem: the Workflow Engine

subsystem sends requests to run each task, and the Task Management subsystem replies the task execution progress and acknowledges the Workflow Engine whether a task run completes or fails. *Interface I₄* provides a set of interfaces for communication between the Workflow Engine and the Provenance Management for provenance tracking and reproducibility support. *Interface I₅* provides a set of interfaces between the Workflow Engine and the Data Product Management subsystem: the Workflow Engine requests data product information from the Data Product Management subsystem, and the Data Product Management subsystem responds to the request by acknowledging the availability of the required data product and delivering data or metadata as requested. Finally, *Interface I₆* provides a set of interfaces to interoperate with other Workflow Engines. Workflow specifications can be passed through *I₆* to another Workflow Engine for execution.

3.4 Discussion

Due to the fundamental difference between scientific workflows and business workflows, our proposed reference architecture is significantly different from the reference architecture for BWFMSs. First, the reference architecture for SWFMSs contains the important components of provenance management and data product management to support scientific result reproducibility and to facilitate and speed up data analysis, respectively, which are not present in the reference architecture for BWFMSs. Second, the separation of the Presentation Layer from the Workflow Management Layer enables the support of user interaction and user interface customizability, thus reducing human cycles to scientific discovery. Third, the separation of the Workflow Management Layer from the Task Management Layer separates workflow engineering from task engineering, therefore allowing the parallel advancement of workflow management and task management. Finally, the separation of the Task Management Layer from the Operational Layer enables the separation of management of uniform workflow tasks from the heterogeneous low-level task implementation strategies and execution environments. Such a layered architectural design is important: for computer scientists, it enables abstractions and different independent implementations for each layer; for domain scientists, it provides the opportunity to develop a stable and familiar PSE where rapid technologies can be leveraged but the details of which are shielded transparently from the scientists who need to focus on science itself.

4 SERVICE-ORIENTED ARCHITECTURE FOR VIEW

In order to validate the feasibility of our proposed reference architecture, we propose a service-oriented architecture for our VIEW system that complies with the reference architecture. In this section, we first present our architectural design principles that serve the foundation for the design of the VIEW system; these principles are desirable requirements from a general software engineering perspective rather than requirements specifically essential for SWFMSs. Second, we introduce the overall VIEW system architecture and the architectures of subsystems. Finally, we discuss the advantages of using SOA in SWFMSs. While the architecture of VIEW conforms to the reference architecture, VIEW uses SOA but the reference architecture does not assume

that interfaces between subsystems are services-oriented. While VIEW has a distributed architecture, the reference architecture does not have this assumption. The separation of the reference architecture from the VIEW architecture allows a generic nature of the reference architecture and provides the freedom for VIEW and other systems to take a particular implementation strategy by using the state-of-the-art technologies, such as SOA.

4.1 Architectural Design Principles

The development of the VIEW system complies with the following principles. In addition to the principles described in [3], we have several principles to satisfy the requirements of SWFMSs.

P1: Loose-coupling. In the VIEW system, each subsystem is a loosely coupled, autonomous, reusable, and discoverable service component, and each service component communicates with others by simply requesting their services with the interfaces described by WSDL. Changing the implementation of one service component while remaining the same interfaces does not affect other service components. Service components interact with each other by Web service invocation using SOAP messages via Internet-based protocols.

P2: Localized database access. In the VIEW system, a service component is not allowed to directly access the databases that are managed by other service components; instead, a service component accesses data by requesting services provided by other service components. There are two reasons behind this: 1) databases for each service component are configurable, and provide a more flexible implementation for each subsystem on demand, so different service components are allowed to share the same database or each service component can use their own databases and 2) the design of models and data management for each service component may change, but such changes can be transparent to other service components by using the same interfaces.

P3: Model-based service component. The granularity of services, i.e., how fine or coarse-grained services should be designed, is an important issue for system development. In the VIEW system, the granularity of services is based on the granularity of data models, that is, all operations over one data model is grouped into one service and described by one WSDL, while operations over different data models are separated into different services. In this way, the modification of one data model (e.g., a task model) will not affect the functionality of another data model (e.g., a task run model).

4.2 Overall Architecture and Subsystem Architectures

The overall architecture of VIEW in Fig. 2 consists of six service components that correspond to the main functional subsystems proposed in the reference architecture. Other than *Workbench*, the interface for each service component is defined and described by WSDL: *I_{WE}*, *I_{WM}*, *I_{TM}*, *I_{PM}*, and *I_{DPM}* for the interface of the Workflow Engine, the Workflow Monitor, the Task Manager, the Provenance Manager, and the Data Product Manager, respectively, which comprises the VIEW *Kernel*. In the following, we focus our discussion on the architectural details of the VIEW Kernel.

Workbench. The Workbench subsystem implements the functions of workflow design, presentation, and

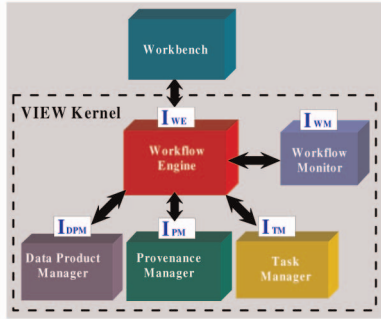


Fig. 2. Overall architecture of the VIEW system.

visualization identified at the Presentation Layer in the reference architecture. Currently, it consists of five components (see Fig. 3a): *Workflow Designer*, *Provenance Explorer*, and the *GUIs* for the VIEW Kernel.

Workflow Designer provides a scientist-friendly GUI for the design and modification of scientific workflows. A scientist can drag and drop registered tasks and data products into the design panel and link them one to another using various dataflow and control flow constructs. *Workflow Designer* is supported by our proposed workflow specification language, called SWL to define a scientific workflow, according to the VIEW *Workflow Model*, which supports hierarchical (nested) scientific workflows. Workflow definitions in the *Workflow Designer* are saved in XML files into a *Local Workflow Repository*. A workflow definition in the VIEW *Workbench* consists of three parts: 1) a *workflow specification* to store the logical structure and its constituent components; 2) *workflow run parameters* to store all parameters for each task run; and 3) a *workflow layout* to store the graphical layout of the scientific workflow that is required to display the workflow in the *Workflow Design Panel*. The first two parts are needed for the execution of a workflow run, and the last part is to display and manipulate a scientific workflow in the VIEW *Design Panel*.

Provenance Explorer enables a user to browse and visualize scientific workflow provenance metadata. Moreover, together with the GUI for the *Data Product Manager*, one can present and visualize various data products from simple data values and plain texts to complex data types.

The VIEW *Workbench* supports Windows-based user interfaces for the VIEW Kernel while reusing the same service components. These scientist-friendly GUIs interact with subsystems via I_{TM} , I_{WM} , I_{PM} , and I_{DPM} , respectively.

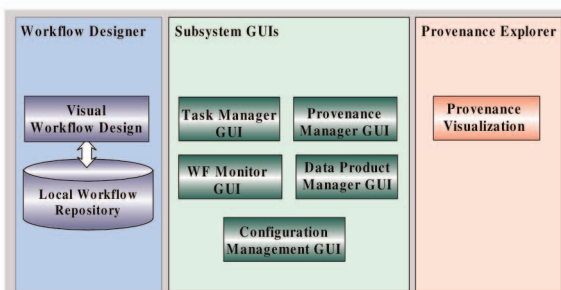
This leads to the architectural flexibility to allow scientists to customize their own GUIs for each particular SWFAS, thus satisfying requirement R1.

Workflow Engine. The architecture of the *Workflow Engine* subsystem is shown in Fig. 3b. Centered around *Scheduler*, the *Workflow Engine* consists of six functional modules: *Scheduler*, *Translator*, *Controlflow Management*, *Dataflow Management*, *Workflow Status Management*, and *Provenance Collector*.

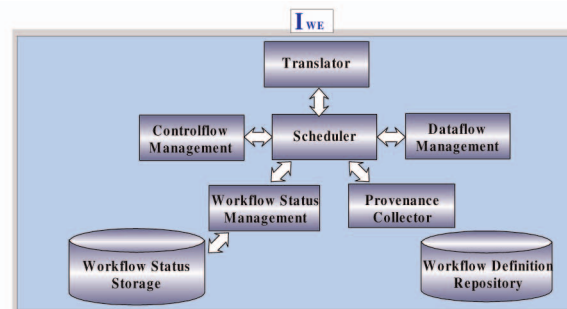
First, *Translator* provides a mapping scheme for translating a workflow specification into an optimized internal executable workflow representation. Workflow definitions delivered from *Workflow Designer* are saved into the *Workflow Definition Repository* via I_{WE} . A workflow definition in *Workbench's* *Workflow Repository* should be consistent with the version in *Workflow Definition Repository* during workflow execution. Second, the separation of controlflow and dataflow management from workflow scheduling greatly improves the extensibility of the VIEW *Workflow Model* since the introduction of additional control flow or dataflow constructs can be achieved by upgrading their individual modules without modifying other modules. Third, as *Scheduler* is able to support multithread processing, it can initialize and maintain a number of workflow runs simultaneously, *Workflow Status Storage* provides a foundation for workflow run monitoring and failure handling (requirement R6). Finally, *Provenance Collector* is responsible for collecting all provenance information and storing them into *Provenance Manager* via I_{PM} . Since the VIEW *Workflow Engine* supports an open and extensible SWL and is loosely coupled with other subsystems, the workflows/subworkflows designed by other SWFMSs can directly request to and invoked by the VIEW *Workflow Engine* via the Web service communication and invocation. Thus, the sharing and mapping between the VIEW *Workflow Engine* and other SWFMSs can be greatly facilitated (requirement R7: level 2).

In contrast to BWFMSs that mostly manage control-flow-oriented workflows, in which the order of task execution is explicitly specified by control flow constructs, such as sequential, conditional, and loop, the VIEW *Workflow Engine* is developed for dataflow-driven scientific workflows. As a result, the availability of input data for a task initiates its execution, and the movement of data via data channels determines the execution order of a workflow.

Workflow Monitor. Our current implementation of the *Workflow Monitor* uses a Publish/Subscribe model [19]



(a)



(b)

Fig. 3. Architecture of (a) the VIEW Workbench and (b) the VIEW Workflow Engine.

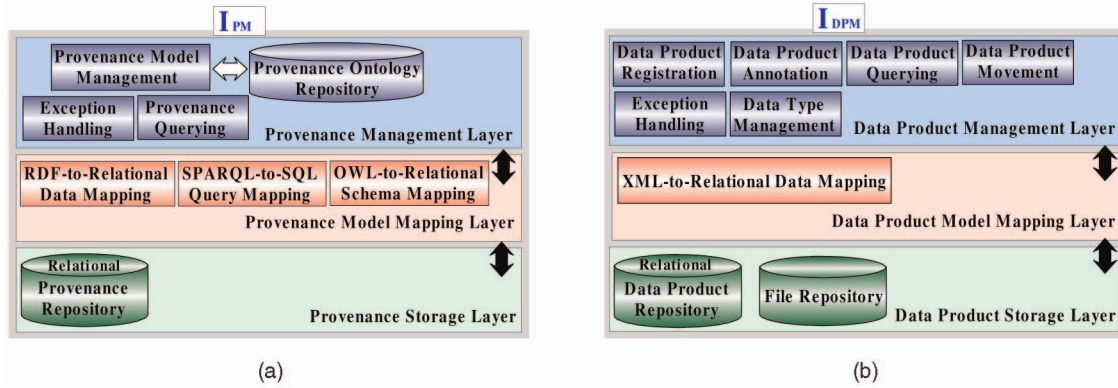


Fig. 4. Architectures of (a) the VIEW Provenance Manager and (b) the VIEW Data Product Manager.

and focuses on the implementation of monitoring workflow execution status. Future implementation will introduce other features including forward and backward recovery in the case of failures. The details of these techniques are out of the scope of this paper.

Provenance Manager. The architecture of the Provenance Manager subsystem shown in Fig. 4a includes three layers: the *provenance management* layer, the *provenance model mapping* layer, and the *provenance storage* layer.

The provenance management layer is responsible for the representation of workflow run provenance via domain ontologies that serve as vocabularies to describe and serialize provenance metadata. It consists of two modules: *provenance model management* and *provenance querying*. Provenance Model Management manages the ontologies that include both general provenance vocabularies and domain-specific ontologies used to represent knowledge in a particular scientific field, e.g., Biology or Physics (requirement R2). To address the requirements of provenance representation interoperability, extensibility, and semantic integration in VIEW, we use Semantic Web technologies for provenance representation. In particular, Web Ontology Language (OWL) is used to express ontologies, and Resource Description Framework (RDF) is used to serialize provenance metadata. Provenance Querying is expressed by RDF query language SPARQL. Exception Handling analyzes all errors reported and implements several strategies to resolve them, so the subsystem can continue functioning.

The provenance model mapping layer serves as an integration medium between the provenance management layer and the provenance storage layer. It currently contains three mappings: 1) *OWL-to-Relational schema mapping* generates a relational database schema based on an ontology that is used to represent provenance metadata; 2) *RDF-to-Relational data mapping* maps provenance metadata in RDF to relational tuples and store them into the relational database; and 3) *SPARQL-to-SQL query mapping* translates provenance queries in SPARQL into relational queries in Structured Query Language (SQL) that can be executed by the Relational Database Management System (RDBMS). The main challenge of this layer is to provide various efficient semantics-preserving mappings between different data models. More details on provenance storage and querying in VIEW are available in [20], where a sample provenance ontology is described and the three mappings are further studied and experimentally evaluated.

The relational model layer includes a relational provenance storage implemented using an RDBMS, which serves as an efficient back end to store and query provenance metadata. In this layer, provenance metadata is stored in relational tables and queried using SQL. The requirements addressed by this layer include efficiency and scalability of provenance metadata management.

Data Product Manager. The architecture of Data Product Manager subsystem shown in Fig. 4b consists of three layers: the *data product management* layer, the *data product model mapping* layer, and the *data product storage* layer.

The data product management layer consists of a set of modules that are responsible for the management of data products based on the VIEW *Data Product Model*. The Data Product Manager allows scientists to access various data products transparently with respect to their heterogeneity and distribution (requirement R4), supported by *Data Product Registration, Annotation, and Querying*. The *Data-Type Management* module defines and manages all required data types to support data storage and task execution. All VIEW subsystems use the same set of data types that are defined by the Data Product Manager, so the introduction of a new data type in *Data-Type Management* becomes effective to all other subsystems. The *Data Product Movement* mainly has three functions: 1) data products are allowed to be moved from client-side to *Data Product Repository* or *File Repository* during data product registration via I_{DPM} ; 2) data products sometimes have to be moved from where they are registered to where a task resides in order to execute the task; and 3) data products produced by workflow execution can be moved back to *Data Product Repository/File Repository*, or registered with the Data Product Manager via I_{DPM} .

The data product model mapping layer serves as an integration medium between the data product management layer and the data product storage layer. The rationale for such an architecture is that for different scientific domains, there are different data product models, implementations, and storage approaches that may require different mapping schema, but sharing the same architecture. One can introduce new data product models, new implementations, or new storage approaches by simply adding new mapping modules in this layer, without affecting modules in other layers.

In the data product storage layer, the Data Product Manager employs a relational *Data Product Repository* to store data products metadata, and *File Repository* to store files, so *XML-to-Relational data mapping* [21], [22] is required

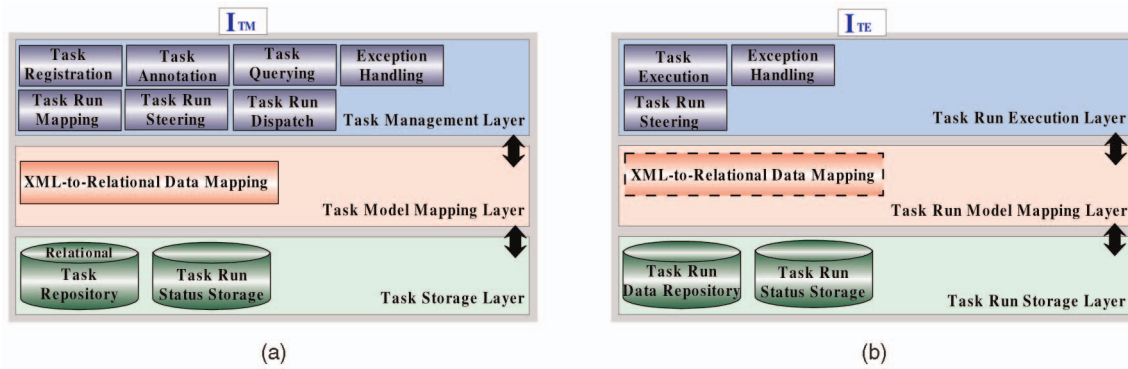


Fig. 5. Architectures of (a) the Task Master and (b) the Task Executor.

in data product model mapping layer to map XML-modeled data products into relational databases.

Task Manager. The VIEW Task Manager supports a distributed architecture, consisting of a Task Master and a set of Task Executors of various types.

Task Master. The architecture of the Task Master shown in Fig. 5a consists of three layers: the *task management* layer, the *task model mapping* layer, and the *task storage* layer. The task management layer provides a set of modules that are responsible for the management of tasks and task runs, based on the VIEW *Task Template Model* and the VIEW *Task Run Model*. The Task Master allows scientists to register/delete a task transparently with respect to its heterogeneity and distribution (requirement R3), which is supported by the modules of *Task Registration*, *Annotation*, and *Querying*. The task run execution and management are handled by the models of *Task Run Steering*, *Mapping*, and *Dispatch*. More specifically, *Task Run Steering* is used to listen to requests from other subsystems to create/abort/pause/resume a task run via *ITM*. *Task Mapping* performs a dynamic mapping from an abstracted task interface to a task component containing a physical implementation, and then delivers the task run to *Task Run Dispatch*, where a Task Executor is dynamically assigned to execute the task component.

In the task model mapping layer, *XML-to-Relational data mapping* [21], [22] is required to map XML-modeled task template specifications into relational *Task Repository*, and map XML-modeled task run descriptors into the relational *Task Status Storage*. The extensions of various mapping mechanisms are allowed to plug-in the task model mapping layer to incorporate heterogeneous data-model storages in the task storage layer (the details of task template specifications and task run descriptors are out of the scope of this paper).

Task Executor. In order to support the distributed execution of tasks in a wide range of heterogeneous environments (requirement R3), a new architectural subsystem called Task Executor is introduced. Task Executor improves the VIEW system in the following aspects:

1. it separates the task and task run management environment in Task Master from the task execution environment in Task Executors;
2. task execution becomes more reliable as tasks can be executed on distributed Task Executors, avoiding the problem of a centralized architecture that may suffer from a single point of failures;

3. different tasks for one workflow can be executed in parallel at distributed nodes to improve performance and efficiency; and
4. the integration of a new type of service or application is achieved by the extension of one Task Executor, without affecting other Task Executors and the Task Master.

The architecture of the Task Executor consists of three layers: the *task run execution* layer, the *task run model mapping* layer, and the *task run storage* layer.

The task run execution layer provides a set of modules that control the task run execution based on the VIEW *Task Run Model*. *Task Run Steering* is used to listen to requests from Task Master to abort/pause/resume a task run via *ITM*. *Task Execution* performs data movement and invokes a task component for a task run.

The implementations at the task run model mapping layer and the task run storage layer vary from the following scenarios: task run status can be simply maintained in the main memory, instead of a persistent storage. However, for large-scale workflows that are composed of a great number of tasks, maintaining a large number of task run status has to rely on a persistent *Task Run Status Storage*, such as a relational database. Then *XML-to-Relational data mapping* [21], [22] is required at the task run model mapping layer to map XML-modeled task run descriptors into relational *Task Run Status Storage*. *Task Run Data Repository* at the task run storage layer is used to save temporary data products that are moved from distributed data sources for a task execution.

4.3 Advantages of Using SOA in SWFMSs

While the emergence of SOA as an architectural paradigm provides many benefits for distributed computing [23], we identify the following advantages of using SOA specifically for the development of an SWFMS:

1. *Service loose coupling*: Service loose coupling minimizes the dependencies among subsystems of an SWFMS by the definitions of a set of language and platform-independent interfaces. In our proposed architecture, each subsystem's functionality is exposed as a Web service. As a result, an SWFMS can be composed on demand from various subsystems provided by different parties as Web services. One can also easily switch from one service to another for each subsystem. For example, there may be several

provenance management services available, and using SOA, one can use and switch any provenance management service on demand for a specific SWFMS.

2. *Service abstraction and autonomy*: A Web service provides an abstract interface that is independent from its implementation. In addition, each Web service is autonomous in the sense that a service provider has the control over the application logic that the Web service encapsulates. As a result, a service provider can dynamically change the implementation and deployment environment of a Web service for a subsystem of an SWFMS with no downtime for the SWFMS as long as such changes do not affect the defined interface. Such autonomy also greatly facilitates the management of the development and evolution of the whole system.
3. *Service reusability*: As each subsystem of an SWFMS becomes a uniform computing unit with standard interface descriptions and universal accessibility through standard communication protocols, it can be reused across various SWFMSs, even simultaneously used by both local SWFMSs and other SWFMSs across the Internet.
4. *Service discoverability*: As each subsystem of an SWFMS is implemented as a Web service that is enriched with a semantic description, one can register the service in some public service registries. As a result, a subsystem becomes discoverable and can be selected and used by other SWFMSs on demand.
5. *Service interoperability*: Service interoperability is enabled by the open standards of messages and communication protocols for Web services, which are supported by a large body of IT industry and the Web Services Interoperability Organization (WS-I). Using Web services, the interoperability across various SWFMSs (requirement R7: level 3) can be greatly improved.

5 SERVICE CONFIGURATION MANAGEMENT

Some SWFMSs are built upon a monolithic system or a centralized database that acts as a single point of failure: when a component of the system or a database fails, there is no way to continue executing workflows. In response to this issue, the VIEW system is composed of a set of loosely coupled and distributed service components, and each service component has multiple alternative services distributed on other machines. Accordingly, there is a need for the management of these services to provide higher system availability. In this section, we present a service configuration management in the VIEW system that provides a flexible configuration functionality for the VIEW Kernel and VIEW Task Executors.

5.1 VIEW Kernel Configuration Management

The VIEW Kernel consists of several loosely coupled service components, and each of them could have multiple backup services deployed on different machines. One service component may have various implementations, but sharing the same WSDL. All of these service components are deployed at distributed environments.

In addition, a database in the VIEW system could specifically serve one service component, or multiple backup service components, or even be shared by several service components of the VIEW Kernel. To support database failover, one serving database could also setup several mirror databases on distributed machines, and each of them is kept synchronized with the serving database, so that once this database fails, its service component(s) can switch to any other mirror databases.

To enable an on-demand VIEW Kernel, the VIEW system provides a service component, called *VIEW Configuration Management*, to manage the configurations of all VIEW Kernel service components and their serving databases. First, the Configuration Management GUI embedded in the VIEW Workbench allows scientists to register all deployed service components for the VIEW Kernel and their serving database(s). The service components for a VIEW Kernel subsystem employ the same WSDL to describe their common interfaces. Second, when the VIEW system is adopted by a specific SWFAS, a template of the VIEW system can be composed on demand by configuring each VIEW Kernel service component and its database(s), which are already registered in the Configuration Management. Third, such template of the system can invoke the chosen services during the runtime. Once a service of the template is unavailable, configuration management will invoke another alternative service. As the alternative service and unavailable service share the same database(s) and repositories in their subsystem storage layer, workflow run and task run status are still valid, which makes it possible to resume the workflow execution starting from the service downtime.

5.2 Task Executor Configuration Management

To support the invocation and execution of various heterogeneous task components, the VIEW Task Manager introduces several types of Task Executors, with each type of Task Executor corresponding to a type of tasks with regard to their programming languages, invocation mechanisms, and computing environments. Each type of Task Executors shares one common implementation that is different from other types. Each Task Executor can be deployed either at the host of the Task Master or at any other standalone host. All Task Executors employ the same architecture and the same WSDL.

To improve the failover in task execution, the VIEW system provides Task Executor Configuration Management to manage the configuration of all Task Executors. First, the Configuration Management GUI embedded in the VIEW Task Manager allows scientists to register all available Task Executors. Second, an appropriate Task Executor can be automatically assigned to a task by the VIEW Task Master during task execution. Finally, if a Task Executor happens to be unavailable during task runtime, an alternative Task Executor will be chosen to retry the execution. In this case, the whole scientific workflow does not need to be aborted or restarted, as other Task Executors will not be disturbed during the failover procedure of the failed Task Executor.

6 SYSTEM IMPLEMENTATION AND CASE STUDIES

In this section, we first present the VIEW system implementation and deployment, and then we introduce

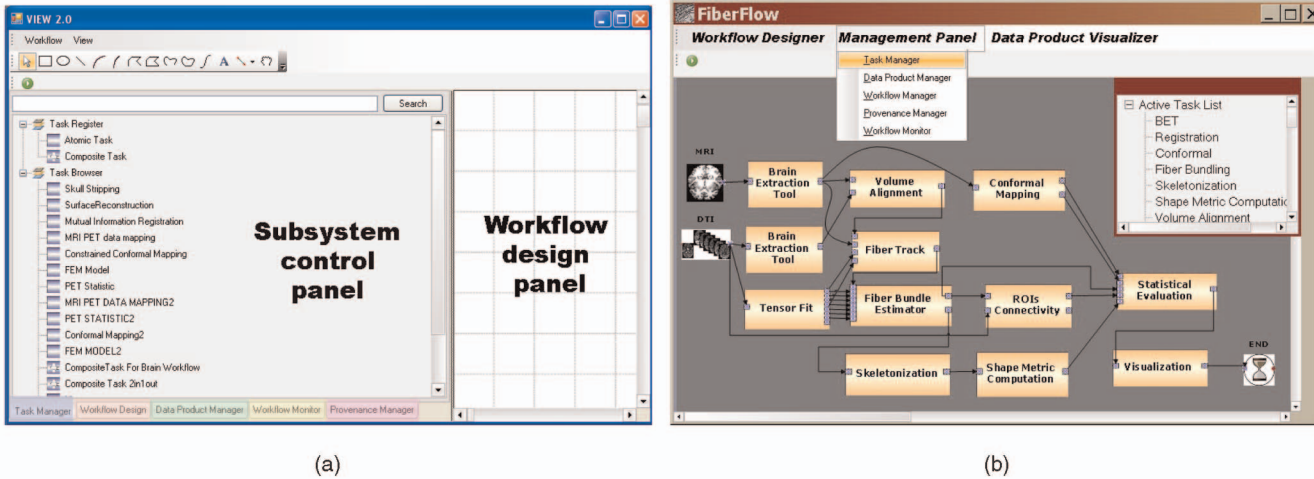


Fig. 6. (a) The VIEW Workbench embedded with five subsystem control panels and a workflow design panel, and (b) a user-interaction-intensive and visualization-intensive scientific workflow displayed in a customized GUI for the VIEW-based FiberFlow system.

an SWFAS, called FiberFlow, to showcase an application of the VIEW system.

6.1 System Implementation and Deployment

We implemented the VIEW system using the Microsoft Visual Studio 2005 and the Microsoft .NET Framework. First, we reused existing components of our previous VIEW prototype [24] by upgrading original codes from Native VC++ to Managed C++, which can interoperate with all other .NET capable languages. Second, we restructured the VIEW system according to the proposed architectures and introduced new service components: the Workflow Monitor, the Task Manager, the Task Executor, the Data Product Manager, and the Service Configuration Management. Third, we implemented a set of operations that need to be exposed as Web services for each service component. Fourth, we manipulated and transported XML-format data among memory, files, and databases using the ADO.NET and the SQL Server 2005. Finally, we enhanced the VIEW Workbench in its presentation and visualization capability using VTK and OpenGL.

The VIEW system allows a flexible deployment for a specific SWFAS. The VIEW Kernel can be deployed as a whole in one machine, or each service component can be deployed in a distributed node. To enhance fault tolerance of the system, a service component could have multiple backup services deployed at different machines, and each backup service has its own distinct URL available on the Internet. In case that one service is down, the VIEW system can dynamically switch to an alternative service.

6.2 VIEW-Based FiberFlow System

VIEW is an *application-independent* SWFMS, serving as a foundation on which various SWFASs can be developed according to their own domain-specific requirements. The FiberFlow system is such an SWFAS developed for automatic transforming the large-scale neuroimaging data to knowledge through cross-subject, cross-modality computation, ultimately leading to high clinical intelligence and more informed and accurate decision making in various neural diseases.

The complexity of workflows in the FiberFlow system poses the grand challenges which can be summarized in the

following three aspects. First, each workflow may produce a large amount of processed data under different experimental parameter settings. All analytical results and intermediate results vary from data types and formats, and therefore generate great challenges for data management. Second, some computation-intensive tasks are required to be performed on distributed Grid environments in order to reduce the wait time. Third, most tasks in FiberFlow are also interaction-intensive and visualization-intensive, as domain scientists need to frequently manipulate, process, and evaluate the imaging data. Due to the aforementioned challenges, an SWFMS is ideal to manage all the research artifacts to speed up the effective FiberFlow exploratory process.

Fig. 6b demonstrates a typical workflow designed in the FiberFlow system which is to automate population-based statistical analysis of the variances of fiber bundle shapes and fiber connectivity strengths among normal individuals and patients [25]. Most data products involved in this workflow are volume image files, such as Magnetic Resonance Imaging (MRI) and Diffusion Tensor Imaging (DTI). They are stored in the *Analyze* format, which is one of the standard formats for 3D medical imaging. All tasks in this workflow are implemented as Windows-based applications, and some of them require client-side user interactions to identify *Regions of Interests* (ROIs).

In a nutshell, the first task is to perform segmentation using the *Brain Extraction Tool* (BET), which segments a subject's neocortex based on DTI and MRI imaging data, and outputs new volume image files with the subject's skull being stripped away. These files become the input for the *Volume Alignment* (VA) task. This task conducts spatial mappings between the two skull-stripped DTI and MRI files and generates a text file containing a matrix of all mapping parameters. Besides BET, another preprocessing task is called *Tensor Fit* (TF). It computes tensor fields using DTI data and generates various invariant metrics which, together with other outputs derived from BET and VA, are inputs of the *Fiber Tracking* task for fiber tractography [26]. To derive a population-based statistics on human brains of varying sizes and shapes, the *Conformal Mapping* (CM) task is applied to perform an intersubject registration, which maps skull-stripped MRI to a common 3D template space.

Meanwhile, the text-based output from the Fiber Track task, together with all volume files from TF, are applied with the Fiber Bundle Estimator task to identify the specific fiber bundle of interest and computes its isosurface. The user interactions for picking up ROIs are required in this task. Then, a volume file recording ROI bundles is produced and supplied to the *ROIs Connectivity* (RC) task. Based on the DTI imaging, RC creates a probabilistic model based on the Bayesian inference theory and estimates connectivity between ROIs. The task's output recording 3D fiber bundles becomes the input of *Skeletonization*, and the result from Skeletonization is supplied to the *Shape Metric Computation* (SMC) task to generate quantitative shape descriptors. Finally, the *Statistical Evaluation* task collects all data products generated from CM, RC, and SMC to produce statistical results using the t-distribution. The statistical results can be visualized by the *Visualization* task, using the MRI as the spatial context with the affected tissues labeled and the statistical variations colored with different colormaps.

The service-oriented architecture of VIEW enables a fast and convenient development of the FiberFlow system, by customizing subsystem GUIs, while reusing the underlying VIEW Kernel. Some customizations are performed on the FiberFlow Workbench without changing any source codes of the VIEW Workbench (see Fig. 6b). For example, the VIEW Workbench provides the options for users to choose which subsystem GUI to be displayed on the control panel. Users can also customize icons, menus, font size, and color in the workflow design panel to make the look-and-feel consistent with other nonworkflow-based subsystems embedded in FiberFlow. We plan to collect more use case scenarios to improve the flexibility and configurability of customizing the VIEW Workbench for different scientific domains. By reusing the VIEW Kernel, the Task Manager and the Data Product Manager manage a library of software tools and data products for this domain. The Workflow Engine manages our entire set of FiberFlow workflows composed by existing software tools and data products. The Workflow Monitor is used to monitor workflow execution progress, and the Provenance Manager is employed for provenance management, on which provenance mining and provenance visualization are being conducted for our domain-specific analysis.

7 EVALUATING REPRESENTATIVE SWFMSs USING THE REFERENCE ARCHITECTURE

In this section, we evaluate five representative SWFMSs using the proposed reference architecture: Taverna [27], Kepler [28], Triana [14], Pegasus [12], and Swift [11]. The analysis is performed based on the seven key architectural requirements in the context of the proposed reference architecture. Our evaluation criteria are as follows: if a system provides a full support to the specified requirement, a score of "+" will be assigned; if no support is provided to a particular requirement, a score of "-" will be assigned; a partial score of "+/-" will be assigned to a system when the support is clearly partial or when there is an ambiguity associated with such support. With respect to each requirement, we also describe a summary of the five systems to shed some lights on the state of the art. We have left out our own VIEW system in this study to avoid a biased evaluation.

| Requirements | Taverna | Kepler | Triana | Pegasus | Swift |
|--------------|---------|--------|--------|---------|-------|
| R1 | +/- | +/- | +/- | - | - |
| R2 | + | + | + | + | + |
| R3 | +/- | +/- | +/- | - | - |
| R4 | - | +/- | - | - | +/- |
| R5 | +/- | +/- | +/- | + | + |
| R6 | +/- | +/- | +/- | +/- | +/- |
| R7 | - | - | - | - | - |

Fig. 7. Architectural evaluation of five SWFMSs.

The evaluation results are presented in Fig. 7. We observe that Pegasus and Swift provide weak user interaction support (R1), mainly due to the technical challenge of supporting interaction in a batch-based grid system, while Taverna, Kepler, and Triana provide better user interaction support. Currently, almost all these systems have poor support to user interface customizability (R1) due to the tightly coupled nature between system interfaces and runtime subsystems. All the five systems currently support provenance (R2), emphasizing the importance of provenance in SWFMSs. However, since the provenance module is closely coupled with its owner SWFMS in these systems, reuse of the provenance subsystem across other SWFMSs is difficult. Taverna, Kepler, and Triana have partial support to the integration of heterogeneous services and software tools (R3), while Pegasus and Swift focus only on grid-based applications. Therefore, a general framework that can provide an abstraction of heterogeneous services and applications as workflow tasks is still missing. Such a framework needs to clearly separate abstraction of a workflow task from its implementation and provides mapping and binding mechanisms between the inputs/outputs of a workflow task to the inputs/outputs to its wrapped service and application component. Although the importance of data management has been recently emphasized in the scientific workflow community [29], data product management (R4), particularly the abstraction of logical data products with transparent data set representations, formats, and locations, is relatively an unexplored area in the scientific workflow community. For example, Pegasus and Swift mainly work at the level of files, while Taverna and Kepler work at the levels of XML messages, files, and database records. Only few systems provide some limited support to the abstraction: Kepler supports the notion of Nested Data Collections by using custom collection-oriented actors (coactors), while Swift introduces the XDTM notation to define a mapping between the logical organization and the underlying physical structure of data sets, which are limited to files and directories so far. Currently, Pegasus and Swift have better support to high-end computing (R5); in the meanwhile, other systems are being enhanced in such support: Kepler and Taverna provide custom tasks to communicate with the Grid environment, while Triana uses the GAT interface to access Grid jobs. One challenge for data management is how to avoid the movement of large amounts of data back and forth from a Workflow Engine to the Grid environment, while seamlessly integrating workflow tasks that are services-based and Grid-based applications. All these five SWFMSs currently provide some degree of support to workflow

monitoring and failure handling (R6); however, failure handling for large-scale and distributed scientific workflows remains a challenge. Finally, interoperability (R7) is poorly supported in all these SWFMSs, although some limited pairwise interoperability has been investigated. A community-based initiative such as the Open Provenance Model [30] is a good effort toward this direction, and we expect that interoperability will become more important when more and more scientific projects become collaborative and need the integration of multiple SWFMSs.

8 RELATED WORK

Although the term “scientific workflows” were first coined by Vouk and Singh in 1996 [31] for workflow applications in scientific PSEs, only recently, there is an increasing momentum for the research and development of SWFMSs and their applications, due to the increasingly demanding requirements of many compute-intensive and data-intensive scientific applications, enabled by the underlying advances of computing technologies, notably Services computing [13], Grid computing [32], and Cloud computing [17]. Scientific workflows leverage existing techniques developed for business workflows but deviate from them as a result of a different set of requirements raised from a wide range of science and engineering problems [33]. While business workflows are control flow oriented with the mission of carrying out business logic to achieve a business goal, scientific workflows tend to be dataflow oriented and aimed at enabling, facilitating, and speeding up the derivation of scientific results from raw data sets.

Although the reference architecture proposed by the WfMC [2] has been well adopted in the development of different BWFMSs, including the recent development of the YAWL system [4] that is aimed at exploring various workflow patterns [34]. Existing architectures for BWFMSs are not appropriate for SWFMSs since business workflows are typically control flow oriented, while scientific workflows tend to be dataflow oriented, introducing a new set of requirements and challenges for system development, from the requirements of intensive user interaction, customized user interface, reproducibility, high-end computing, interoperability, to heterogeneous data product, service, and application management. In particular, this reference architecture does not satisfy the key requirements from R1 to R5 for an SWFMS.

Several SWFMSs have been developed over the past few years. Although some of them provide architectures, they are system and domain specific and fail to satisfy some of the key architectural requirements for SWFMSs identified in Section 2. The Kepler system [28] is a Java-based open source SWFMS. Kepler is built upon the Ptolemy II system and features a monolithic architecture with various extension modules for functionalities needed for scientific workflows [28]. The Taverna system [27] is another Java-based open source SWFMS mainly targeted for life science. Taverna features a three-layered architecture [27]: The *Application Data Flow* layer provides a user’s perspective of a workflow, hiding the complexity of interoperation of services; the *Execution Flow* layer is responsible for workflow scheduling, service discovery, data, and metadata management; and the *Processor Invocation* layer is responsible for the invocation of concrete services. The Triana

system [14] has a sophisticated graphical user interface for workflow composition and modification, including grouping, editing, and zooming functions. Coming from the gravitational wave field, the system contains a large repository of tools for data analysis and processing. The VisTrails system [10] is developed to manage visualizations and is the first system that supports provenance tracking of workflow evolution in addition to tracking the data product derivation history. The Pegasus system [12] provides a framework which maps complex scientific workflows onto distributed grid resources. Artificial intelligence planning techniques are used in Pegasus for workflow composition. Finally, the Swift system [11] combines a novel scripting language called *SwiftScript* with a powerful runtime system to support the specification and execution of large loosely coupled computations over Grid environments.

Although these SWFMSs provide much experience in future research and development, a study from an architectural perspective is still missing. The lack of a reference architecture prevents the interoperability between different systems and limit the reusability, flexibility, and extensibility of systems, as well as the configuration of heterogeneous modular subsystems. As a result, it is hard to execute one scientific workflow across different SWFMSs, and a subsystem in one SWFMS is not able to be reused by another SWFMS, even though they provide the same functionality. Therefore, there is a pressing need for an architectural reference that can provide a high-level organization of subsystems and their interactions in an SWFMS. The availability of such a reference architecture cannot only provide a guidance for the architectural design of a particular SWFMS, but also provide a basis to assess and compare various SWFMSs and promote the interoperability between different systems.

Scientific workflow systems can be seen as one kind of dataflow-oriented PSEs [31], which provide computational facilities needed to solve a target class of problems. Scientific workflow systems are related to and bear some features of dataflow-based visualization systems, such as AVS, IBM’s Data Explorer, and SCIRun. From a software engineering perspective, scientific workflows can be seen as one methodology for component-based software engineering, where workflow tasks are the *abstraction* of software components, and a scientific workflow application is developed by the construction of workflow tasks and their composition. As most scientific workflow systems provide user-friendly interfaces for designing scientific workflows, scientific workflows are closely related to the field of visual programming languages, where not only the components and the structure of scientific workflows need to be described but also the layout of workflows in display should be precisely defined.

Our proposed reference architecture aims to be technology-independent for long-term sustainability. The VIEW SOA solution is an instance of this reference architecture in the light of the well-known S3 SOA reference architecture [35]. As a result, our VIEW SOA solution is an enhancement of the reference architecture with all the advantages brought by the SOA technology.

This paper extends [36] with the following additional contributions:

1. the architecture of the Data Product Manager is provided in details;
2. in Section 4.2, we introduce a new architectural component, called *Task Executor*, to support task execution in heterogeneous and distributed environments;
3. we add an innovative Configuration Management service to dynamically configure, invoke, and manage VIEW subsystems and VIEW Task Executors, resulting in a new section, Section 5;
4. a newly developed SWFAS, called FiberFlow, is introduced in Section 6 to showcase an application of the VIEW system.

9 CONCLUSIONS AND FUTURE WORK

In this paper, we proposed the first reference architecture for SWFMSs and presented an SOA solution for the VIEW system. We implemented VIEW to validate the feasibility of the proposed architectures and introduced an SWFAS, the FiberFlow system, to showcase the applications of our VIEW system. Ongoing work includes the extension of our architecture to address security issues, a more comprehensive evaluation of the VIEW system, and the development of various SWFASs using VIEW.

ACKNOWLEDGMENTS

The architects of this paper were Cui Lin and Shiyong Lu. The developers of the VIEW Kernel were Cui Lin, Xubo Fei, and Artem Chebotko. The developers of the VIEW Workbench were Cui Lin, Xubo Fei, and Zhaoqiang Lai. The developers of the FiberFlow system were Cui Lin and Darshan Pai. The advisors of this paper were Shiyong Lu, Farshad Fotouhi, and Jing Hua. This research was supported in part by Michigan Technology Tri-Corridor basic research grant MTTC05-135/GR686 and US National Science Foundation grant IIS-0713315. The authors give thanks to Mr. Chunhyeok Lim, who was involved in the implementation of the VIEW Data Product Manager.

REFERENCES

- [1] A. Tsalgatidou, G. Athanopoulos, M. Pantazoglou, C. Pautasso, T. Heinis, R. Grønmo, H. Hoff, A. Berre, M. Glittum, and S. Topouzidou, "Developing Scientific Workflows from Heterogeneous Services," *SIGMOD Record*, vol. 35, no. 2, pp. 22-28, 2006.
- [2] D. Hollingsworth, *Workflow Management Coalition Specification: The Workflow Reference Model*, Document Number TC00-1003, v. 1.1, 1995.
- [3] P. Grefen and R. de Vries, "A Reference Architecture for Workflow Management Systems," *Data Knowledge Eng.*, vol. 27, no. 1, pp. 31-57, 1998.
- [4] W. van der Aalst, L. Aldred, M. Dumas, and A. ter Hofstede, "Design and Implementation of the YAWL System," *Proc. Center for Advancement of Informal Science Education Conf. (CAiSE '04)*, pp. 142-159, 2004.
- [5] L. Liu, C. Pu, and D. Ruiz, "A Systematic Approach to Flexible Specification, Composition, and Restructuring of Workflow Activities," *J. Database Management*, vol. 15, no. 1, pp. 1-40, 2004.
- [6] J. Miller, D. Palaniswami, A. Sheth, K. Kochut, and H. Singh, "Webwork: METEOR₂'s Web-Based Workflow Management System," *J. Intelligent Information Systems*, vol. 10, no. 2, pp. 185-215, 1998.
- [7] G. Alonso, R. Günthör, M. Kamath, D. Agrawal, A. Abbadi, and C. Mohan, "Exotica/FMDC: A Workflow Management System for Mobile and Disconnected Clients," *Distributed and Parallel Databases*, vol. 4, no. 3, pp. 229-247, 1996.
- [8] F. Leymann and D. Roller, "Business Process Management with FlowMark," *Proc. IEEE CS Int'l Conf. (COMPCON '94)*, pp. 230-234, 1994.
- [9] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. Lee, J. Tao, and Y. Zhao, "Scientific Workflow Management and the Kepler System," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1039-1065, 2006.
- [10] S. Callahan, J. Freire, E. Santos, C. Scheidegger, C. Silva, and H. Vo, "VisTrails: Visualization Meets Data Management," *Proc. Special Interest Group on Management of Data Conf. (SIGMOD '06)*, pp. 745-747, 2006.
- [11] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. von Vonlaszewski, I. Raicu, T. Stef-Praun, and M. Wilde, "Swift: Fast, Reliable, Loosely Coupled Parallel Computation," *Proc. IEEE Int'l Workshop Scientific Workflows (SWF '07)*, pp. 199-206, 2007.
- [12] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. Berriman, J. Good, A. Laity, J. Jacob, and D. Katz, "Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems," *Scientific Programming J.*, vol. 13, no. 3, pp. 219-237, 2005.
- [13] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, R. Greenwood, T. Carver, K. Glover, M. Pocock, A. Wipat, and P. Li, "Taverna: A Tool for the Composition and Enactment of Bioinformatics Workflows," *Bioinformatics*, vol. 20, no. 17, pp. 3045-3054, 2004.
- [14] S. Majithia, M. Shields, I. Taylor, and I. Wang, "Triana: A Graphical Web Service Composition and Execution Toolkit," *Proc. IEEE Int'l Conf. Web Services (ICWS '04)*, pp. 514-524, 2004.
- [15] L. Zhang, J. Zhang, and H. Cai, *Services Computing*. Springer, 2007.
- [16] C. Lin and S. Lu, "Architectures of Workflow Management Systems: A Survey," Technical Report TR-SWR-01-2008, 2008.
- [17] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," *Proc. IEEE Grid Computing Environments Workshop*, pp. 1-10, 2008.
- [18] D. Georgakopoulos, M. Hornick, and A. Sheth, "An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure," *Distributed and Parallel Databases*, vol. 3, no. 2, pp. 119-153, 1995.
- [19] K. Ostrowski, K. Birman, and D. Dolev, "Extensible Architecture for High-Performance, Scalable, Reliable Publish-Subscribe Eventing and Notification," *Int'l J. Web Service Research*, vol. 4, no. 4, pp. 18-58, 2007.
- [20] A. Chebotko, X. Fei, C. Lin, S. Lu, and F. Fotouhi, "Storing and Querying Scientific Workflow Provenance Metadata Using an RDBMS," *Proc. Second IEEE Int'l Workshop Scientific Workflows and Business Workflow Standards in E-Science*, pp. 611-618, 2007.
- [21] M. Atay, A. Chebotko, D. Liu, S. Lu, and F. Fotouhi, "Efficient Schema-Based XML-to-Relational Data Mapping," *Information Systems*, vol. 32, no. 3, pp. 458-476, 2007.
- [22] A. Chebotko, M. Atay, S. Lu, and F. Fotouhi, "XML Subtree Reconstruction from Relational Storage of XML Documents," *Data Knowledge Eng.*, vol. 62, no. 2, pp. 199-218, 2007.
- [23] T. Erl, *Service-Oriented Architecture Concepts, Technology and Design*. Pearson Education, Inc., 2005.
- [24] A. Chebotko, C. Lin, X. Fei, Z. Lai, S. Lu, J. Hua, and F. Fotouhi, "VIEW: A Visual Scientific Workflow Management System," *Proc. IEEE Int'l Workshop Scientific Workflows (SWF '07)*, pp. 207-208, 2007.
- [25] D. Pai, O. Muzik, and J. Hua, "Quantitative Analysis of Diffusion Tensor Images Across Subjects Using Probabilistic Tractography," *Proc. Int'l Conf. Image Processing (ICIP '08)*, pp. 1448-1451, 2008.
- [26] C. Lin, S. Lu, X. Liang, J. Hua, and O. Muzik, "Coclust Analysis of Thalamo-Cortical Fiber Tracts Extracted from Diffusion Tensor MRI," *Int'l J. Data Mining and Bioinformatics*, vol. 2, no. 4, pp. 342-361, 2008.
- [27] T. Oinn, M. Greenwood, M.J. Addis, M.N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D.J. Marvin, P. Li, P. Lord, M.R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe, "Taverna: Lessons in Creating a Workflow Environment for the Life Sciences," *J. Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1067-1100, 2002.
- [28] I. Altintas, O. Barney, and E. Jaeger-Frank, "Provenance Collection Support in the Kepler Scientific Workflow System," *Proc. Int'l Provenance and Annotation Workshop (IPAW '06)*, pp. 118-132, 2006.
- [29] E. Deelman and A. Chervenak, "Data Management Challenges of Data Intensive Scientific Workflows," *Proc. IEEE Int'l Symp. Cluster Computing and the Grid (CCGRID '08)*, pp. 687-692, 2008.

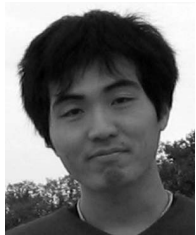
- [30] *Open Provenance Model*, <http://twiki.ipaw.info/bin/view/Challenge/OPM>, 2009.
- [31] M. Vouk and M. Singh, "Quality of Service and Scientific Workflows," *Proc. Working Conf. Quality of Numerical Software*, pp. 77-89, 1996.
- [32] J. Yu and R. Buyya, "A Taxonomy of Scientific Workflow Systems for Grid Computing," *SIGMOD Record*, vol. 34, no. 3, pp. 44-49, 2005.
- [33] I. Taylor, E. Deelman, D. Gannon, and M. Shields, *Workflows for E-Science*. Springer-Verlag London, Ltd., 2007.
- [34] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros, "Workflow Patterns," *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 5-51, 2003.
- [35] A. Arsanjani, L.-J. Zhang, M. Ellis, A. Allam, and K. Channabasaiah, "S3: A Service-Oriented Reference Architecture," *IT Professional*, pp. 10-17, 2007.
- [36] C. Lin, S. Lu, Z. Lai, A. Chebotko, X. Fei, J. Hua, and F. Fotouhi, "Service-Oriented Architecture for VIEW: A Visual Scientific Workflow Management System," *Proc. IEEE Int'l Conf. Services Computing (SCC '08)*, pp. 335-342, 2008.



Cui Lin received the BE degree from Beijing Information Technology Institute. She is currently working toward the PhD degree at the Department of Computer Science, Wayne State University. She spent a few years working for IBM as a system analyst and the project manager, and certified as an IBM DB2 expert. She is currently a member of the Scientific Workflow Research Laboratory (the SWR Lab). Her research interests include scientific workflows, services computing, and bioinformatics. She has published several refereed international journals and conference papers, and currently serves as a publication chair for an international workshop on scientific workflows. She is a student member of the IEEE.



Shiyong Lu received the PhD degree from the State University of New York at Stony Brook in 2002. He is currently an assistant professor in the Department of Computer Science, Wayne State University, and the director of the Scientific Workflow Research Laboratory. His research interests include scientific workflows and databases. He has published more than 70 refereed international journal and conference papers in the above areas. He is the founder and currently a cochair of the IEEE International Workshop on Scientific Workflows, an editorial board member for *International Journal of Medical Information Systems and Informatics*, and for *International Journal of Semantic Web and Information Systems*. He also serves as a program committee member for several top-tier IEEE conferences including SCC and ICWS. He is a member of the IEEE.



Xubo Fei is currently working toward the PhD degree in the Department of Computer Science, Wayne State University. His current research interests include scientific workflows and their applications in bioinformatics and biology simulation.



Artem Chebotko received the BS degree in computer science from Ukraine State Maritime Technical University, the MS degree in management information systems from Ukraine State Maritime Technical University, and the PhD degree in computer science from Wayne State University. He is an assistant professor in the Department of Computer Science, University of Texas—Pan American. His research interests include scientific workflow provenance metadata management and semantic web data management. He currently serves as a program committee member of two international workshops on scientific workflows. He is a member of the ACM and the IEEE.



Darshan Pai received the MS degree in computer engineering with a specialization in software systems from Wayne State University. He is currently working toward the PhD degree at the Department of Computer Science, Wayne State University. Currently he is a part of the Graphics and Imaging Lab conducting research in 3D medical image visualization and analysis. He is also a research contributor to the School of Medicine, Wayne State University. He is a student member of the IEEE.



Zhaoqiang Lai received the BS and MS degrees from Huazhong University of Science and Technology in 2003 and 2006, respectively. He is currently working toward the PhD degree at the Department of Computer Science, Wayne State University, and a research assistant in the Graphics and Imaging Laboratory. His research interests include computer graphics and visualization.



Farshad Fotouhi received the PhD degree in computer science from Michigan State University in 1988. In August 1988, he joined the faculty of Computer Science at Wayne State University, where he is currently a professor and the chair of the department. His major areas of research include XML databases, semantic Web, multimedia systems, and query optimization. He has published more than 100 papers in refereed journals and conference proceedings, served as a program committee member of various database-related conferences. He is on the Editorial Boards of the *IEEE Multimedia Magazine* and *International Journal on Semantic Web and Information Systems*. He is a member of the IEEE.



Jing Hua received the MS and PhD degrees in computer science from the State University of New York at Stony Brook in 2002 and 2004, respectively. He is an assistant professor of computer science at Wayne State University and the director of the Graphics and Imaging Lab. His research interests include computer graphics, visualization, biomedical imaging, and informatics, and his research is currently funded by the US National Science Foundation, the NIH, and the Michigan State Foundations. He serves as an editorial board member for *Scientific Journals International* and *International Journal of Technology Enhanced Learning* and a program committee member for many international conferences. He is a member of the IEEE.