Meta-Workflows: Pattern-based Interoperability between Galaxy and Taverna

Mohamed Abouelhoda
Center for Informatics Sciences
Nile University
Smart Village, Giza, Egypt
mabouelhoda@yahoo.com
mabouelhoda@nileuniversity.edu.eg

Shady Alaa
Center for Informatics Sciences
Nile University
Smart Village, Giza, Egypt
salaa@nileu.edu.eg

Moustafa Ghanem
Department of Computing
Imperial College London
180 Queens Gate
London SW7 2AZ
mmg@doc.ic.uk

ABSTRACT

Taverna and Galaxy are two workflow systems developed specifically for bioinformatics applications. For sequence analysis applications, some tasks can be implemented easily on one system but would be difficult, or infeasible, to be implemented on the other. One solution to overcome this situation is to combine both tools in a unified framework that seamlessly makes use of the best features of each tool. In this paper, we present the architecture and implementation of a high-level system that provides such a solution. Our approach is based on meta-workflows and workflow patterns. We present a case study about the design of universal primers to demonstrate the capabilities of our system and to explain how the interplay between Taverna and Galaxy simplifies the analysis process.

1. INTRODUCTION

1.1 Scientific workflow systems

The use of the scientific workflow paradigm for designing and executing data processing and analysis pipelines has gained wide attention over the past decade. The paradigm addresses many problems faced by researchers working in the bioinformatics domain, and in particular genome analysis applications. For such applications, users typically collect and analyze sequence data from multiple sources. The analysis itself is conducted using multiple software tools and proceeds in a staged fashion with the output of one tool feeding an input to another. Many of the tools can be compute intensive and their implementation typically makes use of high performance computing resources which can be hosted within the user's organization or remotely at another organization. The same applies to the data sources used in the analysis where some could be hosted locally and others can be hosted remotely. Scientific workflow systems provide an easy-to-use metaphor that can be used for both designing and executing bioinformatics applications in such a heterogeneous environment.

Informally, a *workflow* is a general term used to describe the steps needed to solve a certain task. An abstract representation of a workflow is typically a directed graph where each node represents a task to be executed and edges represent either data flow or execution dependencies between different tasks. With the help of a scientific workflow system, the nodes in the graph can be mapped to real data sources and software components that can be executed either locally on the user machine or remotely at distributed locations. Two key advantages typically cited for using scientific workflow systems. The first advantage is that they can provide an intuitive and high-level model that can be used by application scientists themselves for building complex applications. The application scientists with little or no programming expertise can focus on the logic of their applications and no longer need to worry about the technical details of accessing and invoking the software components and/or distributed data sources they need. Such details can be delegated to the workflow system itself. The second advantage is that recording application steps as a workflow provides an efficient means for assuring reproducibility of the analysis results and allows sharing of the workflows themselves between users.

Building on these advantages, a large number of academic scientific workflow systems have been developed in the past decade. Examples include Discovery Net [16, 9], Taverna [14, 11], Triana [18, 17], Kepler [12] and OMII-BPEL [5]. These different tools share many common features. They typically provide a visual front-end that enables users to compose their workflows using the graph-based metaphor. This visual interface also allows users to define individual steps that are possibly mapped to specific data sources and executable software tools and to define the associated parameters. Once the workflow is composed, it is submitted to an execution engine that handles the invocation of the tools and also handles the data transfer between them.

However, different scientific workflow systems have been developed with different applications and use cases in mind. This has inevitability influenced what is considered as an atomic task, or node, in each system. Furthermore, the different systems build on slightly different programming models, and thus the meaning of an arc connecting two nodes

WANDS Indianapolis, IN, USA, June 6th, 2010

typically differs from one system to another. As a result, the same conceptual or abstract workflow designed by the user on a whiteboard typically ends up looking, and behaving, differently on different workflow systems. A recent survey and comparison of some of these workflow systems and their underlying programming models can be found in [6].

1.2 Motivation

Our work in this paper stems from our recent work and experience in building the NUBIOS system, a bioinformatics resource for the Egyptian bioinformatics community (`www.nubios.nileu.edu.eg`) using locally installed high performance computing facilities while still allowing access to remote tools. NUBIOS also aims at providing a simplified user interface enabling users to compose and execute their own workflows, and also to have access to workflow libraries that pre-package widely used tasks. To enable this, we considered and evaluated three existing workflow systems; InforSense, Galaxy and Taverna as candidates for use within our infrastructure. The InforSense system is a commercial product based on the outputs of the Discovery Net project [9]. The other two systems are based on an open source frameworks. Although all systems are widely used in bioinformatics, each offers different advantages in terms of usability and performance due to differences in their internal language assumptions and also differences in their architecture.

Our objective in this paper is to investigate the development of simple methods that enable interoperability between different scientific workflow systems in general, and the two open source systems; Taverna and Galaxy in particular. In doing so, our approach is based on developing a meta-workflow approach with a simplified user GUI that looks more similar to a traditional flow chart, and thus more accessible to the user. We also build mapping tools based on the paradigm of *workflow patterns* that can translate the implementation of the meta-workflow to either system. The outcome of our work that interests the bioinformatics scientists is a software system, called *Tavaxy*, that combines the advantages of both systems for the sequence analysis domain.

This paper is organized as follows: Section 2 provides a brief comparison between Galaxy and Taverna, and introduces the workflow patterns. Section 3 describes a simple demonstration application in sequence analysis and its implementation in both systems. Section 4 introduces Tavaxy and its architecture based on workflow patterns. In Section 5, we implement the workflow of the demonstration application in Tavaxy, and show also the interplay between Galaxy and Taverna in Tavaxy. Conclusions are in Section 6.

2. BACKGROUND

Taverna[14, 11] was developed as general purpose scientific workflow tool as part of the ^{my}Grid e-science initiative with the aim of simplifying access to, and coordination of, remote Web and Grid services within a particular application. As a general purpose workflow tool, Taverna can generally work with any data type and does not come with any prepackaged sequence analysis tools. However, a large library of existing tools that operate on genomic sequences has been developed and integrated by the Taverna user community. In contrast, Galaxy[10] provides an integrated system that supports the retrieval of sequences from genome databases (particularly UCSC and Ensembl) and for processing, annotating and analyzing these sequences using a large library of software tools that comes with the system. It should be noted that both systems are easily extensible allowing integration of different tools, and in general can be provide the same functionality to the user. Yet, each system has a set of unique features that are not supported by the other, thus ultimately affecting their ease of use. In the remainder of this section, we provide an overview of both systems and then provide a brief comparison of their features.

2.1 Taverna

The Taverna workbench follows an explicit model for workflow authoring. Its main entry point is the workflow editor allowing users to drag, drop and connect components representing different data sources and tools. The system is built on a decoupled architecture that separates the editor from the enactment engine. Workflows in Taverna are represented internally in the SCUFL (Simple Conceptual Unified Flow Language) for representing workflows as DAGs (Directed Acyclic Graphs). SCUFL supports predominately a data flow model of execution. Nodes in the graph represent *processors* which transform input data to output data. A processor with no input acts as a data source and a processor with no outputs acts as a data sink. The directed arcs between the nodes are generally channels for passing the output of one processor as input to another.

Taverna supports iterative execution of a processor by providing a set of configurable iteration strategies that specify how to iterate over a list of inputs. In addition, Taverna supports a number of control flow constructs for organizing control flow operations. For example, an arc connecting two nodes can simply indicate sequential dependency between two nodes with no data flowing on it, and conditional branching is achieved by passing a special 'failure' token on one of is output branches.

Data source and sink nodes are widely used in Taverna. The advantage of the feature is that it makes the dependence on the parameters explicit and allows them to be easily controlled either by the user or by other processors allows the user to explicitly control how each output is handled. The key disadvantage is that heavily relying on using them ends up generating workflows with a large number of nodes even for a simple task.

2.2 Galaxy

Galaxy follows mainly an implicit model for workflow creation while still allowing the user to access, modify and share the created workflows explicitly. The main entry point is a portal-like interface where the user is presented with a large list of sequence manipulation tools, each with a special user interface allowing the user to upload data, set execution parameters including where the output is stored and to submit a task for execution. As the user submits more tasks where the output of one task is used as input to another, the system automatically records a history log. This log is then presented to the user as a graphical workflow which can be edited and submitted for further executions as needed.

Similar to Taverna, Galaxy supports a data flow model operation with the outputs of one node flowing as input to other nodes. However, it does not rely on an explicit workflow language to represent workflows. The properties of each node (e.g. its parameters) and the properties of each link (i.e. the associated nodes and flowing data types) are

simply stored in a relational database. If a user modifies a node or a link, the database is directly updated.

2.3 Comparison

As discussed previously, our primary aim was to choose a workflow tool that would simplify access and execution of sequence analysis tasks on our bioinformatics server as well as using remote resources. Some of the tools executing locally are developed in house, e.g. [2, 1] and execute on our local high performance cluster. For other tools such as BLAST which require updated databases we typically rely on using remote servers.

2.3.1 Features of Galaxy not in Taverna

- **Local service execution:** The complete Galaxy system can be installed to run locally or it can be used through web-based interface at the Galaxy server (`http://main.g2.bx.psu.edu`). For our own NUBIOS infrastructure this is ideal. Relying solely on remote servers executing in other countries introduces performance penalties when Internet connectivity becomes limited. This is a common occurrence in many developing countries.
- **Native support for sequence manipulation:** Galaxy has a large built-in library of sequence manipulation and analysis tools. These include, among others, format converters, analysis packages, such as EMBOSS [15], and data processing utilities supporting operations like *joining* and *filtering*. Interestingly, these operations work not only on traditional string or number keys as in the database domain, but works also on intervals specifying positions in the genomic sequences. Taverna, on the other hand, requires integrating these tools separately. This requires programming experience to wrap the locally installed tools within web-service interface.
- **Enhanced usability:** The usability in Galaxy is an attractive feature, because its portal-like interface provides a more natural representation of functionalities. Furthermore, the workflows generated in Galaxy tend to look much simpler than those generated in Taverna.
- **Scheduling support for HPC cluster operation:** Galaxy, once installed, runs directly on local high performance computing cluster, including HPC implementation of single tools and also scheduling of complete workflows. Taverna on the other hand does not provide scheduling functionality which needs to be implemented separately to make use of our HPC facilities.

2.3.2 Features in Taverna not in Galaxy

- **Support for remote services:** Taverna is web-service based and requires no installation of the analysis tools, allowing access to remote servers. This is and advantage using local services only is not possible for other institutions in developing countries that have limited computational resources and do not host up-to-date databases.
- **Support for control flow operations:** Taverna contains more workflow constructs such as *if-else* and *loops* which are typically required in many tasks. This accordingly allows the design and execution of complex workflows. These constructs are not directly supported in Galaxy, and accordingly puts a limitation on the types of workflows that can be executed on Galaxy.
- **Explicit XML representation of workflows:** Workflows in Taverna are represented and stored in the SCUFL format which is simple to share and manipulate outside the editor. Galaxy uses no explicit representation specifying the workflows. Instead, a workflow is stored directly on its built-in workflow database. Sharing a workflow between two users is still possible and is achieved by setting appropriate permissions within their user accounts.
- **Access to a service directory and workflow repository:** Taverna includes support for a service directory and ontology based search tools. These features, which are basically attributed to the simple SCUFL format, have created a wide community of users and contributors. This provides a valuable resource allowing us to locate, retrieve and re-purpose existing tools and workflows. Galaxy currently does not support such registry facility.

2.4 Workflow patterns

Workflow patterns are set of constructs that model a (usually recurrent) requirement (sub-process); the description of these constructs is an integral part of the pattern definition. Workflow patterns, despite being less formal than workflow languages, have recently received increasing popularity due to their practical relevance in comparing and understanding the features of different workflow language implementations.

As originally introduced in [19], workflow patterns were used to characterize business workflows and were categorized into *control flow patterns*, which specify the execution of activities, *data flow patterns*, which specify the handling and access of data items, *resource and operational patterns*, which organize the execution of tasks on the available resources, and *exception handling patterns*, which handle errors during workflow execution. We note that the concept of patterns is in general applicable to scientific workflows and some of the specified patterns for business processes can still be used for scientific ones.

In this section, we review some workflow patterns that can be used in comparing the operation of Taverna and Galaxy. For compactness of presentation, we will specify control flow pattern in association with the flow of involved data.

1. **Sequence:** In this pattern, which can be referred to as *pipeline*, task B runs after the execution of task A. The data that is produced by A and has to be processed by B moves over an edge whose start is an output port at A and destination is an input port at B. The concept of ports allows to select among different results of A the pieces of data to be processed by B. Desired execution dependencies involving no data can be achieved by dummy output from A to B. That is, each edge should involve data transfer and is specified by the data it passes from task A to B. This pattern is supported by both Taverna and Galaxy.
2. **Synchronous Merge:** A task is invoked only if all incident tasks are executed; Figure 1(a) depicts this

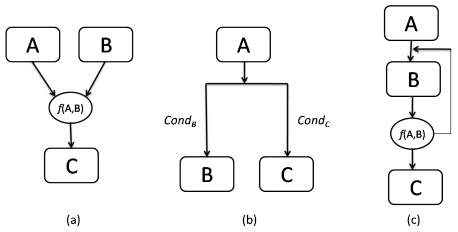

Figure 1: Control workflow patterns modelling the execution of workflow tasks.

pattern with three tasks A, B, and C, where task A and B should be completed before C. This pattern also specifies that task C takes two inputs (one from A and another from B) and the data flowing from A and B to C goes to different input ports. This pattern is supported by both Taverna and Galaxy.

3. **Multi-choice Fork:** This includes the use of *if-else* and *switch-case* constructs to execute a task if a condition is satisfied. In this pattern, the data flows to the pre-specified input ports. This pattern is supported only by Taverna.

4. **(Parallel) synchronized fork:** Figure 1(b) depicts this pattern with three tasks A, B, and C. Tasks B and C run after the execution of A (possibly in parallel depending on the workflow execution algorithm). The data output from A flows according to two schemes: 1) One copy is passed to B and another one to C. 2) Different data items passed to B and C. This pattern is supported by both Taverna and Galaxy.

5. **Simple iteration:** This pattern specifies repetition of a workflow task. In Figure 1(c), the node B, which could be a sub-workflow is repeated many times. The number of iterations can be either fixed or dependent on the data produced at each step. In each iteration, a piece of output of task B can replace the corresponding piece of input. For example, a parameter file can be passed to B and at each iteration this parameter file is modified and passed again to B. But we stress that the *iteration* pattern has specified input and output ports. This pattern is supported only by Taverna.

3. MOTIVATING EXAMPLE

While conducting our evaluation we compared the implementation of a number of sequence analysis workflows in both Taverna and Galaxy. In this section we focus on one of these workflows which is simple enough to highlight and contrast some of their key features.

3.1 Design of Universal Primers

Polychain Reaction (PCR) is a lab technology used to isolate and multiply DNA segments. A *primer* is a short DNA sequence used to start the PCR process. A primer sequence is chosen such that it is complementary to a subsequence of the DNA segment to be isolated and multiplied. In a mixture of DNA segments representing a number of genes, the primer binds only to the complementary part of it and the multiplication will be specific to this DNA segment.

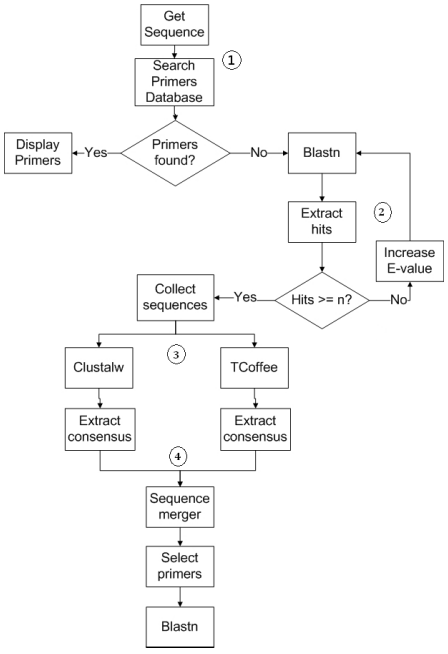

Figure 2: High-level view of the Universal Primer Workflow. The *extract consensus* step identifies the conserved regions. Parsing and handling the sequences are usually implemented using Perl/Pythons scripts or some utility tools.

To reduce the cost, researchers opt to design a single primer that can bind to homologs of a gene existing in multiple taxa or species. This type of primers is called *universal primer*. The design of universal primer is more complicated than the design of traditional primers, because the DNA sequences of these genes are not identical and acquire mutation over time, especially in bacteria and viruses. The computational challenge in universal primer design is thus to select primers which are unique to the gene of interest but similar enough among multiple homologs of it.

To this end, the following workflow is used: First, the gene homologs are searched for in the database using the program BLAST [3, 4]. Then multiple alignment of the genes of interest is computed using programs like ClustalW [13] or T-coffee [13]. (Scientists usually use multiple programs to have more reliable results.) The highly conserved (similar) regions in this alignment are identified and the respective sub-sequences are extracted as potential primers. Finally, the physical properties (basically melting temperature) of these potential primers are evaluated using a program like ePrimer3 (EMBOSS version of Primer3 [13]). Those primers passing the evaluation are reported. Each of these steps has a set of parameters, and a step is repeated with less stringent parameters if no output is obtained. Two confirmatory steps are added to the design of universal primers: The first is to run a search in a universal primer database to check for the availability of known primers for the set of input sequences. The second is to run the program BLASTN after checking the physical properties to ensure the specificity of the primer sequences to the input set of genes.

3.2 High-level workflow representation

Figure 2 shows an abstract whiteboard view of the whole

workflow based on conventional flowchart notation. The user submits his own gene sequence or a reference to it in a biological database. The universal primer database can be available at the user site or at remote site. All the programs and scripts in this workflow can run locally. But for some institutions/users it is better to use the remote BLAST server, which is faster and connected to the up-to-date version of the nucleotide database.

3.3 Involved Patterns

This workflow, despite being simple, includes all the patterns introduced above.

- Sequence: This is easy to observe from the figure. For example, the tasks of running T-coffee and computing the consensus run one after another.
- Simple Fork: The collected sequences from BLAST are passed to ClustalW and T-coffee to be aligned. Each of these two programs take the same copy of the data.
- Multi-choice Fork: We have an *if-else* condition to check availability of primer in a database.
- Merge: The conserved regions obtained by the two programs are merged together in one file. In Galaxy, we can make use of the advanced merge operations, where repeated conserved regions mapped to the same location are filtered out.
- Iteration: The first BLAST step is repeated many times until enough number of sequences is retrieved.

3.4 Galaxy and Taverna implementation

Figures 3 and 4 shows the implementation of the universal primer workflow in both Galaxy and Taverna. The numbers in the diagrams represent the workflow patterns used as discussed in the next section. (`1` refers to multi-choice, `2` refers to iteration, `3` refers to fork, and `4` refers to merge.) The iteration and multi-choice fork are missing in the figure of Galaxy, because they are not supported by it, and in essence our abstract definition of the workflow cannot be directly implemented in the system. These two patterns are, however, implicitly supported by Taverna, especially the iteration involving BLAST. The parameters of each task are more implicit in Galaxy than in Taverna.

4. TAVAXY: A META-WORKFLOW SYSTEM

As seen in previous sections, making a straightforward choice between Taverna and Galaxy for our purposes is not easy. A pragmatic approach is to attempt to maintain a hybrid environment where we can use both systems and to enable interoperability, or at least co-existence, between them.

4.1 Workflow Interoperability Approaches

Investigating interoperability between workflow systems is not a new idea and has been previously attempted in various contexts since the mid nineties. The WfMC (Workflow Management Coalition) [20] defines eight models, or approaches, for achieving interoperability between workflow systems. These are 1) No interoperability; 2) Co-existence; 3) Gateway API; 4) Limited Common API Subset; 5) Complete Workflow API; 6) Shared Definition Formats; 7) Protocol Compatibility; and 8) Common Look and Feel Utilities.

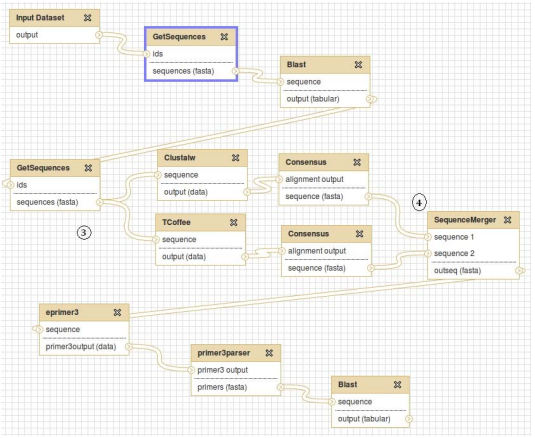

Figure 3: Universal Primer Workflow in Galaxy. Numbers denote workflow patterns used. (For more details, zoom in original PDF.)

We note that in the WfMC models Levels 1-2 require human intervention, Levels 3-5 enable run-time interoperability between two workflow systems allowing one system to invoke workflows on the other, Levels 6-8 typically imply that two workflow tools are based on the same workflow language, or at least an ability to translate workflows expressed in one workflow language to another.

An example of achieving run-time interoperability between scientific workflow systems are presented in [8] in the context of the EU-funded SIMDAT project. There interoperability between both InforSense and Taverna was achieved using a Gateway approach. In this case the native APIs for both workflow engines were exposed as Web/Grid Services. This simple method allowed InforSense workflow, for example, to be treated as remote service invoked from Taverna, and vice versa. This approach went beyond simply being able to invoke the execution of a workflow already stored on either workflow engine. One workflow engine, e.g. InforSense, was allowed to submit a SCUFL workflow definition of the Taverna engine and then to invoke its execution. The approach is not difficult since the InforSense workflow simply treats the SCUFL workflow as an XML file and does not attempt to treat it as a workflow. Similarly, Taverna did not need to understand the internal DPML representation of the InforSense workflow and treated it as an XML file that was passed as input to a remote service call.

An alternative approach, suggested in [7] is based on using an interactive wizard to help a user execute workflows using different workflow systems. Workflow definitions in different workflow languages are stored in a semantically annotated workflow repository. The wizard helps the user locate the best workflow that implements a given task and also to submit it for execution on the appropriate engine. On its own this approach allows a semi-manual co-existence between two workflow systems. The wizard approach does not provide an ability to configure the control flow of the predefined tasks. However, as suggested in [7] it can easily be incorporated into a run-time interoperability framework.

4.2 A Meta-Workflow Approach

As opposed to achieving run-time interoperability between workflow systems, achieving language-based interoperability between such systems is generally not easy in absence of a

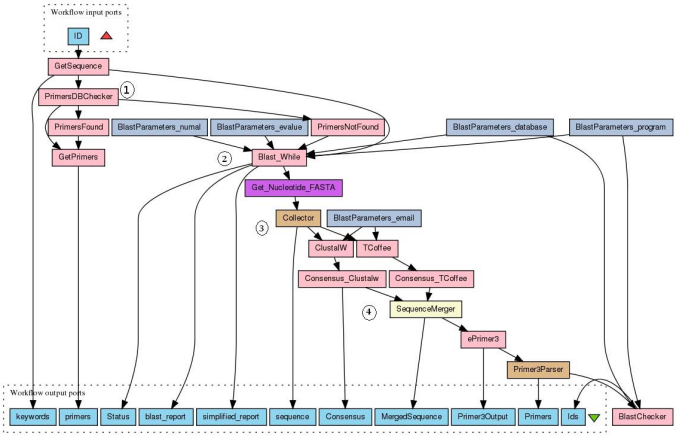

Figure 4: Universal Primer Workflow Taverna. Numbers denote workflow patterns used. (For more details, zoom in original PDF.)

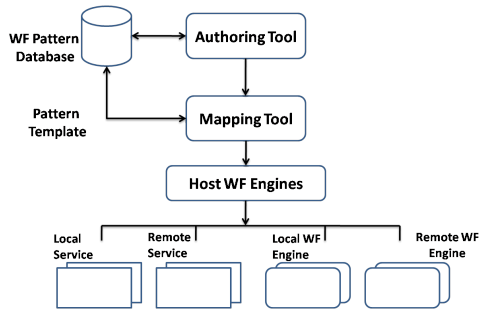

Figure 5: Architecture of the system Tavaxy.

unified standard language. Performing automatic translation from one language to another is also difficult. As discussed in the previous sections different workflow systems are typically built using different assumptions, leading to different execution semantics.

Our approach in this paper is to define a new notion of meta-workflows. A meta-workflow is a high-level description of the steps in workflow, expressed in its own language, but that is not directly executable on any particular workflow system. Instead, the meta-workflow needs to be first translated into fragments of workflows that are expressed in existing workflow languages, with each fragment being submitted to the appropriate workflow engine for execution. The translation could be based on using automatic tools or be semi-automatic as in the case workflow authoring assistant described above. The advantage of the meta-workflow approach, compared to a simple wizard, is that the user can define and user any number of control flow patterns as described below.

4.3 Tavaxy architecture

Figure 5 shows the architecture of Tavaxy, our meta-workflow system built on top of Galaxy and Taverna.

Meta-Workflow Authoring Tool and Language

The meta-workflow authoring tool is web-based and builds on the look and feel of Galaxy. To avoid complications associated with designing yet a new workflow language, a composed workflow is stored in a modified SCUFL format. There are three key modifications. First the modifications allow support for a user defined set of workflow patterns to be included and used in meta-workflow authoring. Second they minimize the use of source and sink nodes in workflow definitions. Parameters for any node, including those representing workflow patterns are explicitly defined in the node itself just as in the Galaxy and InforSense systems. Thirdly, they allows tagging which parts of the meta-workflow should execute on a Taverna engine and which should execute on a Galaxy engine. By default, Taverna sub-workflows will make use of remote service calls and Galaxy sub-workflows will execute on the local infrastructure.

The combination of these variations result in a graphical notation that looks more similar to a traditional flow chart. To avoid loose execution semantics and un-controlled cyclic loops problems we restrict the definition of workflow patterns used to be block structured; loops, conditionals and parallel forks have explicit start and end points that must be defined.

Workflow Pattern Database

The workflow pattern database stores the definition and implementation of the control flow workflow patterns used in Tavaxy. All patterns need to be block structured allowing encapsulation of sub workflows within them. For example no cyclic loops are allowed in an *iteration* pattern. A *switch* and *fork* node must have a matching *merge* pattern, or *end* nodes. Users may update the database with new patterns provided they follow the restrictions and add the appropriate implementation strategies for each workflow engine.

Host Workflow Engines

We have two workflow engines: Primary and Secondary. The primary is the main engine that coordinates the user workflow and can invoke the secondary engine to execute sub-workflows. This invocation of the secondary engine is enabled by the *Meta-workflow Mapper* introduced below. The Taverna engine is already a standalone application and can be invoked from the shell given workflow as input. However, this is not the case with Galaxy. Therefore, we performed some software engineering effort on the Galaxy system, where we separated the user interface, the workflow database, and the workflow engine from each other. After this modification, the primary engine can be either the one of Taverna or the one of Galaxy. Once one of them is chosen as a primary engine, the other serves as the secondary engine. Our current implementation uses the one of Galaxy to make use of the local computational resources.

Meta-workflow Mapper

Based on the parameters set by the user the Meta-workflow mapper performs the following set of tasks to generate executable workflows:

- Parses the SCUFL format and checks its syntax.
- Chooses and generates an appropriate implementation for each workflow fragment and workflow pattern in terms of the primary workflow system.
- Generates calls to the secondary workflow system and encapsulates them in an appropriate manner as sub-workflows in the primary workflow system.

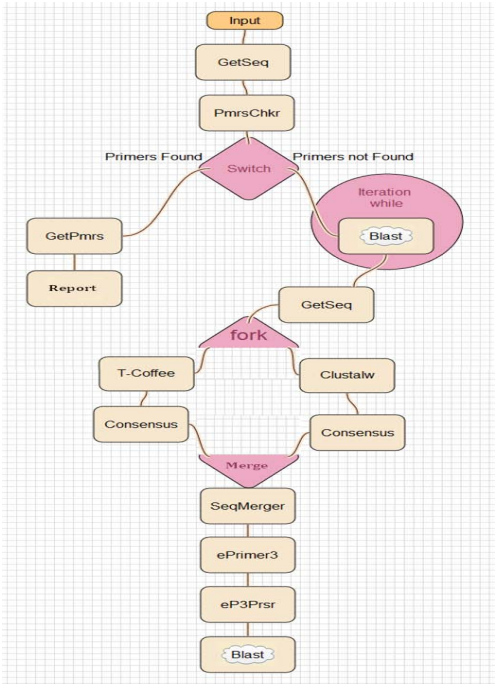

Figure 6: Universal Primer Workflow in Tavaxy. The diamond shape corresponds to *switch-case* or *if-else* construct. The circle encapsulates steps that should be iterated. The *fork* and *merge* operations are given special shapes. The cloud around a tool name indicates remote invocation. (For more details, zoom in original PDF.)

- It enables mapping of input and output ports between sub-workflows on primary and secondary engines.

Implementation of the patterns

In Tavaxy, there are two versions for the implementation of each pattern: One that runs on Taverna and one that runs on Galaxy. The Taverna implementation is based on invoking remote web services, the Galaxy based pattern is invoked to execute on the local infrastructure. Because the patterns *switch* and *iteration* are not supported by Galaxy, we wrote special scripts that realize these patterns. The Galaxy engine then handles these scripts as tasks. In brief, these scripts work as follows:

1. The script for *iteration* pattern takes as input the task (or sub-workflow) that iterates, its parameters, the termination criteria, and information about feedback data. The output specified by the user is passed to the next task upon termination.

2. The script for *switch* pattern takes as input 1) the multi-choice condition, and 2) the data to be passed to the next tasks. It then executes a synchronous fork pattern such that a dummy data is passed to the branch violating the multi-choice condition.

5. TAVAXY IN ACTION

5.1 Universal Primer Workflow Realization

Figure 6 shows the universal primer example implemented in the web-based workflow editor of Tavaxy. The patterns mentioned above are implemented in Tavaxy as integral constructs. The *iteration*, *switch*, *fork*, and *merge* patterns are given special shapes referring to their execution semantics. In the universal primer example, the user runs all the programs locally except for BLAST, by setting the appropriate options. In the Figure, a cloud background indicates that this task runs remotely. Similarly, if the primer database and the attached search engine are remote, Tavaxy by default uses a wrapper to run this step. The *iteration* pattern includes a remote call to BLAST. Hence, all the *iteration* pattern is encapsulated as sub-workflow and passed to the Taverna engine. If the BLAST were called locally, then the local script implementing this pattern is invoked and run on local machines. The *merge* pattern passes the output of T-coffee and ClustalW to the step where a tool is invoked to combine both output sequence lists in one list. (Here two input ports are assigned to the task associated with the merge operation.)

5.2 Interplay between Galaxy and Taverna

Figure 7 shows a part of the SCUFL format of Tavaxy specifying our universal primer workflow. In this example, Galaxy is the primary host engine and executes sub-workflows on Taverna. In the figure, we highlight the sub-workflow involving iteration and shows how it is exported to Taverna. The idea is to call Taverna as a program and passes the sub-workflow as an argument with the input data. The output data are retrieved and passed to the next step in the workflow.

Figure 8 shows a part of the tavaxy SCUFL format assuming Taverna is the primary host engine. The workflow implements the universal primer example. Here, we highlight the part of the workflow that runs ClustalW on Galaxy on the local machines. The idea is to call Galaxy from Taverna through web-service interface. In this invocation, we specify the program Galaxy should execute and the data that is passed to the sub-workflow.

6. CONCLUSIONS

In this paper we introduced Tavaxy, a system that achieves high-level interoperability between two popular tools, Galaxy and Taverna, in the bioinformatics domain. The approach is based on meta-workflows and the use of patterns. We demonstrated the usefulness of this approach via the design of universal primers. The set of patterns we introduced are implemented in the first prototype of Tavaxy. Currently, we are working on improving the prototype we have and on increasing the set of patterns with more higher level ones specific to the sequence analysis domain. These patterns will further ease the composition of sequence analysis workflows and provides efficient solutions to recurrent tasks.

7. REFERENCES

[1] M. Abouelhoda, R. Giegerich, B. Behzadi, and JM. Steyaert. Alignment of minisatellite maps based on run length encoding scheme. *J. Bioinformatics and Computational Biology*, 7(2):287–308, 2009.

[2] M. Abouelhoda, S. Kurtz, and E. Ohlebusch. CoCoNUT: An efficient system for the comparison and analysis of genomes. *BMC Bioinformatics*, 9:476, 2008.

```
<?xml version="1.0" encoding="UTF-8" ?>
 <s:scufl xmlns:s="http://.../xscufl/.." version="0.2">
       :
1.  <s:processor name="switch_getseq" id="Switch" />
       :
2.  <s:processor name="iterated_tav_blast_blast"
      id="taverna2">blast.t2flow</s:processor>
       :
3.  <s:processor name="getseq2" id="passvalue" />
       :
4.  <s:link source="switch_getseq:branch_false"
   sink="iterated_tav_blast:sequence"
   ignore="false" />
       :
5.  <s:link source="iterated_tav_blast:blast_report"
      sink="getseq2:input" ignore="false"/>
       :
 </s:scufl>
```

Figure 7: The primary host engine Galaxy calls Taverna sub-workflows: Here we highlight the iteration step involving BLAST. As shown in Figure 6 of Tavaxy, the input of this step is the data coming from `GetSeq` task through the `Switch` pattern. The output is passed to the `GetSeq` task (We call it here `getseq2` for ease of reading). The previous task to the iteration is in line 1 and next task is in line 3. The iteration is processed in line 3 where Taverna is handled as a program and is called with the sub-workflow stored in the file `blast.t2flow` as an argument. The input and output are defined by the `link` fields (lines 4 and 5). This file `blast.t2flow` (not shown) is stored in Taverna format (specifically t2flow which supporting feedback).

```
<?xml version="1.0" encoding="UTF-8" ?>
<s:scufl xmlns:s="http://.../xscufl/.." version="0.2">
           :
1. <s:processor name="GetSequence">
           :
2. <s:processor name="clustalw_workflow" boring="true">
3. <s:stringconstant>clustalwx.xml</s:stringconstant>
         :
4.   <s:processor name="clustalw">
5.    <s:description>ExecuteWFInGalaxy</s:description>
6.    <s:arbitrarywsdl>
7.     <s:wsdl>http://localhost/RunGalaxy.wsdl</s:wsdl>
8.     <s:operation>ExecuteWorfklow</s:operation>
9.   </s:arbitrarywsdl>
           :
10. <s:processor name="Consensus"> </s:processor>
           :
11. <s:link source="clustalw_workflow:value"
        sink="clustalw:workflowPath"/>
12. <s:link source="GetSequence:outputText"
        sink="clustalw:input1"/>
13. <s:link source="clustalw:output1"
        sink="Consensus:AlginmentOutput"/>
</s:scufl>
```

Figure 8: The primary host engine Taverna calls Galaxy sub-workflows: Here we highlight the step involving the call of ClustalW on the local infrastructure. The previous task is `GetSequence` (line 1) and the next task is `Consensus` (line 10). We assume that there is a local Web Service for Galaxy which take the sub-workflow to be executed (lines 4-9) as input. This sub-workflow is passed to it through its path (lines 2-3, 11) using the `stringconstant` processor of Taverna. The input and output to this workflow are in lines 12 and 13.

[3] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. A basic local alignment search tool. *J. Molecular Biology*, 215:403–410, 1990.

[4] S.F. Altschul, T. L. Madden, A. A. Schäffer, and et al.˙Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, 1997.

[5] J. Bradley, C. Brown, B. Carpenter, and et al.˙The omii software distribution. In *All Hands Meeting*, pages 748–753. Humana Press, 2006.

[6] V. Curcin and M. Ghanem. Scientific workflow systems - can one size fit all? In *Proceedings of CIBEC*. IEEE, 2008.

[7] M. Ghanem, N. Azam, and M. Boniface. Workflow Interoperability in Grid-based Systems. In *Cracow Grid Workshop 2006*, September 2006.

[8] M. Ghanem, N. Azam, M. Boniface, and J. Ferris. Grid-enabled workflows for industrial product design. In *E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, page 96, 2006.

[9] M. Ghanem, V. Curcin, P. Wendel, and Y. Guo. Building and using analytical workflows in discovery net. In *Data mining on the Grid*. John Wiley and Sons, 2008.

[10] B. Giardine, C. Riemer, R.C. Hardison, and et al. Galaxy: A platform for interactive large-scale genome analysis. *Genome Research*, 15(10):1451–5, 2005.

[11] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, and et al.˙Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34:W729–32, 2006.

[12] B. Ludäscher, I. Altintas, C. Berkley, Higgins. D., and et al.˙Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[13] C. Notredame, D.G. Higgins, and J. Heringa. T-Coffee: A novel method for fast and accurate multiple sequence alignment. *J. Molecular Biology*, 302(1):205–17, 2000.

[14] T. Oinn, M. Addis, J. Ferris, D. Marvin, and et al. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–54, 2004.

[15] P. Rice, I. Longden, and A. Bleasby. EMBOSS: the european molecular biology open software suite. *Trends in Genetics*, 16(6):276–7, 2000.

[16] A. Rowe, D. Kalaitzopoulos, M. Osmond, M. Ghanem, and Y. Guo. The discovery net system for high throughput bioinformatics. *Bioinformatics*, 19(90001):225i–231, 2003.

[17] I. Taylor, M. Shields, I. Wang, and A. Harrison. The Triana Workflow Environment: Architecture and Applications. In *Workflows for e-Science*, pages 320–339. Springer, 2007.

[18] Ian Taylor, Matthew Shields, Ian Wang, and Andrew Harrison. Visual Grid Workflow in Triana. *J. Grid Computing*, 3(3-4):153–169, 2005.

[19] W.M.P. van der Aalst, A.H.M. Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(3):5–51, 2003.

[20] WfMC. Workflow Management Coalition Workflow Standard - Interoperability Abstract Specification. Document Number WFMC-TC-1012. Version 1. Technical report, www.wfmc.org.