VizSciFlow: A Visually Guided Scripting Framework for Supporting Complex Scientific Data Analysis

MUHAMMAD M. HOSSAIN, University of Saskatchewan, Canada
BANANI ROY, University of Saskatchewan, Canada
CHANCHAL K. ROY, University of Saskatchewan, Canada
KEVIN A. SCHNEIDER, University of Saskatchewan, Canada

Scientific workflow management systems such as Galaxy, Taverna and Workspace, have been developed to automate scientific workflow management and are increasingly being used to accelerate the specification, execution, visualization, and monitoring of data-intensive tasks. For example, the popular bioinformatics platform Galaxy is installed on over 168 servers around the world and the social networking space myExperiment shares almost 4,000 Galaxy scientific workflows among its 10,665 members. Most of these systems offer graphical interfaces for composing workflows. However, while graphical languages are considered easier to use, graphical workflow models are more difficult to comprehend and maintain as they become larger and more complex. Text-based languages are considered harder to use but have the potential to provide a clean and concise expression of workflow even for large and complex workflows. A recent study showed that some scientists prefer script/text-based environments to perform complex scientific analysis with workflows. Unfortunately, such environments are unable to meet the needs of scientists who prefer graphical workflows. In order to address the needs of both types of scientists and at the same time to have script-based workflow models because of their underlying benefits, we propose a visually guided workflow modeling framework that combines interactive graphical user interface elements in an integrated development environment with the power of a domain-specific language to compose independently developed and loosely coupled services into workflows. Our domain-specific language provides scientists with a clean, concise, and abstract view of workflow to better support workflow modeling. As a proof of concept, we developed VizSciFlow, a generalized scientific workflow management system that can be customized for use in a variety of scientific domains. As a first use case, we configured and customized VizSciFlow for the bioinformatics domain. We conducted three user studies to assess its usability, expressiveness, efficiency, and flexibility. Results are promising, and in particular, our user studies show that VizSciFlow is more desirable for users to use than either Python or Galaxy for solving complex scientific problems.

CCS Concepts: • **Human-centered computing → User interface management systems**; **User centered design**; • **Applied computing → Bioinformatics**.

Additional Key Words and Phrases: Scientific Data Analysis; Workflow Modeling; Workflow Language; DSL

ACM Reference Format:
Muhammad M. Hossain, Banani Roy, Chanchal K. Roy, and Kevin A. Schneider. 2020. VizSciFlow: A Visually Guided Scripting Framework for Supporting Complex Scientific Data Analysis. *Proc. ACM Hum.-Comput. Interact.* 4, EICS, Article 74 (June 2020), 37 pages. https://doi.org/10.1145/3394976

Authors' addresses: Muhammad M. Hossain, University of Saskatchewan, Saskatoon, SK, Canada; Banani Roy, University of Saskatchewan, Saskatoon, SK, Canada; Chanchal K. Roy, University of Saskatchewan, Saskatoon, SK, Canada; Kevin A. Schneider, University of Saskatchewan, Saskatoon, SK, Canada.

2573-0142/2020/6-ART74 $15.00
https://doi.org/10.1145/3394976

1 INTRODUCTION

A workflow captures a set of abstract descriptions of execution steps for a scientific process. Each step involves a data transformation or analysis task based on a set of rules along with the instructions for its execution [71]. Multiple human or computing resources may take part in the processing as data are passed from one resource to another [11]. Scientists often use a workflow language to author a model that specifies the orchestration and order of execution of the services. Specifying complex scientific experiments using workflows is becoming more common. Consequently, a number of generic software tools called Scientific Workflow Management Systems (SWfMS) have emerged that allow specification, registration, execution, visualization and monitoring of scientific workflows [4, 30, 40, 63, 78, 118].

SWfMSs offer text-based or graphical languages to model workflows. Both approaches have strengths and weaknesses. Initially, program comprehension is usually easier with a graphical language. But the model often becomes confusing as the complexity of the workflow increases. It is hard to manage a multiplex layout of wires and elements, follow data or control flow and maintain multiple versions of a large graphical model.

Many scientists prefer text-based programming as it is conducive to rapid prototyping and provides a more compact representation of complicated workflows [76]. Some researchers also argue that a textual program is easy to scan [46]. But scientists usually lack a deep understanding of complex programming constructs. We noticed that those scientists struggle to use general-purpose scientific programming languages such as Python. For example, they have difficulty correctly ordering tool-specific arguments. As well, general-purpose approaches have no mechanisms for automatically generating logs or capturing output [53]. As scientific data grows exponentially, it is more difficult to get time-critical analysis results. Running tools in a high performance computing environment often requires specialized knowledge such as learning the Application Programming Interface (API) of complex distributed systems such as Spark. There is a gap between a scientist's domain knowledge and complex programming constructs, which can be overwhelming and overly complex when using a language like Python, and hard to scale and maintain when using a strictly visual approach. To address these deficiencies and bridge the gap between a scientist's domain knowledge and complex programming constructs, we develop a framework for SWfMSs which incorporates an interactive IDE for specifying scientific workflows with a domain-specific language (DSL). The DSL provides a minimal set of language features and limited vocabulary specific to a scientist's domain knowledge. An evolutionary approach helps to enrich the domain-specific vocabulary in an iterative and incremental manner.

The efficiency of workflow modeling depends on various factors such as the formal syntax of the language, the experience and expertise of the users, effective use of secondary notations [92], and the usability of the supported development tools. The secondary notations refer to the ability of a language system to add extra information to the formal syntax of the language [105] to increase the comprehensibility of a model or program. Indentation, comments, grouping of related elements are some examples of textual secondary notations. On the other hand, the development tools are the auxiliary features that are supported by the language system to ease the authoring of a model. An Integrated Development Environment (IDE) can offer many development tools such as interactive UI elements for automatic pipeline creation, syntax highlighting, automatic indenting, automatic code completion, testing and debugging services, monitoring and visualization facilities, interface for sharing and collaboration of data, services and workflows. For example, in our proposed graphical environment for workflow modeling, workflow artifacts are represented as visual elements which can be converted into code snippets and inserted into a code editor with mouse and keyboard interactions.

As a proof of concept of our proposed framework, we developed a customizable SWfMS – VizSciFlow. Since our framework is not restricted to any particular research domain, we evaluate it for the bioinformatics domain. We chose this domain because it requires numerous scientific analyses of a complex nature. Furthermore, while there are a great many bioinformatics tools (e.g., FastQC [7], PEAR [122], BWA-MEM [67], Flash [75], and SAMtools [68]), they are not intuitive to use and scientists need to have considerable expertise in using these tools to create workflows with existing systems and platforms. In order to address these issues and make it easier for scientists to conduct scientific analyses in bioinformatics, we customized VizSciFlow framework by integrating services and tools from this domain. The availability of numerous bioinformatics tools and pipelines (including quite complex pipelines) provides many examples to test the comprehensiveness and expressiveness of the syntax and constructs of our DSL to model scientific workflows. Moreover, by customizing our framework for computational biology, we aim to address the usability and flexibility problems of existing bioinformatics systems, such as Galaxy, QIIME and Mothur (cf. Section 4).

Galaxy is a web-based system and QIIME and Mothur are desktop-based command-line applications. Desktop applications have the added complexity of requiring the systems to be installed and maintained locally, often by the bioinformaticians. Although Galaxy is a web-based platform, in our user study, we discovered that scientists found it difficult to compose and execute complex scientific tasks using this platform (details can be found in the evaluation Section 7). Moreover, while most SWfMSs support a single execution platform, VizSciFlow accommodates multiple runtime environments - local server, Galaxy server, and Hadoop cloud cluster. A single workflow can be composed of various tasks executing in different execution environments. At the same time, our system also provides three different data storage mechanisms - POSIX, Galaxy datasets, and HDFS.

There are numerous challenges when engineering an interactive environment for managing scientific workflows, including how do we provide scientists the ability to specify workflow both graphically and textually; how do we provide scientists the ability to readily distribute computation from within an interactive environment; and how do we design and build interactive features that enable a scientist to easily extend and adapt a toolset without further software development. Current SWfMS user interfaces do not support these features.

We present the contributions of our work in the form of the following six research questions.

(1) **RQ1**: Can we create a generic SWfMS framework for complex scientific data analysis which can be customized/adapted to different scientific domains?
   - We investigate and identify the challenges and requirements for interactive workflow modeling in a scientific workflow management system.
   - We propose a generic framework for complex scientific data analysis which can be adapted to multiple scientific research domains focusing on ease of interaction for the scientists.
   - As a proof of concept of our proposed framework, we develop VizSciFlow – a scientific workflow management system which supports composition, execution, collaboration and management of scientific workflows.

(2) **RQ2**: Can we combine visual and textual elements to facilitate the composition of workflows for domain scientists?
   - We exploit the benefits of visual elements along with textual scripting in the proposed framework. It provides a domain-specific interactive graphical environment equipped with secondary notations, development tools, and other visual entities to support the functionalities of a scientific workflow management system. Various workflow entities like data sources, computational modules and services, workflows, job histories, data filters

are represented as interactive visual elements which scientists can drag-and-drop on the editor. The visual elements are then transformed into concise code snippets.

(3) **RQ3**: Can we adapt VizSciFlow for a specific scientific domain such as bioinformatics?
- In order to show that the framework is adaptable to different scientific domains, we integrate tools and services from the bioinformatics domain into VizSciFlow to create a scientific workflow management system for computational biology and conducted case studies and user studies to show the effectiveness of VizSciFlow.

(4) **RQ4**: Does the combination of visual and textual elements help the scientists compose scientific workflows easily with less cognitive load?
- In order to show the effectiveness of the use of visual and textual elements in the proposed framework, we conducted both case studies and user studies. In particular, we adapted VizSciFlow for bioinformatics domain as a case study and then conducted two user studies.
- We use this system to evaluate our proposed framework and to compare with a widely used scientific workflow management system, Galaxy and popular programming language, Python in terms of their usability, efficiency and expressiveness. The results of our preliminary studies show that VizSciFlow is a better choice than them.
- To further evaluate the usability of VizSciFlow, we conducted a user study to measure the subjective perceptions of users on the usability of the system using the System Usability Scale (SUS) post-test survey instrument [16, 17]. The use of graphical and textual elements gives a unique novelty in composing complex workflows. First of all, the graphical elements provide easy drag-and-drop of useful workflow entities. Second, the textual scripts provide concise and understandable workflows not only for ease of use but also for maintenance. The participants of the user studies worked with most of the interactive elements of the system to evaluate our framework.

(5) **RQ5**: How flexible is the proposed system in incorporating new services or new tools for a specific domain?
- A user study was conducted to evaluate the flexibility of integrating third party tools into VizSciFlow. The NASA-TLX [49] post-task questionnaire is used to collect the opinions of the participants. This research question shows that the proposed framework makes it easier for users to interact with the system in integrating new analysis tools and computational modules using a graphical window which offers a Python code editor and JSON mapper.

(6) **RQ6**: Does the proposed system facilitate distributed computational environments?
- A case study was conducted to measure the performance of the system in a large data volume and high performance scenario. We evaluate if VizSciFlow can successfully forward data and service calls to a distributed infrastructure. A workflow of our proposed system can consist of different services, some of which may run in local server, some in Galaxy and some in distributed computing environments. The scientists can specify the services, right from the framework, to make use of the hybrid computational environment as necessary.

The remainder of the document is organized as follows. Section 2 provides background information on SWfMSs and workflow modeling languages. The research challenges of scientific workflow modeling are described in Section 3. In Section 4 we provide an overview of the related work in this area. The functional architecture of our proposed framework for workflow specification, execution and management is presented in Section 5. A proof of concept implementation of our proposed workflow, VizSciFlow, is described in Section 6. The evaluation and experimentation to understand the usability, efficiency, expressiveness, flexibility and the support of distributed environments of our proposed workflow modeling framework are discussed in Section 7. The

results of the evaluation are discussed in Section 8. Potential threats to validity are described in Section 9. We conclude the paper and explain the future directions of our work in Section 10.

2 BACKGROUND

This section provides background to our approach with a discussion of scientific workflow, scientific workflow management systems, and workflow management languages.

2.1 Scientific Workflow and Scientific Workflow Management Systems

Lin et al. [70] define scientific workflow to be "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." Scientific workflows can be used during different phases of a larger scientific process, i.e., the cycle of hypothesis formation, experiment design, execution, and data analysis [41, 73]. It composes a collection of interdependent tasks which acquire, generate, transform or analyze complex datasets [73]. The life-cycle of a scientific workflow can be divided into the following four phases [45, 71–73, 109]:

(1) Composition Phase: The definition of the abstract scientific workflow, the activities, and the data dependencies between activities are specified in this phase.
(2) Deployment Phase: The abstract workflow is translated into an executable concrete workflow (cf. Section 2.2). Operations such as validation, resource allocation, scheduling, optimization, parameter binding, and data staging are part of this phase.
(3) Execution Phase: Input data are sent to the computational module, the processing algorithm is executed on the data, and output data are generated. Monitoring, controlling and steering are performed and provenance data are collected during the execution of the workflow.
(4) Analysis Phase: Visualization of data and analysis of provenance data are part of this phase.

A scientific workflow management system (SWfMS) defines, modifies, manages, monitors, and executes scientific workflows by executing scientific tasks [70]. Its primary goal is to automate the execution of scientific workflows [72] and it may additionally automate the entire scientific workflow life-cycle.

2.2 Workflow Modeling Languages

Workflow languages need to support a user-driven, incremental, prototypical approach to workflow composition [11] for conducting scientific experiments. Four important constructs – *Sequence*, *Parallelism (AND-split and AND-join)*, *Choice (OR-split and OR-join)* and *Iteration* are used to create simple to complex workflows [112]. Smaller workflows are composed using these constructs to build a larger workflow. A workflow may take either an *Abstract* or a *Concrete* form [74]. Abstract workflows emphasize the analytical operations or functions to be performed rather than the mechanisms for performing the operations [119]. As a result, they can be targeted to different execution environments at run time. Scientists primarily work with abstract workflows. A concrete (or executable) workflow is a description of the actual executable modules and their relationships in order to perform the operations indicated in the abstract workflow.

Higher-level workflow models are based on either graphical or text-based representations. Graphical modeling tools like Petri nets [93], UML-based [32, 117], YAWL [22, 113], Triana PSE [24], and Ptolemy II [15] offer desktop or web-based interfaces to compose a workflow by dragging and dropping graphical elements and connecting them. These tools hide low-level details and help users to focus on higher-level abstractions. The proponents of visual programming claim that these languages provide cognitive benefits for program comprehension [14]. A disadvantage is that

graphical models become chaotic and confusing as the number of tasks and workflow complexity increase due to the entanglement and overlap of a large number of graphical elements and wires. It can be quite hard to follow the direction of data flow to capture the semantics. Green et al. [47] use *match-mismatch* conjecture to measure program comprehension [42] only to conclude that there is no evidence that a visual language is better than a text-based language in all application contexts. The choice of visual or textual language depends on various factors such as size and complexity of the program, control or data flow, forward or reverse flow of data or control, the expertise level of the programmers.

Text-based (or language-based) modeling is the representation of workflow models in the form of text. Some researchers suggest that a textual program is easy to scan [46]. One type of text-based modeling expresses abstract workflow models as configuration or serialization formats in a markup language, such as XML (e.g., XPDL [23], WS-BPEL 2.0 [5], and Pegasus DAX [30]), YAML (e.g., CWL [6]) and JSON (e.g., Galaxy [2]). These workflow specifications are easy to reconfigure when the data source or execution environment changes, but often need excessive text to meaningfully define a workflow model. Iterative and parallel operations are not easy to define and the execution semantics is often hard to infer from the configuration text. Although Petri nets [93] have a formal mathematical notation in addition to its graphical notation, it is still hard to decipher for non-mathematical users.

Scripting languages, general purpose programming languages (GPLs), and DSLs are also used to create textual workflow models. Configuration As Code (CaC), an approach to store configuration [89, 95] of workflow composition as source code, provides dynamic, maintainable, versionable, testable and collaborative workflow code. Scripting can be considered as the most basic form of text-based modeling. It is not robust due to a lack of dependencies and reentrancy [64]. Dependencies specify the upstream data or tasks that downstream services require as input while reentrancy refers to the resume-ability of a workflow execution from a check point. General purpose programming languages like C/C++, Java or Python can define the most powerful and robust workflow models. But scientists usually lack deep understanding of the complex constructs of these languages. As well, many of these languages have a lot of boilerplate code or need overly complicated code for accessing distributed storage and processing which is beyond the realm of scientists' programming abilities.

A textual domain-specific language (DSL) [37] is designed to hide the complexity of a system and bridge the gap between the problem domain and the solution domain [115]. A DSL supports a minimal set of language features and limited vocabulary specific to a scientist's domain knowledge to increase productivity and decrease maintenance effort [54]. DSLs offer substantial gains in expressiveness and ease of use compared with general-purpose programming languages in their domain of application [60, 82]. Some examples of prominent DSLs are YACC, SQL, HTML [12]. Current Workflow Language (CWL) [6], Nextflow [31], and SnakeMake [61] are some DSLs for workflow composition in the scientific domain.

A DSL is developed in several phases – decision, analysis, design, implementation, and deployment [82]. Domain analysis or domain modeling is the process of extracting the conceptual model of the problem domain (essential concepts, domain knowledge, techniques, and paradigms) and generating tools and artifacts for the solution domain (technical space, middleware, platforms and programming languages) [10]. A domain model generally uses the same terminologies as the problem domain so that it can be used to communicate with domain-users, domain-experts and non-technical stakeholders.

From a software development perspective, there are two kinds of DSLs – internal (or embedded) and external [38]. An internal DSL is implemented on top of a host language and uses the infrastructure of the host [39]. Programs written with an internal DSL look similar to a host language

program. External DSLs are completely separate languages that are parsed into a form that the host language can understand [37]. Developing internal DSLs requires less effort and has the benefit of being able to draw from all of the features of the host language. On the other hand, an external DSL can detect errors early and error messages can be clear and domain-specific. In a collaborative development, domain-users will be restricted from using complex syntax of the host language. Otherwise, complex syntax may confuse other collaborators who might have less expertise in that host language. As well, it is possible to statically check the programs with an external DSL compiler. For our proposed framework, we opted for the external DSL.

3 RESEARCH CHALLENGES

In order to design a framework for complex scientific data analysis, we first identify the key challenges and requirements. We studied the existing literature for DSLs [12, 37, 60, 82], interactive workflow workbenches [4, 78] and bioinformatics domains [96]. We also experimented with existing SWfMSs such as Galaxy [40], Kepler [4], Pegasus [30], Taverna [118], Workspace [25] and Bpipe [98] which gave us good insights into the problem domain. We discussed with scientists their problems and requirements for workflow modeling and management in their respective domains. To gain further knowledge, we consulted several workflow system reference architectures [51, 71], and in particular the approach outlined in our earlier study [97]. Furthermore, recently we conducted extensive experiments on designing collaborative Groupware Systems [84], proposed a micro-level data management scheme [21], studied provenance Modeling for workflows [35], experimented an attribute level locking scheme [83] and even proposed a novel mechanism for storing workflow outputs optimally [20] to Support Complex Scientific Data Analysis, which also gave us insights on the research challenges in this domain. In this section we describe the challenges and requirements we identified for developing a DSL-driven SWfMS framework.

3.1 Gathering Essential Knowledge for Domain Modeling

The automated scientific problems are usually implemented in a general purpose programming language like C, C++, Java or Python. Workflow languages orchestrate many of such solutions in a loosely-coupled manner. Developers are naturally inclined to use a general purpose programming language. Unfortunately, such solutions are often complex and too cumbersome to grasp easily for domain scientists. In a large distributed system, there are inherent complexities of connecting to servers, binding parameters, calling service methods, and comprehending return patterns. Domain modeling emphasizes abstracting the complexity of the system and extracting the most useful features for the domain-users. Developers need to work with domain experts, consult domain-specific standards, ontologies and best practices, use existing similar systems to determine the proper level of abstraction, and to identify the appropriate vocabulary in terms of the target domain.

3.2 Supporting Visual Guidance

Non-expert users are interested in graphical languages as there is less need to understand programming. For example, graphical languages intrinsically capture parallelism without explicit parallelism control structures [76]. Expert users can make rapid prototyping and compact representation of potentially complicated workflows using textual languages [76]. Visual elements of a graphical interface can complement textual modeling to increase the interest of non-expert users in textual languages. In a visually guided workflow modeling environment, developers must provide interactive graphical elements to represent workflow and composition artifacts. The development life-cycle of a scientific workflow has similar steps to using general-purpose programming languages – planning, designing, composing, testing and debugging [76]. The IDE for textual workflow modeling should provide support for the entire life-cycle. The IDE needs to provide visual elements

that can be added to the workflow editor using mouse or keyboard interaction (e.g., drag-and-drop). The developers of such UI environments need to determine the visual representations to use and to determine how they are then transformed into code when inserted into the editor.

In most graphical modeling frameworks for data-intensive scientific analysis, users create a Data-Flow Graph (DFG) by dragging visual elements into the graph to form a chain of services [4, 40]. A data-flow graph improves the cognition of the model. The developers need to generate a compact data-flow graph to fit in a manageable layout with less cluttering of wires. The DAG-based Galaxy workflow system packs a lot of information in a graph but unfortunately, it results in a complex layout even for a relatively modest workflow.

3.3 Deriving Minimal DSL Constructs

The syntactic constructs to glue together various domain model elements become the syntax of the DSL [39]. General purpose programming languages provide many constructs each with multiple variants to address many possible application types. The constructs used in workflows are mostly specified to define rulesets and to compose services effectively. The main challenges of developing a workflow DSL is to derive a minimal set of constructs with precise syntax that can express the workflow semantics in a clear and concise way. The keywords for the syntax to specify the DSL constructs should be as few as possible.

We require a DSL with minimal constructs for specifying scientific workflow, but that can be customized for different scientific domains. The vocabulary of the DSL should be easily expandable by integrating domain-specific services. If a DSL is required to span multiple domains, developers should be able to integrate services from other domains and enhance the DSL vocabulary accordingly.

3.4 Supporting Extensibility and Balancing Flexibility and Usability

While designing any information system, it is important to balance flexibility and usability [50, 91, 102]. A system is expected to support common activities effectively and efficiently as well as be flexible enough to perform whatever the users want from it. Due to conflicting requirements, it is not always possible to provide both with the same level of effectiveness and a trade-off is often inevitable which is widely known as the flexibility-usability trade-off [69]. It is a challenge to determine the threshold of this trade-off to adequately fulfill both flexibility and usability. An extensibility model is necessary for experienced users to exploit the system capabilities and extend the system to improve flexibility.

General purpose programming languages support extensibility, but at the cost of usability. DSLs provide a higher level of abstraction but too much abstraction may hinder the system's flexibility. The internal system is much more powerful than the abstracted domain model. Experienced domain-users need another level of abstraction to exploit this power.

3.5 Allowing Sharing and Collaboration

Workflow description and execution capabilities offer a new way of sharing and managing information [41]. As the complexity of a scientific experiment increases, it becomes difficult for a single user to author and test a large workflow. It is even harder if knowledge and resources of multiple domains are needed in the same workflow. Sharing and collaboration of services and workflows support reuse of scientific computational modules [43, 84] and help to stop reinventing the wheel. A further challenge of workflow sharing is to ensure that there exists appropriate authorization for a particular user to access the resources which the workflows and services use.

3.6 Large Storage Management and High Processing Performance

Data-intensive scientific domains use heterogeneous data types, often in a massive volume. Data for computational biology or genetics, for example, often requires many terabytes (TB) or even petabytes (PB) of disk space. Traditional IT infrastructures are limited in handling such volumes of data. New technologies have emerged for managing large datasets that are often referred to as *Big Data*. At the same time, many scientific algorithms are compute-intensive and need enormous processing power to execute, especially when large-scale data is involved. For instance, powerful computing resources are essential for processing genomic data with an acceptable response time. Next Generation Sequencing uses massively parallel methods, where thousands or even millions of reactions occur simultaneously. Considerable complex boilerplate code is usually needed for general-purpose programming languages to support high processing capabilities during workflow execution. Developers face challenges to manipulate huge datasets as well as to forward service execution to high processing clusters transparently. A carefully designed workflow system should support large storage and high performance such that domain-users do not need any extra constructs to use high performance services.

4 RELATED WORK

There are various graphical workbenches for workflow modeling, such as Galaxy, iPlant [40], Workspace [25], QIIME 2 Studio [19, 94], and Pegasus [30], which offer visual editors for composing workflows. The user interface of Galaxy and iPlant are web based while Workspace, QIIME 2 Studio and Pegasus provide desktop interfaces for workflow modeling which need to be installed on the user's machine. Galaxy and Pegasus have JSON and XML based serialization formats respectively, and thus have similar pitfalls as the text-based modeling languages stated in Section 2. Our proposed framework provides an interactive graphical web interface to compose workflow models in DSL and to submit them for execution. Since our approach follows the service calls, it can express the semantics of the natural flow of pipeline execution. While most SWfMSs support a single execution platform, VizSciFlow accommodates a hybrid runtime environment consisting of local server, Galaxy server and Hadoop cluster on a cloud. Users can explicitly choose an execution environment for a task as they model the workflow. A single workflow can be composed of various tasks executing in different execution environments. At the same time, our system also provides three different data storage mechanisms - POSIX, Galaxy datasets and HDFS. Moreover, the VizSciFlow system exposes its functionalities as RESTful web services. Network enabled client programs, such as *curl* and *wget*, can communicate with the system using standard HTTP requests and exchanging JSON-based messages. Furthermore, the system can generate a data-flow graph of the service composition to support workflow comprehension.

Nextflow [31] specifies composition of parallel and reactive workflows. SnakeMake [61] and Bpipe [98] define rules for composing bioinformatics tools as a pipeline. Current Workflow Language (CWL) [6] uses YAML and JSON for workflow composition. None of these DSLs provide a dedicated interactive user interface for rapid prototyping. These languages only define language constructs for workflow composition and do not provide services for any specific scientific domain. The domain-users are responsible for installing the tools themselves before chaining them as a workflow using these languages. Some users find CWL code excessively verbose and clumsy [33]. Universal configuration standards like CWL have benefits for creating multi-domain and cross-platform workflows. But efficiency is not their strength.

Text-based domain-specific languages can be seen in many other domains such as User Interface Management Systems (UIMSs) [28, 36, 111], Model Driven Engineering (MDE) [56, 107] and Language Oriented Programming. There are also various IDEs to develop textual DSLs, for example,

Xtext [34], Eclipse Epsilon [59], ANTLR IDE [90], DSL Forge [62], and Microsoft DSL Tools [27]. Investigating the benefits of these efforts gave us motivation to identify the requirements of a DSL and a domain-specific interactive graphical environment for workflow modeling for the scientific data analysis domain.

Table 1 demonstrates a comparison of our proposed framework with some graphical workflow management systems in the scientific domain. From a UI engineering perspective, all the features listed in the table may not be interesting, but it is important to offer domain-users all the required features along with the interactive elements to model workflow seamlessly. The REST interface supports sending a DSL script to the VizSciFlow server for execution and for retrieving the results using *curl* or *wget*. This operation acts like a network-enabled command line interface. Galaxy provides an API for a similar operation, but domain-users must know Python programming and the client machine must have Python installed to use it. Domain-users need very little or no prior programming knowledge to write a modest workflow in our proposed DSL. They will either quickly learn the limited language constructs and control statements that it offers or use the *autocomplete* feature of the editor to insert the language constructs easily. No other framework in the list supports large data storage and high performance processing as VizSciFlow do which is important for today's data-intensive scientific analysis. Using Big Data support of our proposed framework, selected service requests are forwarded to a *Hadoop* cluster for execution and results are collected.

Table 1. VizSciFlow vs Existing Workflow Modeling Systems

Workflow System	UI	Services	Big Data	DFG	Text	Workflow Modeling Language	Little or No Programming
Galaxy	Web API	✓	✗	Implicit	JSON	Visual	✓
Kepler	Desktop	✗	✗	Implicit	XML KML	Visual	✓
Pegasus	CLI	✗	✗	Implicit	XML	Textual (GPL)	✗
Taverna	Desktop CLI	✗	✗	Implicit	RDF/XML	Visual	✓
Workspace	Desktop	✗	✗	Implicit	XML	Visual	✓
VizSciFlow	Web REST	✓	✓	✓	DSL	Textual (DSL)	✓

DFG = Data-Flow Graph; **Services** = Domain-specific services preinstalled; **CLI** = Command Line Interface; **Big Data** = Distributed storage and processing support; **Text** = Serialization format

5 A FRAMEWORK FOR SCIENTIFIC DATA ANALYSIS

In this section, we describe the architecture of our proposed framework to build a generic scientific workflow management system using both graphical and textual elements.

5.1 Architecture

We follow the recommendations of functional architecture of workflow management systems [51, 52, 71, 97] to derive the architecture of our proposed framework. The architecture consists of four layers – presentation, domain, execution and infrastructure – which implement the phases of a workflow life-cycle as depicted in Figure 1. JSON-based messaging protocol is used for communication between layers. This framework is in general applicable to any scientific domain.

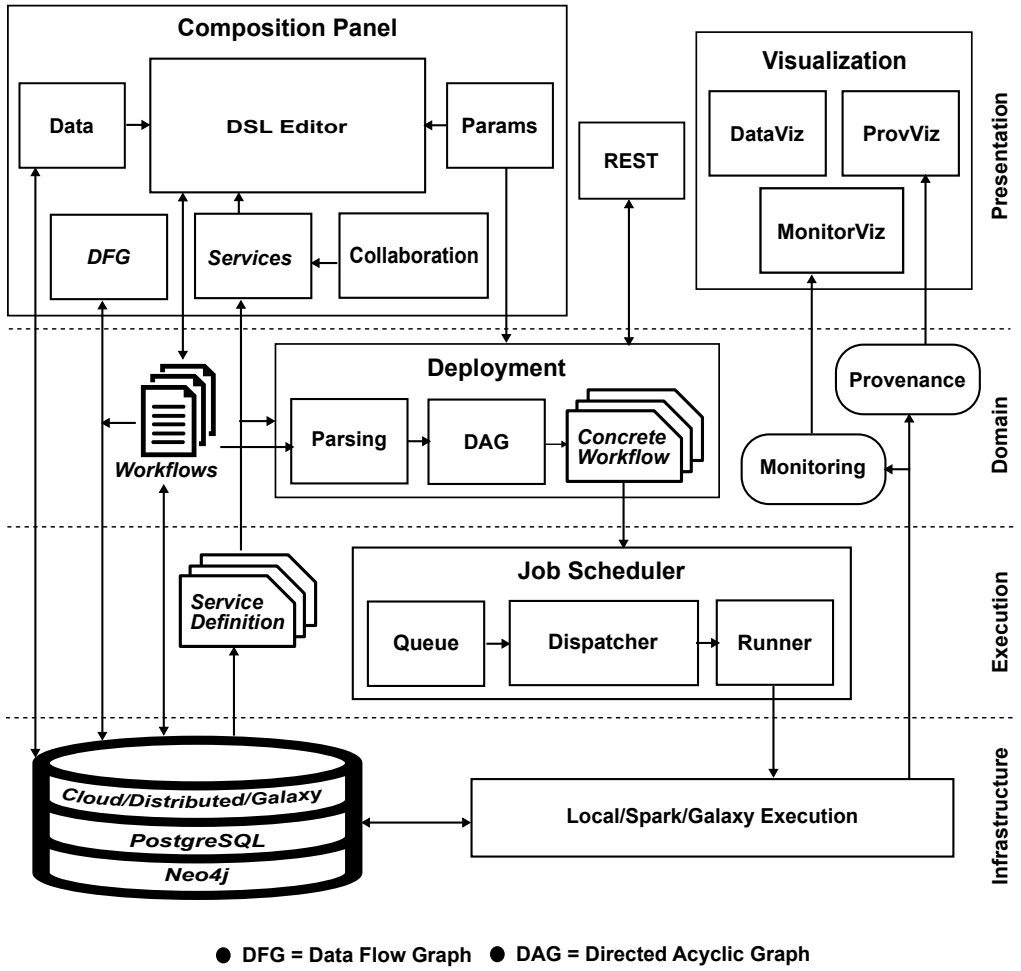


Fig. 1. Functional Architecture of a SWfMS with Visually Guided Scripting Framework

The *Presentation Layer* consists of a workflow composition panel, REST interface, visualizers for data, monitoring and provenance as well as graphical elements for sharing and management of the workflow system. The composition panel, which is a web interface, supports many features of a typical Integrated Development Environment (IDE). It provides interactive graphical elements to help make the process of writing the workflow scripts intuitive, fluent and comfortable. This layer is comprised of the following components:

- **DSL Editor**: The DSL editor is built using the powerful ACE [1] online source code editor. ACE supports familiar secondary notations and development tool features like syntax highlighting (110 languages), automatic code completion, automatic indentation, quick info, and parameter

info for many popular languages. We extend and adjust the ACE code to support domain-specific keywords and constructs. For example, domain-users can type ***for*** and press Ctrl + Space to insert a template of a code snippet for the Iteration construct.

- **Data Source Panel**: The heterogeneous data sources are shown in a hierarchical layout. It can currently show the data tree of POSIX, Galaxy and HDFS file systems. An authorization mechanism manages the access privileges of a user on a particular dataset. The data from this layout can be dragged over to the code editor as code snippets. A strong search-and-filter feature powered by regular expression is provided to find data in many possible ways – name, size, date, metadata, etc. The search history is preserved and can be saved with proper names for later use. The saved search-and-filter query can be dragged over to the code editor and inserted as a code snippet for execution-time querying of data.
- **Service Panel**: The loosely-coupled computational modules for data analysis are listed in an interactive grid panel. Domain-users can search for a service, observe its features and usage syntax, configure it, and drag it to the code editor as a code snippet. An authorization mechanism manages the access privilege of a service – public, shared or private. A new service can be added to the system using the plug-in architecture, described below.
- **Workflow Panel**: It manages all the workflows accessible by the currently logged-on user. A workflow can have public, shared or private access.
- **Commands Panel**: Domain-users use this panel to submit the script for execution. Workflows are usually submitted to the job scheduler for asynchronous processing. But for quick testing of tools or small and short-running workflows, domain-users can select *Immediate* for synchronous execution and immediately view the results.
- **Job History Panel**: It manages the execution histories of all the workflows run by the currently logged-on user. Domain-users can stop and restart a job.
- **Logging and Reporting Panel**: This panel shows the compiler error, execution error, user-defined logs and system generated logs.
- **Monitoring Panel**: The monitoring panel displays the status of each service execution, errors, output, running time etc. of an executed or currently executing workflow. Users can select the job history of a workflow execution from the Job History Panel. They can also decide to stop the execution and restart later.

The visualization mechanism currently uses the capabilities of the browsers to display data. The output and provenance data are either visualized or downloaded based on browser support for the data. Custom visualization of data will be gradually implemented. As stated below, our plugin architecture as well as sharing and collaboration also provide visual interfaces to ease the corresponding operations.

The *Domain layer* parses a workflow along with its parameters and generates an optimized Abstract Syntax Tree (AST). The AST is then transformed into a concrete workflow by attaching the service modules from a library manager, which keeps a catalog of available service modules in the system and forwards calls to an appropriate service provider during execution. The library manager holds the information about the inputs and outputs of a service and is able to automatically connect the outputs of a service to the inputs of the next service.

The *Execution Layer* receives the concrete workflows and queues them in a *job scheduler*. It uses the PersistentScheduler of Celery Beat[1] to dispatch the workflows for execution. The PersistentScheduler is a distributed task queue for asynchronous task scheduling which uses the

[1] https://docs.celeryproject.org/en/latest/userguide/periodic-tasks.html/

Publish-Subscribe messaging pattern to perform unattended operations using Redis[2] message brokering server.

The *Infrastructure Layer* is responsible for managing physical resources for workflow storage and execution. Individual services can run in different runtime environments, for example, a local cluster, a Galaxy server or a cloud-deployed Hadoop cluster. More details about the infrastructure layer are given in Subsection 5.5 and Subsection 5.6.

The developer uses this framework to build a domain-specific SWfMS which provides an intuitive graphical environment to help model workflows in DSL easily. The GUI of the developed system supports interactive visual elements to extend the system with a plugin architecture, share and collaborate services and workflows, manipulate large datasets of distributed storage, and delegate execution to high performance distributed processing. The following subsections describe how we addressed the other research challenges, which were listed in Section 3, in this framework.

5.2 Plugin Architecture

The proposed framework uses a service-oriented approach to extend its tool support by consuming external services and computational modules. There are several ways to extend the capabilities of VizSciFlow such as developing a new algorithm in Python, making an existing service call more user-friendly, wrapping a library programmed in GPL, writing a full-featured plug-in. The new service is added to the DSL as an enriched vocabulary. A basic set of services is pre-installed in the system by default on the basis of popularity among the domain scientists. The plugin architecture of our proposed framework is shown in Figure 2.

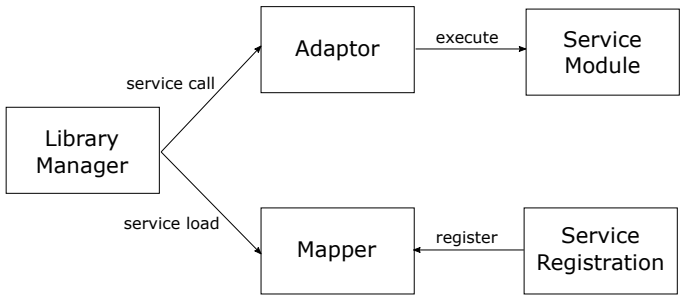


Fig. 2. Plugin Architecture of VizSciFlow

[2] https://redis.io/

```
{
   "package": "fastqc",
   "module": "app.biowl.libraries.fastqc.adaptor",
   "name": "CheckQuality",
   "internal": "run_fastqc",
   "desc": "Measures quality of a FASTQ file",
   "params": [
      {
         "name": "data",
         "type": "file"
      }
   ],
   "returns": [
      {
         "name": "html",
         "type": "file"
      },
      {
         "name": "zip",
         "type": "file"
      }
   ],
   "example": "html, zip = fastqc.CheckQuality(data)"
}
```

Listing 1. VizSciFlow Library Mapping

A JSON mapper associates the services provided by tools or modules with the DSL vocabulary. This mapping mechanism, guided by a dialog in the user interface, simplifies the integration of a tool or custom Python module into the system. The dialog provides a Python code editor and a JSON mapper. It also allows domain-users to specify Python packages from PyPI[3] or other indexes for pip[4] installation as well as to upload other packages to the system. In Listing 1, an example mapping of the *FastQC* tool to DSL is shown. FastQC is a Java program used in bioinformatics to check the quality of a genome sequence file. The mapper adds a new service *CheckQuality* to the DSL vocabulary and internally binds it to an adaptor method named *run_fastqc* which checks the arguments and forwards the call to the FastQC tool installed in the system. An adaptor class, associated with each service, is responsible for packing the arguments, running the tool with those arguments and fetching the results. The LibraryManager manages all adaptors and mappers.

5.3 Sharing and Collaboration

Services which are integrated by domain-users into the system using the plugin capabilities of the framework, can be shared with other users. A visual interface is offered to publish a service to all users or to share with specific users. Likewise workflows written by the domain-users can be shared and modified collaboratively with other users using another visual interface. The workflow descriptions and the data sources used in the workflows are made accessible with read, write or read-write permissions to the target users. For the VizSciFlow implementation, we store the workflows in a PostgreSQL database as a JSON datatype. In the future, we will store the workflows in a Git repository using GitPython[5] to support easier collaborative creation.

[3] https://pypi.org/
[4] https://pypi.org/project/pip/
[5] https://pypi.org/project/GitPython/

5.4 DSL

As described in Section 2, domain modeling maps a scientist's domain to a solution domain. A DSL provides an abstracted view of the solution domain to domain-users. Developers work with domain-experts, consult domain-specific standards and ontologies [88, 108], and use existing workflow systems in the domain [3, 44, 78, 103] to pick the tools for integration into the system and consequently derive domain-specific vocabulary terms for the DSL. The syntax of the DSL glues these terms together to create a workflow model.

Our proposed DSL was developed in two phases. In the first phase, the syntax for composition rules of the workflow pipeline was defined using Backus–Naur Form (*BNF*) [80], a notation for specifying context-free grammars. Some basic constructs from the Python language are borrowed for specifying keywords and control structures of the DSL. A *parameter sweeping* feature is added to allow for the execution of the same workflow repeatedly with different configurations. In the first phase, the DSL is not bound to any specific scientific domain. In the second phase, the DSL becomes bound to a scientific domain by adding domain-specific vocabulary terms, which are derived during the domain modeling. Most of the vocabulary terms correspond to computational modules or services. Using a plugin architecture, services can also be integrated dynamically which enhances the DSL vocabulary accordingly. Services from other domains can also be integrated. This dynamic service integration enables the DSL to incrementally evolve. Due to the advantage of external DSL, as explained in Section 2, we opted for it.

5.4.1 Keywords. The keyword set for the DSL is kept to a minimum to ensure a clear and concise script. These keywords provide the expressiveness necessary for specifying a Directed Acyclic Graph (DAG) style workflow model, which is common for data-intensive scientific domains [71]. Table 2 lists the keywords of the DSL.

Table 2. Keywords of proposed DSL

Name	Description
if ... else	Choice
for	Iteration
parallel ... with, lock	Parallelization
task	Subworkflow Definition
return	Exit Task

5.4.2 Operators. The proposed DSL defines a subset of arithmetic, logical and relational operators. Several operators are overloaded with special meanings for data and workflow types as indicated in Table 3.

5.4.3 Workflow Constructs. Our DSL supports four important pipeline constructs – *Sequence*, *Choice*, *Parallelization*, and *Iteration*. They can be chained together in a hierarchical fashion to form a complex workflow. A scientific workflow is often represented as a DAG or simply as a pipeline [71]. These constructs can define a DAG or a Directed Cyclic Graph (DCG) of services. An example workflow using these constructs is shown in Listing 2.

Table 3. Operators of proposed DSL

Type	Operator	Meaning
Arithmetic	+	Addition Union for Data Concatenation for String, List, Subworkflow
	-	Subtraction Difference for Data Removal for List, Subworkflow
	*	Multiplication
	/	Division
Relational	<	Less than
	>	Greater than
	<=	Less than or equal to
	>=	Greater than or equal to
	==	Equal to
	!=	Not equal to
Logical	and	True if both the operands are true
	or	True if either of the operands is true
	not	True if operand is false

```
# Iteration or Loop
for data in GetFiles('/public'):
    # Choice
    if GetType(data)=='fastq':#OR-split
        # Parallelization
        parallel: # AND-split
            CheckQuality(data)
        with:
            data=Align('Chr1.cdna',data)
            data=SamToBam(data)
        # AND-join
    else:
        print(data)
    # OR-join

    SortBam(data)
```

Listing 2. VizSciFlow Script Example

The DSL also supports nesting of function calls which makes sequential code even more concise as shown in Listing 3.

```
data=SamToBam(Align('Chr1.cdna', data))
```

Listing 3. VizSciFlow Function Composition Example

5.4.4 Modular Operations. The *task* keyword helps to construct a larger workflow from multiple smaller subworkflows. An *anonymous task* starts executing as soon as the interpreter finds the *task* keyword in the script. A *named task*, on the other hand, must be called from the downward code in the script. Additional properties can be attached to a *task* definition as parameters. Listing 4 defines a *task* for gene sequence analysis.

```
task AnalyzeSeq(data):

    CheckQuality(data)

    data = Align('Chr1.cdna', data)
    data = SamToBam(data)

data = AnalyzeSeq('R_001.fastq')
```

Listing 4. VizSciFlow Task Example

5.5 Middleware

The middleware of the system is primarily responsible for workflow management, scheduling and execution. It is based on the software component model of a loosely-coupled message-based structure [97]. The central component of our proposed middleware is implemented as a *Web Service*. There are many web frameworks like Flask[6] and Django[7] in Python, Spring in Java, ASP.NET in C# for the web service development. We used Flask for VizSciFlow. A distributed task queue is used for asynchronous task scheduling. The scheduler uses the *Publish-Subscribe* messaging pattern to perform asynchronous unattended operations using message brokering servers like Redis, Apache Kafka, JBoss Messaging.

The middleware layer also manages the multi-platform data storage and execution environment. It hides the complexity of the communication to externally-deployed remote systems like FTP, HTTP, Galaxy, Hadoop, etc. A generic interface is programmed to uniformly access local and remote files. The infrastructure layer of the proposed framework, as shown in Figure 3, is derived from the heterogeneous infrastructure model [97]. If remote data or services are referenced in the script, the system transparently connects to the corresponding external storage or processing units.

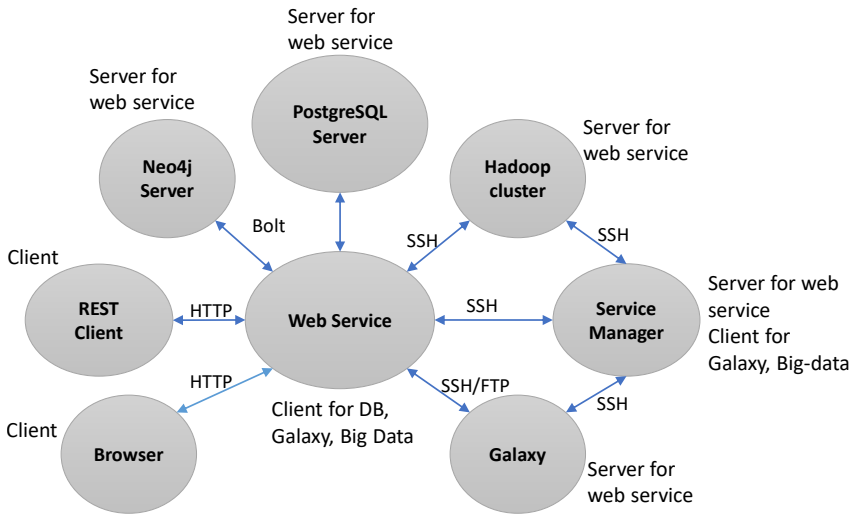


Fig. 3. Infrastructure Layer of Proposed Framework

[6] https://palletsprojects.com/p/flask/
[7] https://www.djangoproject.com/

5.6 Back End

The workflow details, provenance data, execution logs and user information are all stored in a relational database (i.e., PostgreSQL). An object-relational mapper (ORM) allows an object-oriented programming model to access the database. Graph database platforms like Neo4j[8] along with graph query languages like *Cypher* are used for generating, storing, and querying data-flow graphs of the scripts. The graph platform can be used for static analysis of scripts, optimization of data routing, and other graph analyses.

The system facilitates the storage of TB-scale data with the help of *Hadoop* [116]. The Hadoop Distributed File System (HDFS) is a distributed, scalable, and portable data storage mechanism designed to run on a cluster of inexpensive commodity hardware. A generic data source interface is needed for uniform and interoperable access to POSIX, HDFS, HTTP, FTP, and Galaxy data sources. The interface can also plug in new data sources into the system.

The high processing performance is provided by MapReduce [77], a distributed processing framework with fine-grained fault-tolerance, and Apache Spark [120], a lightning-fast cluster computing engine for large-scale data processing. A new paradigm for big data processing has recently emerged. *Apache Beam* provides a unified programming model to design and construct distributed data pipelines on a higher level of abstraction [57]. A single beam program can run on multiple runners such as MapReduce [29], Spark [121], Apache Flink [55], and Apache Samza [87].

6 VIZSCIFLOW: A PROOF OF CONCEPT OF OUR PROPOSED FRAMEWORK

In order to answer research questions **RQ1**, **RQ2** and **RQ3** regarding a generic scientific workflow management system using our proposed framework, we developed VizSciFlow as a proof of concept and adapted it to bioinformatics domain. It was developed with the Python 3.6 programming language which is very popular among data scientists for its rich support for scientific data analysis. External services, programmed in different languages including Python, can be plugged into the system using Python wrapper for command line access. The composition panel and visualization for the presentation layer are developed using standard web tools, including *HTML5*, *CSS3*, JavaScript, jQuery, and Bootstrap. The ACE [1] online source code editor is used to implement the interactive scripting editor. Domain-specific syntax highlighting, code completion, automatic indentation, quick info, parameter info, list services, logging, and error reporting are supported by the graphical interface. The data-flow graph is visualized using the *D3.js* library. A screenshot of the user interface during scripting is shown in Figure 4. The corresponding screenshot of data-flow graph visualization is depicted in Figure 5.

The middleware web service is written in *Flask*, a Python-based micro web framework. The DSL compiler uses PyParsing [81], an object-oriented recursive decent parser, to parse the script. Celery[9], a distributed task queue, is used for asynchronous task scheduling. Celery uses the *Publish-Subscribe* messaging pattern to communicate with Redis for long running operations. Additionally, workflows can be executed synchronously which returns results immediately, and is useful for simulation and quick testing.

VizSciFlow facilitates the storage of TB-scale data with the help of *Hadoop* [116]. Our multi-node HDFS cluster is hosted on the *Compute Canada*[10] cloud infrastructure. Another 3-node Hadoop cluster is deployed in our local infrastructure for simple evaluation purposes. POSIX, HDFS and Galaxy data sources interoperate over a generic data source interface which also makes plugging

[8] https://neo4j.com/
[9] http://www.celeryproject.org/
[10] https://www.computecanada.ca/

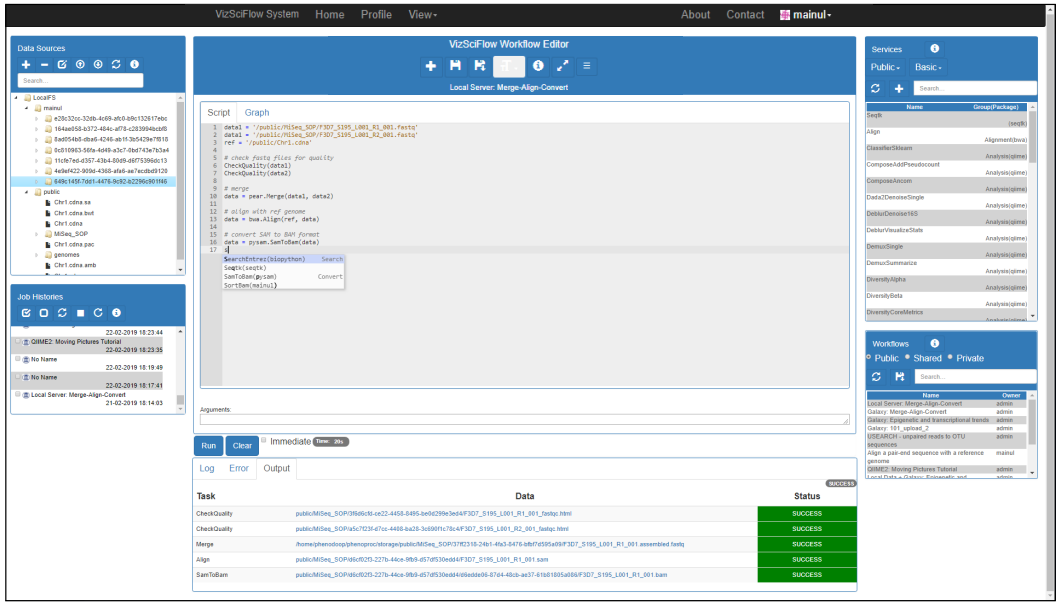


Fig. 4. VizSciFlow Script Interface

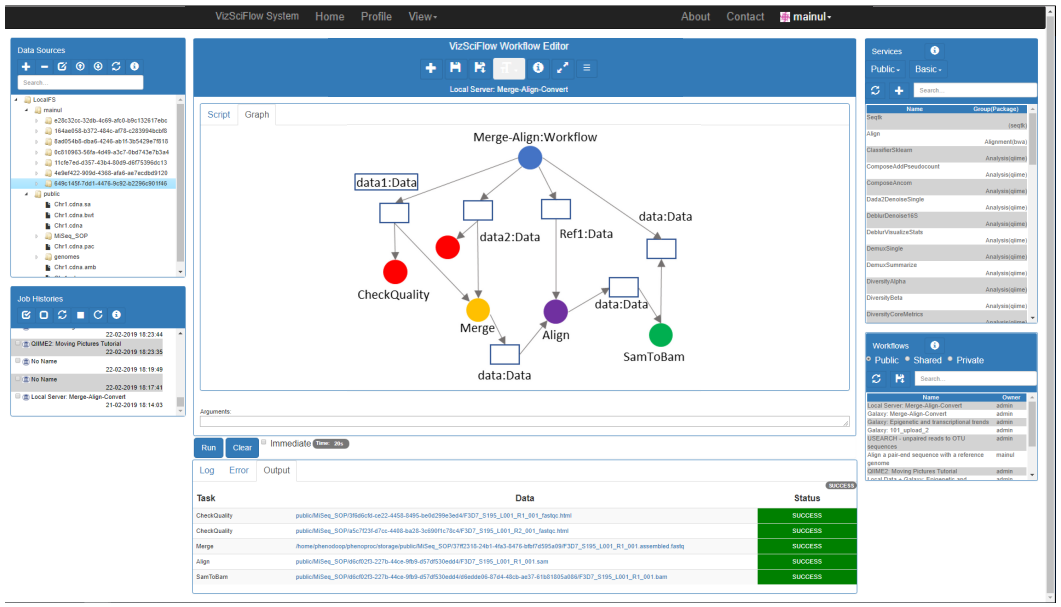


Fig. 5. Data-Flow Graph (DFG) of VizSciFlow

new data sources into the system easy. High performance distributed processing is supported by MapReduce [77] and Spark [120] programming models on a Hadoop YARN cluster [114].

Answer to RQ1

RQ1: Can we create a generic SWfMS framework for complex scientific data analysis which can be customized/adapted to different scientific domains?
Answer: VizSciFlow demonstrates that our proposed framework can be used to create an effective SWfMS. The system was developed using the Python programming language and the Flask web framework. The interpreted Python language supports plugging in new tools or modules dynamically and enhances the DSL vocabulary accordingly.

Answer to RQ2

RQ2: Can we combine visual and textual elements to facilitate the composition of workflows for domain scientists?
Answer: The code editor and interactive visual elements can be used to model workflows more intuitively. Graphical elements representing various workflow modeling entities like data sources, computational services, data filters, and service configuration can be used to compose workflow. The secondary textual notations like indenting, syntax highlighting, and comments can increase the readability of the script. Other visual elements can help with parameter sweeping, plugin integration, data manipulation, workflow execution and management, job scheduling and monitoring, sharing and collaboration etc.

VizSciFlow includes a basic set of services which can be used in any scientific domain. In order to answer research question **RQ3** of evaluating its capability in adapting to a new scientific domain, we conducted a case study by integrating tools and services from computational biology to derive a bioinformatics workflow management system. During domain modeling for this, we worked with biologists, consulted the Software Ontology (SWO) [79] of OBO Foundry [104], experimented with various bioinformatics tools [7, 19, 75, 94, 101, 122] and workflow management systems [4, 18, 30, 40, 63, 78, 118] to extract potential services for the system and subsequently derived the vocabulary of the DSL of the adapted system. This system encompasses tools for quality control of sequence data, alignment, normalization, quantitative analysis, differential expression and evaluation, trimming, error correction, QIIME 2™ [19] microbiome platform, OpenCV image processing, etc. Our system uses BioBlend [103] to integrate tools from Galaxy server and forward selected services to Galaxy server for execution.

Answer to RQ3

RQ3: Can we adapt VizSciFlow for a specific scientific domain such as bioinformatics?
Answer: The customization of VizSciFlow to derive bioinformatics workflow management system demonstrates that our proposed framework can be adapted to the scientific workflow management system for a particular scientific domain.

7 EVALUATION

In this section we describe the experiments and user studies that were conducted to evaluate our proposed workflow modeling framework and answer research questions **RQ4**, **RQ5** and **RQ6** on the evaluation of VizSciFlow.

In order to answer research questions **RQ4** and **RQ5**, three user studies were carried out with VizSciFlow. The evaluation procedure and results based on these user studies are explained below.

7.1 Evaluation of usability, efficiency and expressiveness in comparison to Galaxy and Python

A preliminary user study was carried out to compare the usability, efficiency, and expressiveness of our proposed workflow modeling framework with Galaxy and Python. We chose Galaxy because it is a popular workflow system in the bioinformatics domain with 168 enlisted servers. Python is not a workflow language, but it is a popular language for scientific analysis and is used by many bioinformaticians. When the required tools and libraries are installed, it is similar to other python-based textual packages or pipeline managers for bioinformatics like QIIME [19] or Biopython [26]. Since our DSL is an integral part of our graphical framework, we did not evaluate either separately but evaluated both together.

7.1.1 Participants. 10 graduate students took part in the user study – 8 male and 2 female. 9 were computer science students and 1 was a bioinformatics student. Ages included 6 between 20-30 and 4 between 31-40. All of the participants were experienced with Python programming and 4 worked previously with Galaxy while 6 have worked with other scientific workflow management systems. 3 of the participants were familiar with DSLs. The participants were given three workflow modeling tasks, each to be completed in VizSciFlow, Galaxy, and Python. The workflows were ordered with increasing complexity, larger data volume, and more diverse characteristics (e.g. iteration, modification). Table 4 lists the workflow tasks used for the user study.

7.1.2 Experiment Procedure. All the tools and services used in these tasks were pre-installed for Python and Galaxy. We first demonstrated the three systems to the participants with two example tasks. In those tasks, we described them how to run an external tool in Python using the *subprocess* module as well as how to create and execute a workflow in Galaxy. The participants' machines were connected to the Internet and they were advised to search for references in Internet if necessary. Each participant was allocated a 30-minute slot for each platform. We employed an observer and time tracker during the user study in order to maintain fairness in the evaluation process. The workflow modeling tasks were completed first in VizSciFlow. At this stage, participants had already gathered knowledge on the required tools, datasets for the tasks and the outputs. Next they performed the same tasks in Galaxy and then in Python. The participants were given a questionnaire with three questions and asked to give an assessment score between 1-10 for each quality attribute under consideration. The questions are listed in Table 5. They were allowed to give feedback for that task as well. Additionally, the system logs the interactions with data, services, workflows, reports and other resources in real-time.

7.1.3 Results. The system log shows that the users interacted with the visual elements very often while creating the models in VizSciFlow. They were most intrigued by the search facilities for data sources and services. Once the services and datasets were found, they were inserted into the code editor either by dragging or double clicking the item or selecting from a context menu. *Autocomplete* was also frequently used for inserting DSL constructs and services. The participants evaluated the workflow modeling system based on three quality attributes – usability, efficiency, and expressiveness.

All the participants were able to complete the assigned tasks in VizSciFlow successfully, but failed to do the same tasks in Galaxy and Python within the 30-minute time frame. They found that it is very easy to find what they need and express what they want to do in VizSciFlow. All of them could complete the modeling tasks correctly with relative ease. They asserted that VizSciFlow is user-friendly and easy to use. Most users could also complete Task 1 in Galaxy easily which is relatively simple. But as soon as the complexity rises in Task 2 and Task 3, they find it difficult to design and configure the workflow in Galaxy. One participant even commented, "Its user interface

Table 4. Workflow Tasks for User Study

Name	Type	Description
Example 1		Check Quality of a paired-end FASTQ file
Example 2		Check Quality of multiple paired-end FASTQ files
Task 1	Basic	Check Quality, Merge paired-end files to single-end file, Align with reference genome, Convert SAM to BAM of a paired-end FASTQ file
Task 2	Complex	Check Quality, Merge paired-end files to single-end file, Align with reference genome, Convert SAM to BAM of multiple paired-end FASTQ files
Task 3	Modification	Check Quality, Align forward and reverse files with reference genome, Convert SAM to BAM, merge BAM files of multiple paired-end FASTQ files

Table 5. Questionnaire for Usability, Efficiency, Expressiveness comparison

Quality	Question
Usability	How easily could you complete the task?
Efficiency	How fast could you complete the task?
Expressiveness	How simply could you express your task?

is fully incompatible for large size workflows". The users need to provide the parameters every time even if the workflow should execute with the same parameters. On the other hand, VizSciFlow can run workflow with fixed parameters. Almost all the users had a hard time completing any of the tasks in Python. The participants could find the appropriate services in VizSciFlow easily in contrast to both Galaxy and Python thanks to our naming mechanism which we chose based on the functionality of a tool rather than the tool name. For example, Burrows-Wheeler Aligner (BWA) [67] tool aligns sequences against a reference genome. The vocabulary of our system specifies this tool as *Align*, which a Genomicist can easily recognize. The participants made a perceived assessment of the system based on how easily they can author a workflow model in the given languages. The assessment scores are shown in Figure 6a.

Domain-users can write a modest workflow in VizSciFlow quickly without prior programming knowledge. The generation of code snippets by mouse interactions (drag-and-drop, mouse click, context menu) with graphical elements representing data sources, services, workflows, and reports make the scripting experience fluent and comfortable. The language constructs can be inserted into the code editor with autocomplete. One user commented, "Pretty user friendly interface and straightforward design". Another wrote, "The task is easy to execute and the editor is very helpful". Another participant said, "A little bit complicated UI at first. But it becomes easier with time" which was totally expected as text-based modeling appears more difficult than visual modeling at first. But as the size and complexity of the workflow increase and users become more experienced, they feel comfortable with textual modeling. Listing 5 and Figure 8 are two equivalent workflows in VizSciFlow and in Galaxy respectively for **Task 2**, listed in Table 4. Galaxy is easy to use for very

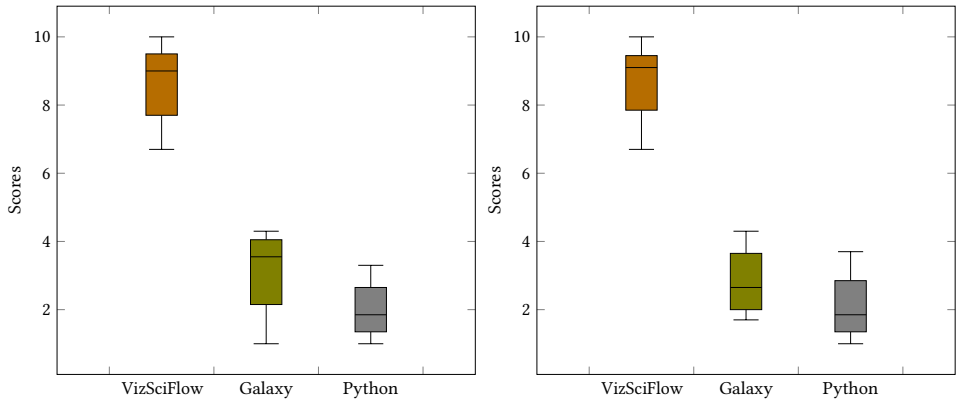


Fig. 6. Comparison of Quality Attributes

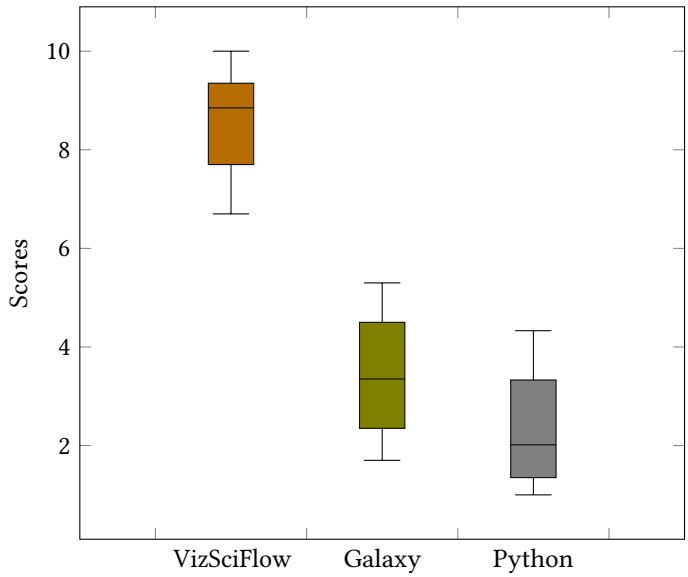


Fig. 7. Comparison of Expressiveness Attribute

small and simple workflows. The empty space in the editor is quickly filled up even with just a few tools if they have multiple parameters to bind. It is very difficult to follow the wires if a single dataset is connected to multiple tools. One participant commented, "Configuring selected tool as required is really hard to do" and another wrote, "Too much information to digest and too much learning curve". Even though all our participants were experienced in Python programming and all the required tools were pre-installed, they found the Python tasks very complex to model. They needed to consult the tool documentation frequently for parameter passing in the right format and for output interpretation. Some tools generated output to a file at a tool-specific location, some to the console, and some to a memory buffer. Some participants were even concerned when it was

observed that they needed the *subprocess* module to execute external tools. Another participant found it difficult to create a pipeline in Python and commented, "Hard to understand the sequence of the workflow. It is hard to detect and control the flow of data saving". VizSciFlow hides the complexity of tool execution and output interpretation. Users need only to call a function and pass parameters. The output and other provenance data are automatically tracked and made persistent.

```
task alignseqs(ref, data, data2):
    CheckQuality(data)
    CheckQuality(data2)

    data = Align(ref, data)
    data = SamToBam(data)

    data2 = Align(ref, data2)
    data2 = SamToBam(data2)

    return MergeBam(data, data2)

ds = ['Chr1.cdna', 'Albugo.cdna',
    'Bacillus.cdna']
fs = ['S1_R1.fastq','S2_R1.fastq',
     'S3_R1.fastq']
rs = ['S1_R2.fastq','S2_R2.fastq',
     'S3_R2.fastq']

for i in range(0, 3):
    alignseqs(ds[i], fs[i], rs[i])
```

Listing 5. VizSciFlow Sequence Alignment

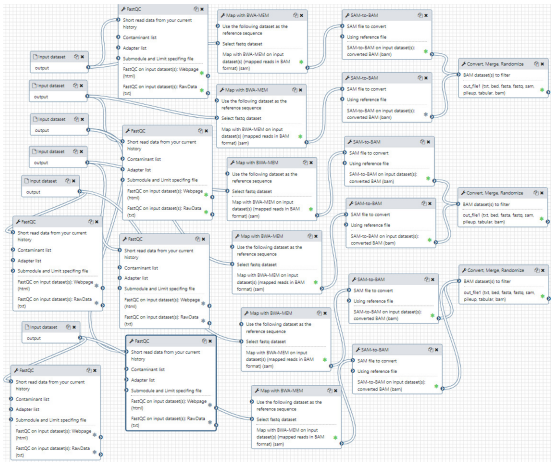


Fig. 8. Galaxy Sequence Alignment

All the participants could complete the workflows in VizSciFlow within the allotted 30 minute time limit. For Galaxy, most users were able to complete only one task within the time limit. For Python, none of the participants were able to complete a single task within the time limit. The participants assessed how fast they were able to create the workflow task correctly and gave assessment scores which are plotted in Figure 6b. The DSL code editor offers automatic code completion, context-sensitive help, incremental search for data sources and services, parameter info, logging and reporting mechanisms which help to create a workflow model efficiently. The participants could find the desired resources quickly. Although some participants were initially happy with the visual interface of Galaxy, they indicated later that figuring out how things work is time consuming and sometimes seems impossible. They also needed to spend considerable amount of time editing the workflow and managing the layout in Galaxy. On the other hand, almost all participants explicitly complained that tasks for Python were the most time consuming to model. Even experienced participants struggled to use the *subprocess* module correctly. They spent a lot of time frequently checking the API documentation of *subprocess* and external tools. This is not an absolute assessment of efficiency for VizSciFlow system since efficiency is best measured with objective assessment, not with the subjective assessment that we did. We employed an observer and time tracker who monitored how long a participant took to complete each task. Our intention is to compare if the system can model workflows faster than Galaxy and Python. We found that all participants could complete VizSciFlow tasks in the allocated 30-minute time slot while no one could complete the tasks in Galaxy or Python within this time.

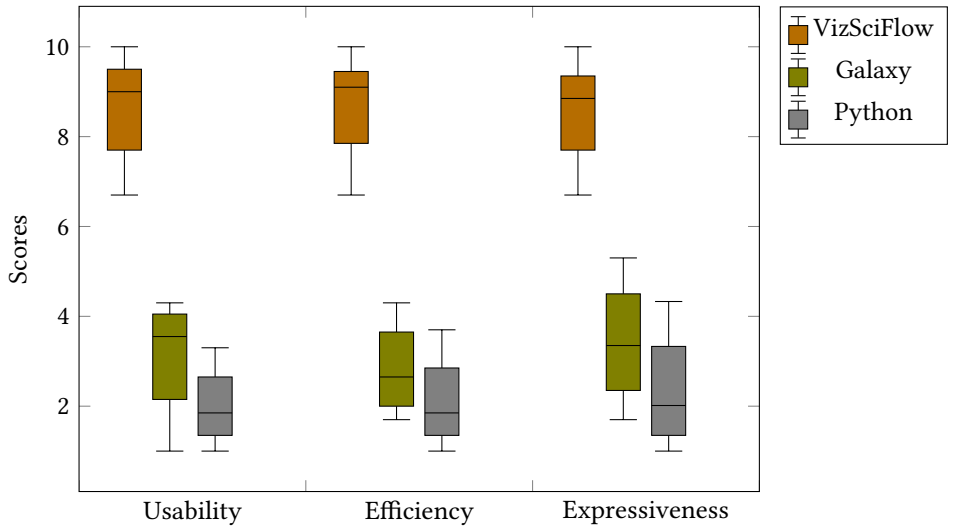


Fig. 9. Quality of VizSciFlow vs Galaxy vs Python

The participants evaluated how easily they could express the task in the chosen language. The assessment score is shown in Figure 7. All participants rate the tasks of VizSciFlow to be the most expressive among these three modeling languages. VizSciFlow users create a workflow using the supported language constructs and vocabulary. The DSL provides a minimal keyword set, precise syntax, and only four control statements to form a clear and concise workflow script. The participants used autocomplete to insert the control statements (Sequence, Choice, Iteration, Parallelization [112]) into the code editor. All the participants in the user study concluded that the Galaxy layout becomes overly complex and unmanageable even for modest workflows. One participant found the wiring of Galaxy confusing. As well, it is not possible from the tool element to distinguish between required and optional parameters. Some participants skipped assigning parameters, only to get an execution error later. The VizSciFlow service mapping indicates required and optional parameters. The vocabulary provides standard and meaningful names to the services to increase the expressiveness of the language for better readability and comprehensibility which generally leads to better workflow maintenance.

In order to compare the participants' scores for usability, efficiency and expressiveness quality attributes in VizSciFlow, Galaxy and Python, we plot the assessment scores in Figure 9. We can notice that the quality scores for VizSciFlow is significantly better than Galaxy and Python.

7.2 Evaluation of Usability

In order to study the usability of VizSciFlow further, we evaluated its user interface more rigorously using the System Usability Scale (SUS) [16, 17]. SUS is a post-test instrument, given to a participant after an entire usability testing session is over. It is one of the most widely used tools for assessing the perceived usability of a system or product. It has been shown to be robust even with relatively small numbers of participants (e.g., 8–10) [110]. SUS uses 10 Likert-type statements with responses based on a 5-point scale. The Likert formatting ranged from strongly disagree to strongly agree. The SUS score of a participant, in the range of possible values of 0 to 100, is calculated from his responses to the 10 questions. The final SUS score is derived by averaging the individual scores of the participants. The details of the SUS score calculation method can be found in [17].

7.2.1 Participants. 8 graduate students took part in this user study – 7 males, 1 female. 4 participants have used scientific workflow management systems before and 1 participant used domain-specific languages before. 5 of them were aged between 20-30 years, 2 between 31-40 years and 1 preferred not to divulge the age. 50% of the participants have worked with web-based SWfMSs before, but not with VizSciFlow.

7.2.2 Experiment Procedure. The participants were first given an introduction to the SUS survey instrument and then were asked to complete 5 tasks which required them to interact with most of the graphical elements in the system – code editor, data sources, services, workflows, job histories, monitoring, logs and so on. The operations spanned over the following tasks: i) create a new workflow model, ii) search and drag data to code editor, iii) search and drag services to code editor, iv) test a service by running it and checking logs and outputs, v) write DSL constructs and use editor supports, vi) modify an existing workflow, vii) check the executed and executing workflow instances, viii) execute and monitor a workflow, and ix) save and load a workflow. After finishing the tasks, the participants were asked to fill out the SUS questionnaire and their response scores are collected.

7.2.3 Results and Discussion. In Figure 10 the responses of the participants for each question are shown. The green bars at the upper and lower X-Axes indicate whether the question is positive when the score is high or low. The SUS score of each participant is calculated from the response scores. The average of the individual scores gives the final SUS score. Once the assessment was completed by all participants, we calculated the final SUS score to be *86.56*. As determined with over 5000 users across 500 different evaluations, an SUS score of 68 is considered *average*, over 68 is *above average* and below 68 is *below average* [100]. In the *Adjective Rating Scale*, our score *86.56* is considered *excellent* or has a grading scale of *Grade A* [8]. The result of the SUS score indicates that the usability of our proposed framework is promising.

While SUS was initially intended to measure a single dimension of perceived ease-of-use only, research found that it provides a global measure of system satisfaction and sub-scales of usability and learnability. The learnability dimension is provided by items 4 and 10 while the other 8 items provide the usability dimension. Hence it is possible to track and report on both subscales and the global SUS score [66]. But the same authors later suggested to treat the SUS as a unidimensional measure of perceived usability, rather than as usability and learnability subscales [65].

Answer to RQ4

RQ4: Does the combination of visual and textual elements help the scientists compose scientific workflows easily with less cognitive load?
Answer: The result of our first user study, although preliminary, shows that a workflow system can be derived from our framework and can solve real-world scientific data analysis problems. It showed good usability, efficiency, and expressiveness quality attributes in respect to two other popular alternatives for the same domain, Galaxy and Python.
The result of our second user study, the SUS-based post-test usability assessment, reveals that the participants are satisfied with the usability of our system.

7.3 Evaluation of Flexibility

In order to answer research question **RQ5** regarding the flexibility of VizSciFlow, we conducted another user study and collected participants' opinion using the NASA Task Load Index (NASA-TLX) workload assessment tool [48, 49]. Unlike the SUS instrument which collects *post-test* opinions,

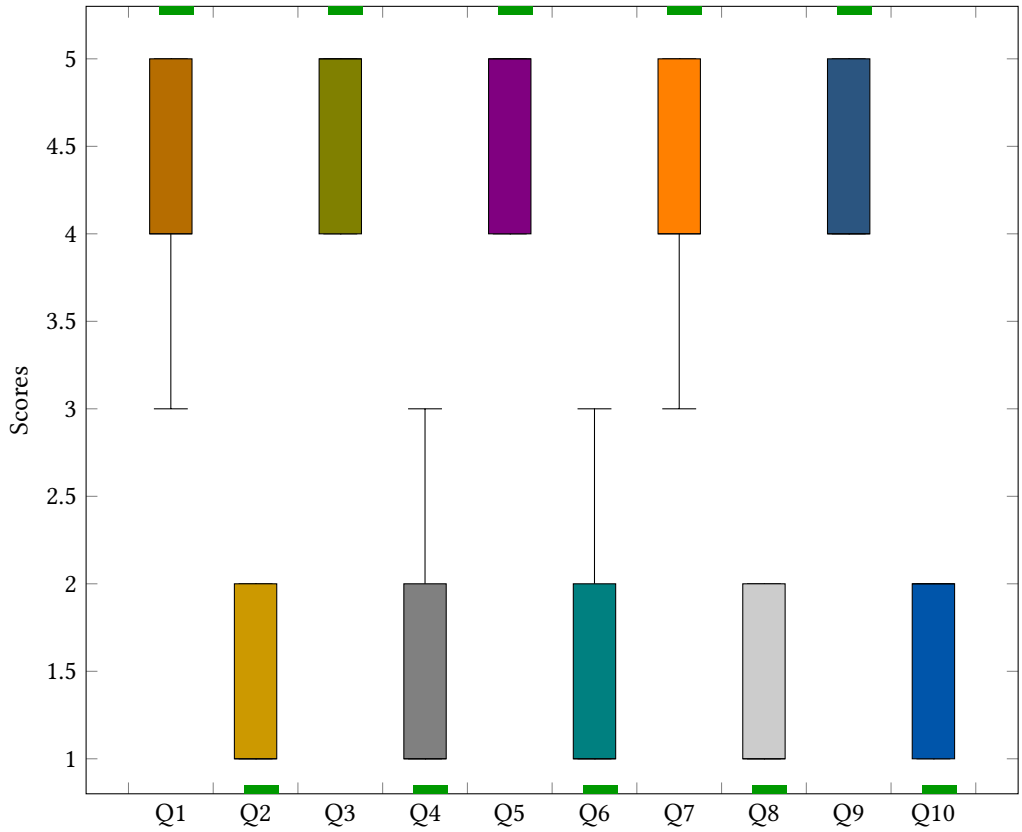


Fig. 10. SUS Usability of VizSciFlow

NASA-TLX is a *post-task* questionnaire that is useful for studying complex products and tasks. NASA-TLX consists of a set of six rating scales to evaluate the workload of the users for a task. The rating scales are mental demand, physical demand, temporal demand, performance, effort and frustration. The participants give a score in a range of possible values of 1 to 21. A coding system similar to SUS is used to transform and map the raw responses to a hundred-point scale.

7.3.1 Participants. We selected 5 graduate students to conduct this user study. All of them were males. 4 of them had experience with bioinformatics tools and technologies while 3 had worked with web-based SWfMSs before. Though 2 of the participants took part in our SUS-based usability study stated before, but they never used the flexibility feature.

7.3.2 Experiment Procedure. At the start of the study, the participants were introduced to the NASA-TLX questionnaire and then they were asked to start a task. After task completion, they were requested to fill out the NASA-TLX questionnaire. In this study, the participants start with an existing workflow and are required to use a new service, which is not currently available in the system. So this study guides the participants to integrate this service into the system which consequently enhances the DSL vocabulary. They can use this service in the workflow immediately.

7.3.3 Results and Discussion. The NASA-TLX responses of the participants are shown in Figure 11. The green bars at the upper and lower X-Axes indicate whether the rating scale is positive

when the value is high or low. The average score of performance is 19 which indicates that all the participants could add a new service to the system with ease.

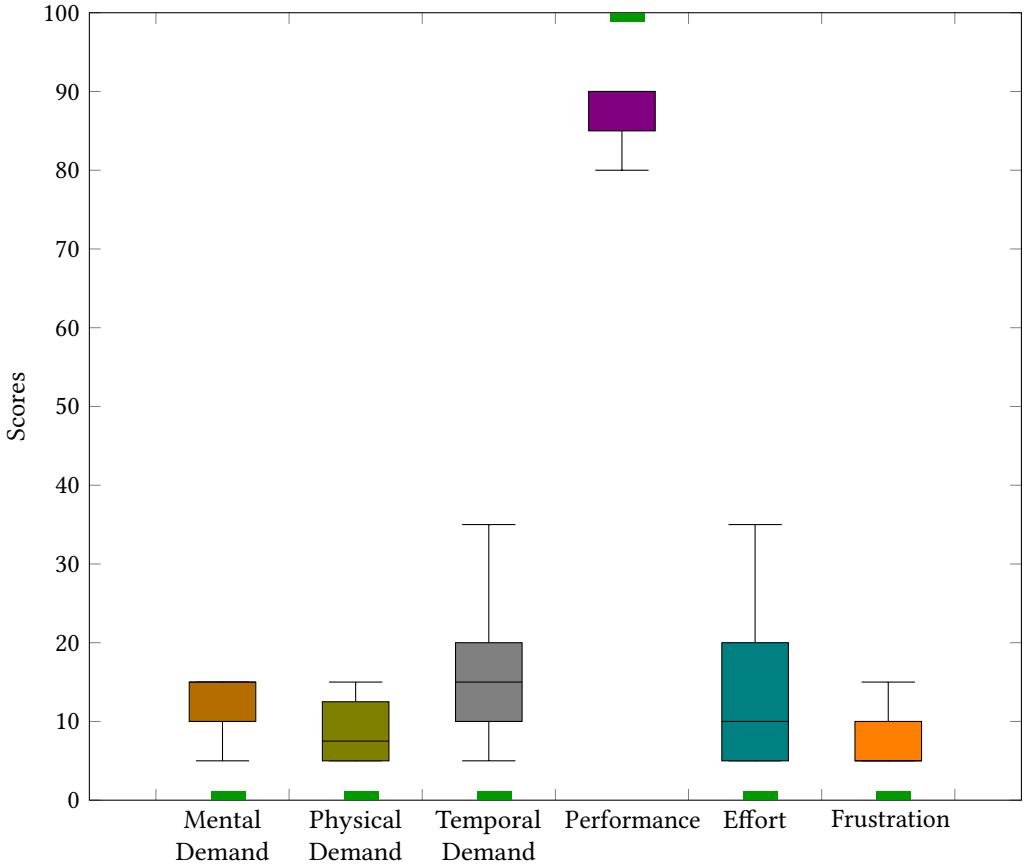


Fig. 11. NASA-TLX Flexibility of VizSciFlow

Answer to RQ5

RQ5: How flexible is the proposed system in incorporating new services or new tools for a specific domain?
Answer: At the presentation layer, our plugin mechanism provides a dialog with a Python code editor and JSON mapper to attach a service or computational module to the system and enhance the DSL vocabulary accordingly. Our post-task user study based-on NASA-TLX questionnaire suggests a low perceived workload as well as high performance while examining the flexibility quality goal.

7.4 Performance Study of Large Data and High Performance Support

In order to answer research question **RQ6** regarding the support of large data volume and high performance processing, we want to demonstrate that our proposed framework can access large

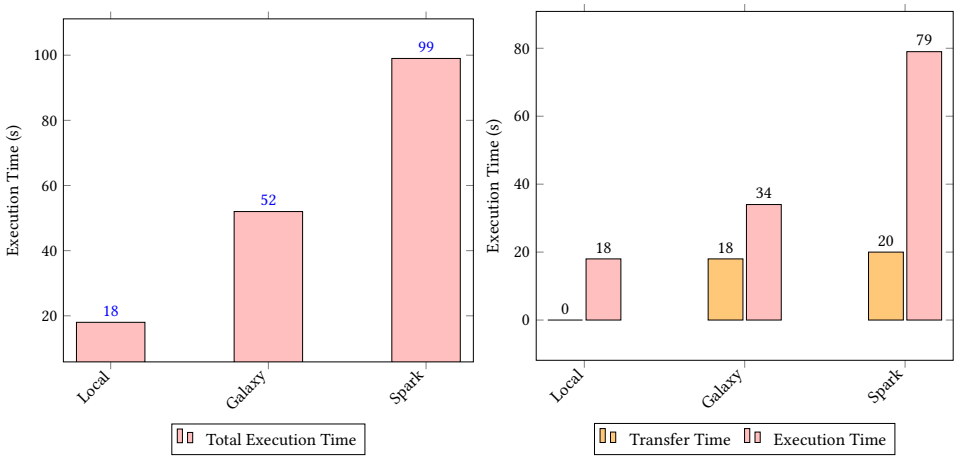
volumes of data and can forward datasets and instructions of selected services to high performance remote machines for execution using the DSL syntax. As the framework suggested, VizSciFlow system uses the Hadoop Distributed File System (HDFS) [116] for large data storage and the Spark [120] cluster computing engine for high performance computing. A scientific workflow spends much of its time during the execution phase. The execution time depends on i) the tool or algorithm which implements the service, ii) network latency, iii) the location of the data, and iv) the location of the runtime host. We selected a set of our services that are hosted on the local VizSciFlow server, Galaxy server, and Spark on a Hadoop YARN cluster. In order to ease the performance comparison we set up a small Hadoop cluster of 3 nodes, each having the same hardware and software configuration as the local server. Similarly, we installed the Galaxy server on another identical machine. We used the following equation to mathematically capture the performance of a workflow execution:

$$T = T_n + T_l + T_e \qquad \text{where,} \quad \begin{aligned} T_n &- \text{data transfer time} \\ T_l &- \text{loading time} \\ T_e &- \text{execution time} \end{aligned} \tag{1}$$

We conducted three experiments with MiSeqSOPData [11], the 16S rRNA example dataset from the Schloss lab [101], which is used for many bioinformatics tutorials and benchmarking. The dataset consists of 40 FASTQ files with total size of 163MB. For sequence alignment we used reference genome **Chr1.cdna**[12](size 14MB) from Rice Annotation Project Database (RAP-DB) [99]. Figure 12a depicts a performance comparison of a bioinformatics workflow which uses POSIX data sources residing on the VizSciFlow server for service execution. It can be noticed from the figure that Galaxy and Spark have much higher execution times than VizSciFlow server. For local execution on VizSciFlow server, no data is transferred over the network and the service module load time is negligible, therefore, $T_n = 0$ and $T_l \approx 0$. For Galaxy and Hadoop services, data is first transferred to the respective servers and then the service is executed. For this experiment, we did not calculate the data transfer time explicitly. Hence, the total execution times for Galaxy and Hadoop services are much higher. In another experiment, the datasets are first transferred to the service hosts and then the same services are executed on those datasets which are now residing on the respective native data storage of service hosts. The transfer time and execution time of this experiment are collected separately and shown in Figure 12b. The execution times on Galaxy and Hadoop hosts are now better than the previous experiment.

In the above scenarios, we notice that Hadoop execution is substantially slower than local or Galaxy execution. Although Spark is intended for long running application on huge volumes of data, the Hadoop YARN cluster has an indispensable library load time for each service execution ($T_{L_h} \gg T_{L_l}$). We depicted another test scenario in Figure 13 which was similar to previous experiment but ran on 40 FASTQ files (163MB). Since Galaxy cannot manipulate directories conveniently, we omitted it from this scenario. In our small Hadoop cluster of 3 nodes, we observed nearly 3 times higher performance than local execution.

[11] https://www.mothur.org/w/images/d/d6/MiSeqSOPData.zip
[12] https://tinyurl.com/u6pdwqz



(a) – Total Execution Time: Data on VizSciFlow Server (implicit transfer to Service Host)

(b) – Data Transfer: VizSciFlow to Service Host Execution Time: Data on Service Host

Fig. 12. Performance: Execution on Few Datasets (Local and Native)

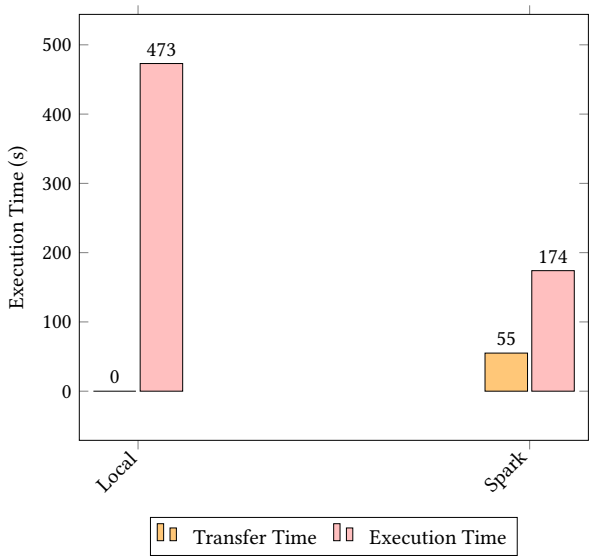


Fig. 13. Performance: Execution on Many Native Datasets

Answer to RQ6

RQ6: Does the proposed system facilitate distributed computational environments?
Answer: We experimented with VizSciFlow to evaluate its support for large data storage and high performance processing. The experiment demonstrates that the system can transparently connect to a distributed storage and high performance computing system to access large data volumes and forward selected services for compute-intensive execution.

8 RESULT ANALYSIS

We proposed a framework for a visually guided DSL workflow modeling system in Section 5 and evaluated it in Section 7. We also implemented the VizSciFlow system as a proof of concept of the proposed framework and demonstrated that it can address the requirements of DSL workflow modeling within an interactive graphical development environment and that it supports extensibility with a flexible plugin architecture, asynchronous job scheduling, sharing and collaboration of services and workflows, large data storage and high performance processing, data-flow graph visualization, and so on.

From our user studies we observed that our DSL-based workflow system with support for secondary notations and development tools in the form of visual representation of workflow elements, autocomplete, context-sensitive help, visualization, reports and other resources leads to a usable, efficient, expressive, and flexible workflow modeling system. With VizSciFlow we addressed real-world problems from the bioinformatics domain and compared the process with two other common alternatives – Galaxy and Python; VizSciFlow performed better in key quality attributes. The post-test assessment of usability using the SUS instrument also shows user satisfaction. Additionally, we demonstrated that the implemented VizSciFlow system using our proposed framework can transparently connect to remote high performance systems, manipulate large data storage, trigger workflow execution and combine output with local datasets.

9 THREATS TO VALIDITY

There are thousands of bioinformatics tools in research and commercial domains as well as in the Galaxy toolshed. We evaluated our system only with a small subset of these tools. Although we evaluated the proposed framework for usability, efficiency, expressiveness, and flexibility of workflow modeling, there still exists the threat of result bias since using other tools may show different results. Scientific tools are mainly involved with data analysis and developers typically provide similar abstract usage patterns to ease tool use by scientists. Therefore, using other tools may not change the modeling procedure and participants' behavior much if a similar abstraction is used. Almost all participants who took part in the user study were computer science graduate students rather than domain experts. However, most of them had prior experience using bioinformatics tools and concepts which partly minimizes the concern. The proposed framework is implemented and evaluated only for a single scientific domain which may potentially have result biases. However, nearly all scientific domains have data analysis as the primary task for workflow. So the workflow patterns are expected to be similar as the ones our proposed framework addresses. We understand that the efficiency is best measured with objective data captured by an instrument independent from a participant's opinion, not with perceived efficiency from subjective assessment as in our user study. Along with our subjective data collection, we also allocated a specific time (30 minutes) to complete the tasks in each system and an observer monitored if the tasks were completed by each participant in the allocated time period. We wanted to show a comparison of VizSciFlow with Galaxy and Python and that domain users can model the same workflow in VizSciFlow faster than in the other two. Another threat to validity is that only a small number of participants took part in the user studies. Usability experts claim that a small number of users can detect a high number of usability improvement opportunities in a product [85] and recommended to plan for five users to catch 85% of the usability problems [86]. An experiment by Kieburtz et al. demonstrates how a meaningful assessment can be performed with only four participants [58]. On the other hand, recent findings found that more participants are required, especially in web testing [13, 106]. As an early evaluation of the framework, though, using a small number of participants is also interesting [9]. We can therefore state that from a preliminary evaluation with few users, we have

gathered some indications that VizSciFlow provides a usable scientific workflow composition and management system, but more participants would be required to evaluate the statistical significance of the results. The test scenarios for performance study definitely do not provide an exhaustive benchmarking of our system. We only experimented with a limited sample size and few operations. However, the intention for the performance study was only to demonstrate that our framework can transparently connect to a remote high performance distributed system, access large data volumes, execute selected services, and return results. The Hadoop cluster is scalable both in storage capacity and performance. New hardware can be attached to it in order to increase its storage capacity and performance. The hardware configuration is also not important for our performance comparison as long as the VizSciFlow server and each computing node of the distributed system are running on the same configuration.

10 CONCLUSION AND FUTURE WORK

In this paper, we identified the requirements of a usable, efficient, expressive, and flexible workflow modeling system and proposed a graphical framework for workflow systems that provides domain-specific interactive visual elements to ease modeling workflow with a domain-specific language. As a proof of concept, we developed a general SWfMS system, VizSciFlow, and customized it for the bioinformatics domain as a product to solve real-world scientific problems. We also demonstrated that code generation by mouse and keyboard interactions with graphical elements representing data sources, services, workflows, reports and other resources along with autocomplete, context-sensitive help, syntax highlighting and incremental search improved the usability, efficiency and expressiveness of the workflow modeling system substantially. Our flexible framework provides an evolutionary approach to enrich the DSL vocabulary by extending the capabilities of the workflow system dynamically. The data-flow graph of the workflow helps to identify the routing of data during service execution. We propose that a visually guided DSL scripting framework is a viable alternative to existing workflow systems and improves usability, expressiveness, efficiency and flexibility.

Real-time collaboration using visual languages has been shown to be effective for multi-disciplinary scientific data analysis [84]. In the future, we will investigate whether workflow modeling with textual languages is also accelerated through real-time collaborative development. We would also like to investigate the usability of VizSciFlow in other domains such as Code Cloning, Source Code Analysis, Plant Phenotyping and Genotyping, and Image Processing. We also want to conduct a more comprehensive qualitative and quantitative evaluation of the framework with real world users in the respective domains.

ACKNOWLEDGEMENTS

This research is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), and by two Canada First Research Excellence Fund (CFREF) grants coordinated by the Global Institute for Food Security (GIFS) and the Global Institute for Water Security (GIWS).

REFERENCES

[1] Ajax.org B.V. 2012. *ACE: The High Performance Code Editor for the Web.* Ajax.org B.V. Retrieved Dec 12, 2019 from https://ace.c9.io

[2] Enis Afgan, Dannon Baker, Marius Van den Beek, Daniel Blankenberg, Dave Bouvier, Martin Čech, John Chilton, Dave Clements, Nate Coraor, Carl Eberhard, et al. 2016. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update. *Nucleic acids research* 44, W1 (2016), W3–W10.

[3] Ilkay Altintas. 2011. Distributed Workflow-driven Analysis of Large-scale Biological Data Using Biokepler. In *Proceedings of the 2Nd International Workshop on Petascal Data Analytics: Challenges and Opportunities* (Seattle, Washington, USA) *(PDAC '11)*. ACM, New York, NY, USA, 41–42. https://doi.org/10.1145/2110205.2110215

[4] Ilkay Altintas, Chad Berkley, Efrat Jaeger, Matthew Jones, Bertram Ludascher, and Steve Mock. 2004. Kepler: An Extensible System for Design and Execution of Scientific Workflows. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM '04)*. IEEE Computer Society, Washington, DC, USA, 423–. https://doi.org/10.1109/SSDBM.2004.44

[5] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, Francisco Curbera, M. Ford, Y. Goland, A. Guzar, Neelakantan Kartha, C. Liu, and Rania Khalaf. 2007. Web Services Business Process Execution Language Version 2.0 (OASIS Standard). *OASIS Standard* 11 (01 2007).

[6] Peter Amstutz, Nebojša Tijanić, Stian Soiland-Reyes, John Kern, Luka Stojanovic, Tim Pierce, John Chilton, Maxim Mikheev, Samuel Lampa, Hervé Ménager, Scott Frazer, Venkat S. Malladi, and Michael R. Crusoe. 2015. Beyond Galaxy: portable workflows and tool definitions with the CWL. https://cesgo.genouest.org/resources/129

[7] Simon Andrews et al. 2010. FastQC: a quality control tool for high throughput sequence data.

[8] Aaron Bangor, Philip Kortum, and James Miller. 2009. Determining what individual SUS scores mean: Adding an adjective rating scale. *Journal of usability studies* 4, 3 (2009), 114–123.

[9] Ankica Barišic, Vasco Amaral, Miguel Goulão, and Bruno Barroca. 2014. Evaluating the usability of domain-specific languages. In *Software Design and Development: Concepts, Methodologies, Tools, and Applications*. IGI Global, 2120–2141.

[10] Ankica Barišić, Pedro Monteiro, Vasco Amaral, Miguel Goulão, and Miguel Monteiro. 2012. Patterns for Evaluating Usability of Domain-Specific Languages. In *Proceedings of the 19th Conference on Pattern Languages of Programs* (Tucson, Arizona) *(PLoP '12)*. The Hillside Group, USA, Article 14, 34 pages.

[11] Adam Barker and Jano Van Hemert. 2007. Scientific workflow: a survey and research directions. In *International Conference on Parallel Processing and Applied Mathematics*. Springer, New York, NY, 746–753.

[12] Jon Bentley. 1986. Programming pearls: little languages. *Commun. ACM* 29, 8 (1986), 711–721.

[13] Nigel Bevan, Carol Barnum, Gilbert Cockton, Jakob Nielsen, Jared Spool, and Dennis Wixon. 2003. The magic number 5: is it enough for web testing?. In *CHI'03 extended abstracts on Human factors in computing systems*. ACM, New York, NY, USA, 698–699.

[14] Alan F Blackwell. 1996. Metacognitive theories of visual programming: what do we think we are doing?. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*. IEEE, 240–246.

[15] Shawn Bowers and Bertram Ludäscher. 2005. Actor-oriented design of scientific workflows. In *International Conference on Conceptual Modeling*. Springer, New York, NY, 369–384.

[16] John Brooke. 2013. SUS: a retrospective. *Journal of usability studies* 8, 2 (2013), 29–40.

[17] John Brooke et al. 1996. SUS-A quick and dirty usability scale. *Usability evaluation in industry* 189, 194 (1996), 4–7.

[18] Steven P Callahan, Juliana Freire, Emanuele Santos, Carlos E Scheidegger, Cláudio T Silva, and Huy T Vo. 2006. VisTrails: visualization meets data management. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. ACM, New York, NY, USA, 745–747.

[19] J Gregory Caporaso, Justin Kuczynski, Jesse Stombaugh, Kyle Bittinger, Frederic D Bushman, Elizabeth K Costello, Noah Fierer, Antonio Gonzalez Pena, Julia K Goodrich, Jeffrey I Gordon, et al. 2010. QIIME allows analysis of high-throughput community sequencing data. *Nature methods* 7, 5 (2010), 335.

[20] Debasish Chakraborti, Banani Roy, Chanchal Roy, and Kevin Schneider. 2018. Optimized Storing of Workflow Outputs through Mining Association Rules. In *2018 IEEE International Conference on Big Data (Big Data)*. 508–515.

[21] Debasish Chakroborti, Banani Roy, Amit Kumar Mondal, Golam Mostaeen, Ralph Deters, Chanchal K. Roy, and Schneider Kevin A. 2020. A Data Management Scheme for Micro-Level Modular Computation-intensive Programs in Big Data Platforms. In *In: Alhajj R., Moshirpour M.,and Far B. (eds), Data Management and Analysis: Case Studies in Education, Healthcare and Beyond, Studies in Big Data*, Vol. 65. 1–20.

[22] Jinjun Chen and Wil van der Aalst. 2007. On scientific workflows. *IEEE Computer Society's Technical Committee for Scalable Computing* 9 (2007).

[23] Michele Chinosi and Alberto Trombetta. 2012. BPMN: An introduction to the standard. *Computer Standards & Interfaces* 34, 1 (2012), 124–134.

[24] David Churches, Gabor Gombas, Andrew Harrison, Jason Maassen, Craig Robinson, Matthew Shields, Ian Taylor, and Ian Wang. 2006. Programming Scientific and Distributed Workflow with Triana Services: Research Articles. *Concurr. Comput. : Pract. Exper.* 18, 10 (Aug. 2006), 1021–1037. https://doi.org/10.1002/cpe.v18:10

[25] Paul Cleary, Matt Bolger, Lachlan Hetherton, Chris Rucinski, David Thomas, and Damien Watkins. 2014. Workspace: A Platform for Delivering Scientific Applications. *Proceedings eResearch* (2014), 4.

[26] Peter JA Cock, Tiago Antao, Jeffrey T Chang, Brad A Chapman, Cymon J Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, et al. 2009. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics* 25, 11 (2009), 1422–1423.

[27] Steve Cook, Gareth Jones, Stuart Kent, and Alan Cameron Wills. 2007. *Domain-specific development with visual studio dsl tools*. Pearson Education.

[28] Fredy Cuenca, Jan Van den Bergh, Kris Luyten, and Karin Coninx. 2014. A domain-specific textual language for rapid prototyping of multimodal interactive systems. In *Proceedings of the 2014 ACM SIGCHI symposium on Engineering interactive computing systems*. ACM, New York, NY, USA, 97–106.

[29] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

[30] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira Da Silva, Miron Livny, et al. 2015. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems* 46 (2015), 17–35.

[31] Paolo Di Tommaso, Maria Chatzou, Evan W Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. 2017. Nextflow enables reproducible computational workflows. *Nature biotechnology* 35, 4 (2017), 316–319.

[32] Marlon Dumas and Arthur ter Hofstede. 2001. *UML Activity Diagrams as a Workflow Specification Language*. Springer Berlin Heidelberg, Berlin, Heidelberg, 76–90. https://doi.org/10.1007/3-540-45441-1_7

[33] elephantlaboratories. 2019. Does anyone use CWL? Does it actually help you get work done? https://www.reddit.com/r/bioinformatics/comments/7gxsk0/does_anyone_use_cwl_does_it_actually_help_you_get/, visited 2019-07-08.

[34] Moritz Eysholdt and Heiko Behrens. 2010. Xtext: implement your language faster than the quick and dirty way. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*. ACM, New York, NY, USA, 307–309.

[35] Rayhan Ferdous, Banani Roy, Chanchal Roy, and Kevin Schneider. 2020. Workflow Provenance for Big Data: From Modelling to Reporting. In *Alhajj R., Moshirpour M.,and Far B. (eds), Data Management and Analysis: Case Studies in Education, Healthcare and Beyond, Studies in Big Data*, Vol. 65. 1–18.

[36] Peter Forbrig, Anke Dittmar, and Mathias Kühn. 2018. A Textual Domain Specific Language for Task Models: Generating Code for CoTaL, CTTE, and HAMSTERS. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. ACM, New York, NY, USA, 5.

[37] Martin Fowler. 2010. *Domain-specific languages*. Pearson Education.

[38] Martin Fowler. 2010. *Domain Specific Languages* (1st ed.). Addison-Wesley Professional, Reading, MA.

[39] Debasish Ghosh. 2011. DSL for the uninitiated. *Commun. ACM* 54, 7 (2011), 44–50.

[40] Belinda Giardine, Cathy Riemer, Ross C Hardison, Richard Burhans, Laura Elnitski, Prachi Shah, Yi Zhang, Daniel Blankenberg, Istvan Albert, James Taylor, et al. 2005. Galaxy: a platform for interactive large-scale genome analysis. *Genome research* 15, 10 (2005), 1451–1455.

[41] Yolanda Gil, Ewa Deelman, Mark Ellisman, Thomas Fahringer, Geoffrey Fox, Dennis Gannon, Carole Goble, et al. 2007. Examining the challenges of scientific workflows. *Computer* 40, 12 (2007), 24–32.

[42] David J. Gilmore and Thomas R. G. Green. 1984. Comprehension and recall of miniature programs. *International Journal of Man-Machine Studies* 21, 1 (1984), 31–48.

[43] Carole A Goble et al. 2010. myExperiment: a repository and social network for the sharing of bioinformatics workflows. *Nucleic acids research* 38, suppl_2 (2010), W677–W682.

[44] Jeremy Goecks, Anton Nekrutenko, et al. 2010. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome biology* 11, 8 (2010), R86.

[45] Katharina Görlach, Mirko Sonntag, Dimka Karastoyanova, Frank Leymann, and Michael Reiter. 2011. Conventional workflow technology for scientific simulation. In *Guide to e-Science*. Springer, New York, NY, 323–352.

[46] Thomas RG Green and Marian Petre. 1992. When visual programs are harder to read than textual programs. In *Human-Computer Interaction: Tasks and Organisation, Proceedings of ECCE-6 (6th European Conference on Cognitive Ergonomics). GC van der Veer, MJ Tauber, S. Bagnarola and M. Antavolits. Rome, CUD*. 167–180.

[47] Thomas RG Green, Marian Petre, and RKE Bellamy. 1991. Comprehensibility of visual and textual programs: A test of superlativism against the'match-mismatch'conjecture. *ESP* 91, 743 (1991), 121–146.

[48] Sandra G Hart. 2006. NASA-task load index (NASA-TLX); 20 years later. In *Proceedings of the human factors and ergonomics society annual meeting*, Vol. 50. Sage publications Sage CA, Los Angeles, CA, 904–908.

[49] Sandra G Hart and Lowell E Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Advances in psychology*. Vol. 52. Elsevier, 139–183.

[50] Petra Heinl, Stefan Horn, Stefan Jablonski, Jens Neeb, et al. 1999. A comprehensive approach to flexibility in workflow management systems. In *ACM SIGSOFT Software Engineering Notes*, Vol. 24. ACM, New York, NY, USA, 79–88.

[51] D Hollingsworth. 1995. Workflow Management Coalition: The Workflow Reference Model. *Workflow Management Coalition* 68 (01 1995).

[52] David Hollingsworth et al. 2004. The workflow reference model: 10 years on. In *Fujitsu Services, UK; Technical Committee Chair of WfMC*. Citeseer, 295–312.

[53] Shawn Hoon, Kiran Kumar Ratnapu, Jer-ming Chia, Balamurugan Kumarasamy, Xiao Juguang, Michele Clamp, Arne Stabenau, Simon Potter, Laura Clarke, and Elia Stupka. 2003. Biopipe: a flexible framework for protocol-based bioinformatics analysis. *Genome Research* 13, 8 (2003), 1904–1915.

[54] Paul Hudak. 1998. Modular domain specific languages and tools. In *Software Reuse, 1998. Proceedings. Fifth International Conference on*. IEEE, 134–142.
[55] Kevin Jacobs and Kacper Surdy. 2016. *Apache Flink: Distributed stream data processing*. Technical Report.
[56] Frédéric Jouault, Jean Bézivin, and Ivan Kurtev. 2006. TCS:: a DSL for the specification of textual concrete syntaxes in model engineering. In *Proceedings of the 5th international conference on Generative programming and component engineering*. ACM, New York, NY, USA, 249–254.
[57] Holden Karau. 2017. Unifying the open big data world: The possibilities∗ of apache BEAM. In *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 3981–3981.
[58] Richard B Kieburtz, Laura McKinney, Jeffrey M Bell, James Hook, Alex Kotov, Jeffrey Lewis, Dino P Oliva, Tim Sheard, Ira Smith, and Lisa Walton. 1996. A software engineering experiment in software component generation. In *Proceedings of the 18th international conference on Software engineering*. IEEE Computer Society, 542–552.
[59] Dimitrios S Kolovos, Richard F Paige, and Fiona AC Polack. 2006. Eclipse development tools for epsilon. In *Eclipse Summit Europe, Eclipse Modeling Symposium*, Vol. 20062. 200.
[60] Tomaž Kosar, Sudev Bohra, and Marjan Mernik. 2016. Domain-specific languages: A systematic mapping study. *Information and Software Technology* 71 (2016), 77–91.
[61] Johannes Köster and Sven Rahmann. 2012. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics* 28, 19 (2012), 2520–2522.
[62] A. Lajmi, J. Martinez, and T. Ziadi. 2014. DSLFORGE: Textual modeling on the web. *CEUR Workshop Proceedings* 1255 (01 2014), 25–29.
[63] Peter Lawrence (Ed.). 1997. *Workflow Handbook 1997*. John Wiley & Sons, Inc., New York, NY, USA.
[64] Jeremy Leipzig. 2017. A review of bioinformatic pipeline frameworks. *Briefings in bioinformatics* 18, 3 (2017), 530–536.
[65] James Jim R Lewis and Jeff Sauro. 2017. Revisiting the factor structure of the System Usability Scale. *Journal of Usability Studies* 12, 4 (2017), 183–192.
[66] James R Lewis and Jeff Sauro. 2009. The factor structure of the system usability scale. In *International conference on human centered design*. Springer, New York, NY, 94–103.
[67] Heng Li. 2013. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *ArXiv* 1303 (2013).
[68] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, and Richard Durbin. 2009. The sequence alignment/map format and SAMtools. *Bioinformatics* 25, 16 (2009), 2078–2079.
[69] William Lidwell et al. 2010. *Universal principles of design, revised and updated: 125 ways to enhance usability, influence perception, increase appeal, make better design decisions, and teach through design*. Rockport Pub.
[70] Cui Lin, Shiyong Lu, Xubo Fei, Artem Chebotko, Darshan Pai, Zhaoqiang Lai, Farshad Fotouhi, and Jing Hua. 2009. A reference architecture for scientific workflow management systems and the VIEW SOA solution. *IEEE Transactions on Services Computing* 2, 1 (2009), 79–92.
[71] Ji Liu, Esther Pacitti, Patrick Valduriez, and Marta Mattoso. 2015. A survey of data-intensive scientific workflow management. *Journal of Grid Computing* 13, 4 (2015), 457–493.
[72] Bertram Ludäscher et al. 2009. Scientific process automation and workflow management. In *Scientific Data Management: Challenges, Technology, and Deployment*. CRC press.
[73] Bertram Ludäscher, Mathias Weske, Timothy McPhillips, and Shawn Bowers. 2009. Scientific workflows: Business as usual?. In *International Conference on Business Process Management*. Springer, New York, NY, 31–47.
[74] Bertram Ludäscher, Ilkay Altintas, and Amarnath Gupta. 2003. Compiling Abstract Scientific Workflows into Web Service Workflows. In *15th International Conference on Scientific and Statistical Database Management, 2003.*, Vol. 2003. IEEE, 251–254. https://doi.org/10.1109/SSDM.2003.1214990
[75] Tanja Magoč and Steven L Salzberg. 2011. FLASH: fast length adjustment of short reads to improve genome assemblies. *Bioinformatics* 27, 21 (2011), 2957–2963.
[76] Ketan Maheshwari and Johan Montagnat. 2010. Scientific workflow development using both visual and script-based representation. In *2010 6th World Congress on Services*. IEEE, 328–335.
[77] Seema Maitrey and C.K. Jha. 2015. MapReduce: Simplified Data Analysis of Big Data. *Procedia Computer Science* 57 (2015), 563 – 571. https://doi.org/10.1016/j.procs.2015.07.392 3rd International Conference on Recent Trends in Computing 2015 (ICRTC-2015).
[78] Shalil Majithia, Matthew Shields, Ian Taylor, and Ian Wang. 2004. Triana: A graphical web service composition and execution toolkit. In *Web Services, 2004. Proceedings. IEEE International Conference on*. IEEE, 514–521.
[79] James Malone, Andy Brown, Allyson L Lister, Jon Ison, Duncan Hull, Helen Parkinson, and Robert Stevens. 2014. The Software Ontology (SWO): a resource for reproducibility in biomedical data analysis, curation and digital preservation. *Journal of biomedical semantics* 5, 1 (2014), 25.
[80] Daniel D McCracken and Edwin D Reilly. 2003. Backus-naur form (bnf). (2003).
[81] Paul McGuire. 2007. *Getting started with pyparsing*. " O'Reilly Media, Inc.", 1005 Gravenstein Highway North, Sebastopol, CA 95472, USA.

[82] Marjan Mernik, Jan Heering, and Anthony M Sloane. 2005. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)* 37, 4 (2005), 316–344.

[83] Golam Mostaeen, Banani Roy, Chanchal Roy, and Kevin Schneider. 2018. Fine-Grained Attribute Level Locking Scheme for Collaborative Scientific Workflow Development. In *2018 IEEE International Conference on Services Computing (SCC)*. 273–277.

[84] Golam Mostaeen, Banani Roy, Chanchal Roy, and Kevin Schneider. 2019. Designing for Real-Time Groupware Systems to Support Complex Scientific Data Analysis. *Journal Proceedings of the ACM on Human-Computer Interaction* 3, EICS, Article 9 (June 2019), 28 pages.

[85] Jakob Nielsen. 1994. *Usability engineering*. Elsevier.

[86] Jakob Nielsen. 2000. Why You Only Need to Test with 5 Users, Jakob Nielsen's Alertbox. "https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/"

[87] Shadi A Noghabi et al. 2017. Samza: stateful scalable stream processing at LinkedIn. *Proceedings of the VLDB Endowment* 10, 12 (2017), 1634–1645.

[88] Andres Ojamaa, Hele-Mai Haav, and Jaan Penjam. 2015. Semi-automated generation of DSL meta models from formal domain ontologies. In *Model and Data Engineering*. Springer, New York, NY, 3–15.

[89] Chris Parnin, Eric Helms, Chris Atlee, Harley Boughton, Mark Ghattas, Andy Glover, James Holman, John Micco, et al. 2017. The top 10 adages in continuous deployment. *IEEE Software* 34, 3 (2017), 86–95.

[90] Terence Parr. 2013. *The definitive ANTLR 4 reference*. Pragmatic Bookshelf.

[91] Maja Pesic, Helen Schonenberg, and Wil van der Aalst. 2010. Declarative workflow. In *Modern Business Process Automation*. Springer, 175–201.

[92] Marian Petre. 1995. Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming. *Commun. ACM* 38, 6 (June 1995), 33–44. https://doi.org/10.1145/203241.203251

[93] Carl Adam Petri. 1962. *Kommunikation mit Automaten*. Ph.D. Dissertation. Universität Hamburg.

[94] q2studio: A graphical user interface for QIIME 2 2017. QIIME 2 Studio (q2studio). Retrieved December 12, 2019 from https://docs.qiime2.org/2019.10/interfaces/q2studio/

[95] Akond Rahman et al. 2018. What questions do programmers ask about configuration as code?. In *Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering*. ACM, New York, NY, USA, 16–22.

[96] Anthony Rowe, Dimitrios Kalaitzopoulos, Michelle Osmond, Moustafa Ghanem, and Yike Guo. 2003. The discovery net system for high throughput bioinformatics. *Bioinformatics* 19, suppl 1 (2003), i225–i231.

[97] Banani Roy, Amit Kumar Mondal, Chanchal K Roy, Kevin A Schneider, and Kawser Wazed. 2017. Towards a reference architecture for cloud-based plant genotyping and phenotyping analysis frameworks. In *2017 IEEE International Conference on Software Architecture (ICSA)*. IEEE, 41–50.

[98] Simon P Sadedin, Bernard Pope, and Alicia Oshlack. 2012. Bpipe: a tool for running and managing bioinformatics pipelines. *Bioinformatics* 28, 11 (2012), 1525–1526.

[99] Hiroaki Sakai et al. 2013. Rice Annotation Project Database (RAP-DB): an integrative and interactive database for rice genomics. *Plant and Cell Physiology* 54, 2 (2013), e6–e6.

[100] Jeff Sauro. 2011. Measuring usability with the system usability scale (SUS).

[101] Patrick D Schloss, Sarah L Westcott, Thomas Ryabin, Justine R Hall, Martin Hartmann, Emily B Hollister, Ryan A Lesniewski, Brian B Oakley, Donovan H Parks, Courtney J Robinson, et al. 2009. Introducing mothur: open-source, platform-independent, community-supported software for describing and comparing microbial communities. *Appl. Environ. Microbiol.* 75, 23 (2009), 7537–7541.

[102] Helen Schonenberg, Ronny Mans, Nick Russell, Nataliya Mulyar, and Wil van der Aalst. 2008. Process flexibility: A survey of contemporary approaches. In *Advances in enterprise engineering I*. Springer, New York, NY, 16–30.

[103] Clare Sloggett, Nuwan Goonasekera, and Enis Afgan. 2013. BioBlend: automating pipeline analyses within Galaxy and CloudMan. *Bioinformatics* 29, 13 (2013), 1685–1686.

[104] Barry Smith, Michael Ashburner, Cornelius Rosse, Jonathan Bard, William Bug, Werner Ceusters, Louis J Goldberg, Karen Eilbeck, Amelia Ireland, Christopher J Mungall, et al. 2007. The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. *Nature biotechnology* 25, 11 (2007), 1251.

[105] Kenia Sousa, Jean Vanderdonckt, Brian Henderson-Sellers, et al. 2012. Evaluating a graphical notation for modelling software development methodologies. *Journal of Visual Languages & Computing* 23, 4 (2012), 195–212.

[106] Jared Spool and Will Schroeder. 2001. Testing web sites: Five users is nowhere near enough. In *CHI'01 extended abstracts on Human factors in computing systems*. ACM, New York, NY, USA, 285–286.

[107] Jonathan Sprinkle, Marjan Mernik, Juha-Pekka Tolvanen, and Diomidis Spinellis. 2009. Guest editors' introduction: What kinds of nails need a domain-specific hammer? *IEEE software* 26, 4 (2009), 15–18.

[108] Robert Tairas, Marjan Mernik, et al. 2008. Using ontologies in the domain analysis of domain-specific languages. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, New York, NY, 332–342.

[109] Gabor Terstyanszky, Tamas Kukla, Tamas Kiss, Peter Kacsuk, et al. 2014. Enabling scientific workflow sharing through coarse-grained interoperability. *Future Generation Computer Systems* 37 (2014), 46–59.

[110] Tom Tullis and Bill Albert. 2013. Chapter 6 - Self-Reported Metrics. In *Measuring the User Experience (Second Edition)* (second edition ed.), Tom Tullis and Bill Albert (Eds.). Morgan Kaufmann, Boston, 121 – 161. https://doi.org/10.1016/B978-0-12-415781-1.00006-6

[111] Jan Van den Bergh and Kris Luyten. 2017. DICE-R: Defining human-robot interaction with composite events. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. ACM, 117–122.

[112] Wil van der Aalst and Arthur ter Hofstede. 1999. Workflow Patterns Initiative. Retrieved December 12, 2019 from http://www.workflowpatterns.com

[113] Wil van der Aalst and Arthur ter Hofstede. 2005. YAWL: yet another workflow language. *Information systems* 30, 4 (2005), 245–275.

[114] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. 2013. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, New York, NY, USA, 5.

[115] Markus Völter, Sebastian Benz, Christian Dietrich, et al. 2013. *DSL Engineering - Designing, Implementing and Using Domain-Specific Languages*. dslbook.org. http://www.dslbook.org

[116] Tom White. 2009. *Hadoop: The Definitive Guide* (1st ed.). O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, USA.

[117] Guido Wirtz, Mathias Weske, and Holger Giese. 2000. *Extending UML with Workflow Modeling Capabilities*. Springer Berlin Heidelberg, Berlin, Heidelberg, 30–41. https://doi.org/10.1007/10722620_3

[118] Katherine Wolstencroft, Robert Haines, Fellows, et al. 2013. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic acids research* 41, W1 (2013), W557–W561.

[119] Xiaorong Xiang and Gregory Madey. 2007. Improving the reuse of ScientificWorkflows and their by-products. In *IEEE International Conference on Web Services (ICWS 2007)*. IEEE, 792–799.

[120] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster computing with working sets. *HotCloud* 10, 10-10 (2010), 95.

[121] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. 2016. Apache spark: a unified engine for big data processing. *Commun. ACM* 59, 11 (2016), 56–65.

[122] Jiajie Zhang, Kassian Kobert, Tomáš Flouri, and Alexandros Stamatakis. 2013. PEAR: a fast and accurate Illumina Paired-End reAd mergeR. *Bioinformatics* 30, 5 (2013), 614–620.

Received July 2019; revised December 2019; accepted January 2020