Building Scientific Workflow with Taverna and BPEL: a Comparative Study in caGrid

Wei Tan[1] , Paolo Missier[2], Ravi Madduri[3] and Ian Foster[1]

1 Computation Institute, University of Chicago and Argonne National Laboratory, Chicago, IL, USA
2 School of Computer Science, University of Manchester, Manchester, U.K
3 Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL,
wtan@mcs.anl.gov, pmissier@cs.man.ac.uk, madduri@mcs.anl.gov, foster@mcs.anl.gov

Abstract. With the emergence of "service oriented science," the need arises to orchestrate various services to facilitate scientific investigation -- that is, to create "science workflows." In this paper we summarize our findings in providing a workflow solution for the caGrid service-based grid infrastructure. We choose BPEL and Taverna as candidate solutions, and compare their usability in the full lifecycle of a scientific workflow, including service discovery, service composition, workflow execution, and workflow result analysis. We determine that BPEL offers a comprehensive set of primitives for modeling processes of all flavors, while Taverna provides a more compact set of primitives and a functional programming model that eases data flow modeling. We hope that our analysis not only helps researchers choose a tool that meets their needs, but also provides some insight on how a workflow language and tool can fulfill the requirement of scientists.

1 Introduction

More and more data and computation resources used by the scientific community are built on a service-oriented architecture (SOA) [1]. Given the proliferation of web services, service-oriented science [2] is becoming an emerging paradigm in facilitating scientific investigation, and scientific workflow has become an important approach to orchestrate various services [3]. For example, caGrid [4] is the service-based grid software infrastructure that underpins the cancer Biomedical Informatics Grid. This infrastructure, based on the Globus Toolkit [5], enables the sharing of information and analytical resources (via grid services). By this means it helps domain scientists to easily contribute to and leverage caBIG resources, accelerating biomedical research in a multi-institutional environment.

There are already many languages, tools and systems exist for scientific workflow [6]. Through a comprehensive survey on existing workflow tools [7], the caGrid team decided to choose Taverna [8] and BPEL [9] as candidate workflow solutions: as Taverna is representative of many scientific workflow systems, while BPEL is an well-accepted standard in business domain and is gaining momentum in science.

BPEL: WS-BPEL (Web Service-Business Process Execution Language, or BPEL for short) is a meta-model and an XML-based specification for describing the behavior of a business process that is composed of Web services and also exposed as a Web service. Although originally designed for business workflows, BPEL has also attracted attention from the scientific community because of its support for the SOA paradigm. BPEL can be seen as a good representative of those languages originated from business domain and are now been adopted by the scientific community.

Taverna: Developed in the UK by the myGrid consortium (http://www.mygrid.org.uk), Taverna is an open-source workbench for the design and execution of scientific workflows. Aimed primarily at the life sciences community, its main goal is to make the design and execution of workflows accessible to bioinformaticians who are not necessarily experts in web services and programming. A Taverna workflow is a linked graph of *processors*, which represent Web services or other executable components, each of which transforms a set of data inputs into a set of data outputs. These workflows are represented in the Scufl language (using an XML syntax), and executed according to a functional programming model [10]. The data-driven model is briefly presented in Section 4. Taverna also provides a plug-in architecture so that additional applications, such as secure Web Services, can be

populated to it. The caGrid plug-in recently implemented by members of our group [11] is an example.
The design and implementation of workflow systems for scientific purposes has been a subject of considerable research [6]. The goals of this paper are to communicate practical experiences based on our work in the caGrid project. In this analysis, we consider the entire scientific workflow lifecycle, from service discovery to service composition, workflow execution, and workflow result analysis. The analysis is based on our understanding of caGrid's requirements for a workflow language and tooling, but we believe is also applicable to other areas in data intensive and exploratory science. We hope that our work not only helps researchers choose a tool that meets their needs, but also provides some insight on how a workflow language and tool can fulfill the requirement of scientists.
In our comparison of BPEL and Taverna we consider not only the two workflow languages but also their associated tooling. This is because scientific workflow users are generally scientists that have expertise in their specific domain (biology, physics, astronomy, etc.) but understandably limited knowledge of IT technology and thus require easy-to-use tooling. In the remaining of this paper, the term Taverna is used to refer to both the Scufl workflow language that Taverna uses and the Taverna tool, while BPEL to both the language and the open source tools supporting it.
In the remainder of this paper we first present a caGrid use case and then examine the lifecycle and features of scientific workflows. Then, we compare Taverna and BPEL from three perspectives: service discovery, service composition and workflow execution, and workflow results analysis. Finally, we draw conclusions.

2 A caGrid use case

We present a caGrid use case that relates to the querying of semantic data in cancer research. The use of a standardized metamodel and semantic annotation to enable the formal description and harmonized use of data is a primary feature that caGrid moves beyond the basic grid infrastructure.

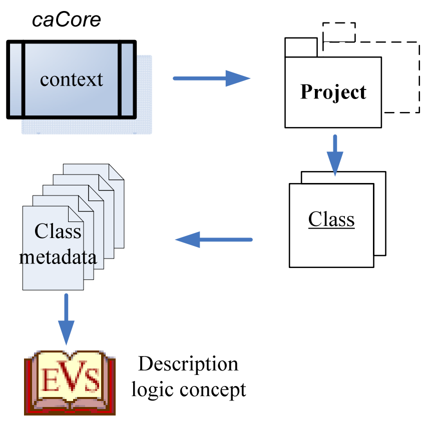


Fig. 1. The caGrid use case used in the paper

In the use case illustrated in Fig. 1, a user wants to query description logic concepts that relate to a particular context, namely "caCore." First, the user queries all projects related to context "caCore"; second, they find UML classes in each project; third, they use project and UML class information to query the semantic metadata; and finally, they retrieve the concept code.

3 The lifecycle and features of scientific workflows

To define the scope of the comparative study, we first discuss the lifecycle of a scientific workflow.

This lifecycle involves four stages: discover relevant data/analytical services, compose these services into a workflow, execute workflow, and analyze the results (see Fig. 2).

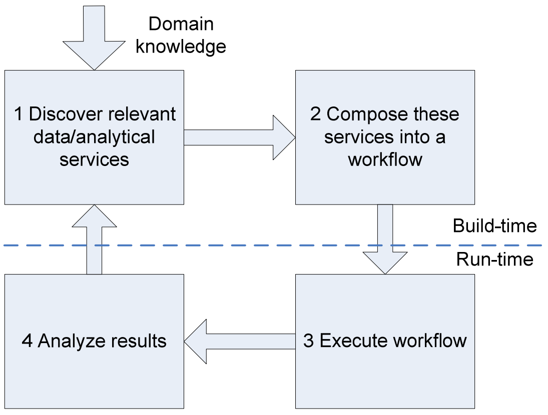

Domain knowledge
1 Discover relevant data/analytical services
2 Compose these services into a workflow
Build-time
Run-time
4 Analyze results
3 Execute workflow

Fig. 2. Lifecycle of a scientific workflow

1. *Discover relevant data/analytical services*. Data/analytical services are developed, owned and maintained by different institutions, organizations, etc. Usually the URL of these services is not well-known. Moreover, the scientific community is too autonomous to share a common terminology, so domain knowledge is needed to get the exact semantics of the services whose syntax is already known.
2. *Compose these services into a workflow*. After the individual services are found, the next step is to compose them into a workflow. This step involves the addition of data and control dependencies between services; it may also involve data transformations, if the output of one service is not in the format required for another service's input. Composition can be performed by using a general-purpose programming language like Java, a proprietary script like Scufl, or a GUI. (A GUI usually depends on a script format for persistent storage.)
3. *Execute workflow*. A workflow definition is sent to an engine for execution. The engine invokes the services in the pre-defined order, providing input and retrieving output of services during the ordered execution.
4. *Analyze results*. Scientific workflow is for the purpose of exploratory research, and therefore, the intermediate results generated by component services, as well as the final results yield by the workflow, are of great value and deserve to be analyzed carefully. Scientific researches are usually undertaken in an iterative manner so the analysis results often initiate another round of workflow modeling/execution.

We summarize some features of scientific workflows in caGrid, and these features can also be seen as challenges encountered when providing a scientific workflow solution in a more general sense (the number of the bullets represents their position in the lifecycle shown in Fig. 2).

- (1) Resources are highly distributed. Compared to business domain, users in scientific domain usually use services owned by other organizations, like data storage, high-performance computing, etc.
- (2) Data-flow oriented. Data is considered to be the first-class citizen in scientific workflows, because scientific workflows are mostly pipelines of parallel data processing. In a data flow, tasks and links represent data processing and data transport, respectively; parallel execution of independent tasks is desired to be modeled for free -- tasks can execute once their input are ready.
- (2&3) Large scale. Scientific workflows often contain many tasks, involve large data sets, and require intensive computation. The modeling tool should make it easy to model such complex workflows.
- (4) Data analysis and provenance is an important step and the workflow execution can be in an

iterative manner.

In the rest of this paper we highlight some of the differences between the BPEL and Taverna, from the point of view of their impact on the users' experience in the lifecycle of scientific workflows. The discussion is organized according to the lifecycle model of Fig. 2.

4 Support for service discovery

As suggested in Fig. 2, a user's first task involves finding appropriate services that can be composed into a workflow. In a Grid setting, these services are virtualizations of data storage, computation capability or other resources. Service endpoints are not naturally known to users, either because users are not familiar with the service itself, or because the service deployment may have changed in time. Support for service discovery is therefore needed.

Taverna offers two levels of support for this. Firstly, it is often the case that a website is known to host one or more services. In these cases, a *scavenger* meta-service can be used to locate endpoints within the site which correspond to valid WSDL service definitions. The WSDL is automatically analyzed and a description of the service is added to the Taverna workbench's library, ready to be used in workflows.

The Taverna plug-in framework simplifies the creation of new scavengers that may offer advanced service discovery features, making it easy for users to enrich the collection of available services according to their own needs. As part of the caGrid project, for example, we have developed a *caGrid scavenger* that supports semantic/metadata based query to caGrid services. Users can use multiple query criteria to get the list of desired services. For example, they can query the services that are developed by *Ohio State University*, whose names are *CaDSRService*, and with the class *Project* as output. Through this query we find the matching service for the first step in the use case shown in Fig. 2 – use context to get related projects.

As a second, more general-purpose option, a semantic discovery facility called *Feta* [12] also offered natively as part of the Taverna distribution. Feta includes a semantic service registry that maintains annotated description of services, and can be searched using terms from a publicly available ontology. The annotations describe (1) the task performed by a service, for example *bioinformatics task*, (2) the type of resource used by the service, e.g. *bioinformatics data resource*, (3) the types of input data it accepts and of output data it produces (*protein structure*, for example), and more. The terms used in the annotations belong to the myGrid ontology [13], a controlled ontology of terms for the bioinformatics domain.

These discovery facilities stand in contrast with the lack of analogous integrated tools for BPEL design environments. To the best of our knowledge, no open-source BPEL tool is available that works with a service query component in an integrated way. This seems at odds with one of the stated motivations for BPEL, namely to integrate widely distributed and heterogeneous services.

5 Service composition & workflow execution

The second and third phase of the lifecycle involves composing the discovered services into complete workflows and executing them. In this section we focus on the modeling style, the definition of data, the iteration strategies adopted by BPEL and Taverna, respectively, and their influence to the run-time engine.

5.1 Data-driven vs. control-driven modeling

When modeling a workflow, users are confronted with the choice among the different modeling paradigms offered by Taverna and BPEL. While the former follows a pure data flow approach to workflow modeling and execution, the latter exposes a fundamentally procedural language.

In a data flow model, the workflow is described as a graph where nodes represent processors that can be executed on input provided along the incoming arcs, and whose output is forwarded to other processors through outgoing arcs. In this model, the order in which the processors are executed is determined primarily by the order in which the data appears on the various inputs. Any processor for which the input data is available can be scheduled for execution.

In Taverna, scheduling is simple: processors are executed as soon as possible, in a greedy fashion as long as a new execution thread can be started (a limit on the number of threads can be defined on the scheduler). This means, in particular, that parallelization of processor execution is managed by the scheduler, based on the available data, without the need for explicit user directives. Also, the order of execution of two processors that have no data dependencies amongst each other may be different for different executions of the same workflow, even on the same input, due to the possible variations in execution speeds of some of the other processors.

In contrast, a procedural workflow language like BPEL includes the explicit definition of the control flow that determines the order of execution of the processors. In particular, parallel execution of independent processors must be specified explicitly.

A comprehensive analysis on the differences and relative merits of control-driven and data-driven execution is beyond the scope of this paper. A more in-depth discussion can be found in [14]. In the rest of this section we focus on the specific differences between Taverna and BPEL, summarized in Table 1.

Table 1. Comparison of BPEL and Taverna (Scufl) w.r.t. control/data-flow

	BPEL	Taverna (Scufl)
Activities in model	Basic and structure activities	Processors as data processing units with in/output ports
Semantics of links	Transfer of control	Transfer of data
Data definition	Explicitly defined (global variables)	Implicit defined (processor's input/output)
Data initialization	Complex data type need to be explicitly initialized	Automatically
Control logic	Full-fledged: sequence, conditional, parallel, event-triggered, etc	Limited: sequential, parallel and conditional
Parallel execution	Defined in <flow> or <ForEach>	By default

5.2 Implicit vs. explicit definition of data

Complementary to the control model described in the previous section is the data specification model. In Taverna, processors have input and output *ports* with an associated data type, and data travels from the output port of a processor to the input for of one or more downstream processors. No other data structure specification is needed besides the port types, and interaction among processors is defined entirely by the arcs in the dataflow graph.

In contrast, BPEL requires the explicit definition of variables to hold data structures that are meant to be shared amongst activities; furthermore, each activity can be specified as either a producer or a consumer for values associated to a variable. Although BPEL's requirement for explicit data definition takes additional effort, it also brings about flexibility. For example, in BPEL you can easily define a data that controls the overall flow but is not the input/output of any activities, but in Taverna you have to add a processor to hold this data (as either input or output).

In BPEL, variables of complex type, must also be initialized prior to their first use (i.e., by means of the <*copy*>syntactic construct – see Section 8.4.2 of the WS-BPEL Specification in [9]). In contrast, Taverna provides a special built-in processor, called an *XML splitter*, which automatically pulls apart a complex XML message defined in a WSDL interface so that its components can be easily accessed by other user-defined processors. An example of its use is provided in the next sub-section.

5.3 Implicit vs. explicit iteration on data

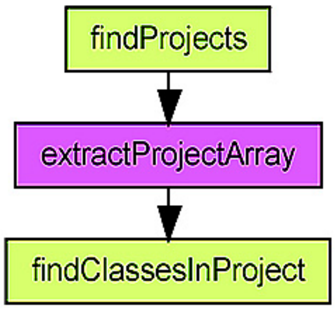
Each port in a Taverna processor has a type, which is either a simple type value (i.e., a string, a number) or a list, possibly nested, of simple type values. As part of normal processing, it may be the case that an input port receives a value of a type that does not correspond exactly to its declared type. A processor that outputs a value of type "list of strings," for example, can legally be connected to a processor with an input port of type "string." Taverna interprets this type mismatch as an indication that the destination processor must be invoked repeatedly, once for each element of the input list. This behavior is consistent with Taverna's functional programming model, whereby the application of a function *f* with a formal argument of type *t*, to an actual parameter *x* of type list(*t*), is interpreted as (map *f x*).

In practice, each mismatch in the *nesting level* of the input value with respect to the declared type on the port, triggers an automatic, implicit iteration of the processor invocation over each element in the input (note, however, that any type mismatch other than in the nesting level, for instance providing a string instead of a number, is an error).

In general, when mismatches appear simultaneously on multiple input ports, Taverna performs either a cross-product (i.e., a Cartesian product) or a dot product (if the cardinalities of the two lists are the same) involving the elements of each of the unexpected lists. Users may explicitly choose which of these two iteration strategies is appropriate for each processor. In the current implementation, iterations may be executed concurrently if sufficient resources are available to support parallel execution threads.

The implicit iteration feature is commonly used in Taverna scientific workflows. The implication, from the user's perspective, is that the design of a service can be simplified by assuming that it will manage individual data items, while the execution engine takes care of managing input collections.

See an example from the caDSR (Cancer Data Standards Repository) service that access and generate the information related to caGrid standard metadata. caDSR has two operations: *findProjects* and *findClassesInProject*. Operation *findProjects* returns a set of projects (i.e., an array *Project []*); *findClassesInProject* receives an instance of data type *Project* and find all the UML classes in this project. Fig. 3 illustrates the Taverna and BPEL presentation of how to connect them into a workflow. The left and right parts are Taverna and BPEL representation, respectively. In the left part, the output of *findProjects* is put to an xml-splitter which extracts out the project array, and sends it to *findClassesInProject*. In the right part, since BPEL does not have an implicit iteration mechanism, a <*ForEach*> construct is added and configured to iterate on project array. After each invocation of *findClassesInProject*, result data need to be collected and merged into the final results set.



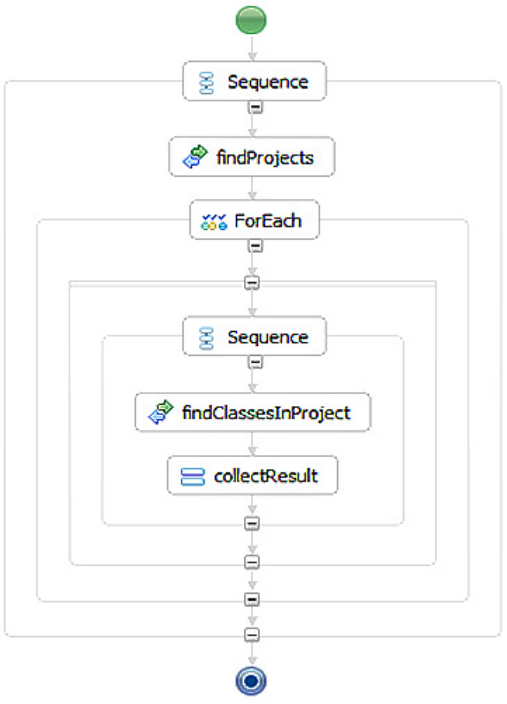


Fig. 3. Implicit iteration: case 1

Fig. 4 shows another example. Operation *findProjects* yields an array of projects, and *findClassesInProject* yields an array of UML classes. In Taverna you simply combine two arrays in an xml-splitter *assignInput*, configuring the iteration strategy to Cartesian-product, and the workflow will iterate on all combinations of the item project and class. To achieve this in BPEL, two nested <*ForEach*> is needed.

From these two examples one can see that, BPEL handles the iteration like an imperative programming language, a <*ForEach*> construct and the iteration method (a counter, an array or an expression) is to be configured. It is verbose (suppose you have more than 2 arrays for cross-product or a combination of dot and cross products) and exposes too many implementation details to the end users (and thus error-prone). Taverna deals with this issue in a straightforward way -- its implicit iteration framework requires (in the simplest cases) no additional configuration, and the user simply connects an output containing a collection of items into an input which consumes a single item of the same type. This leaves the complexity to the workflow engine instead of the users.

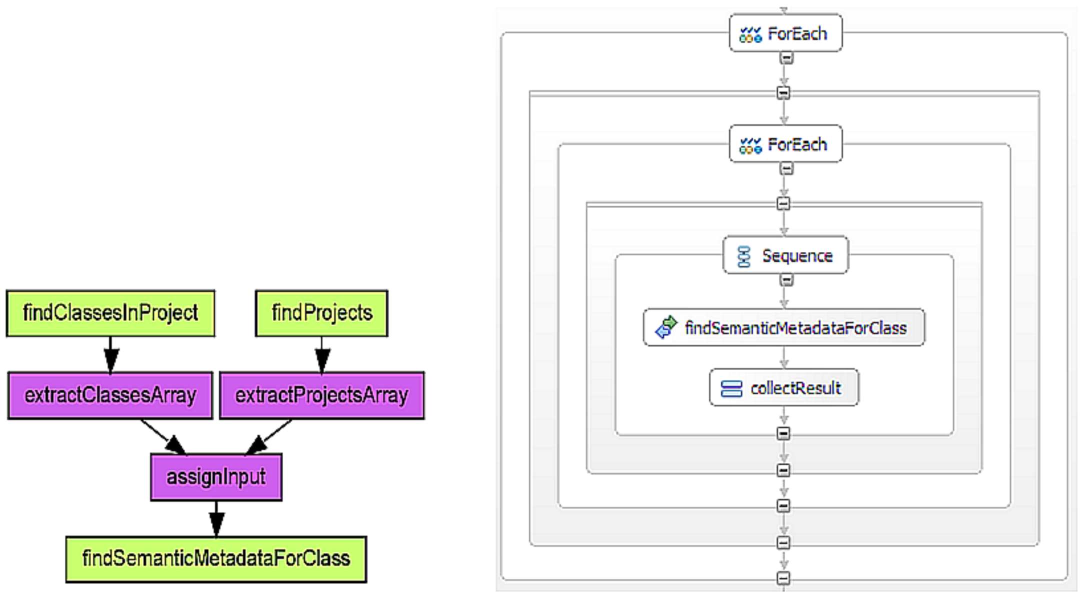


Fig. 4. Implicit iteration: case 2

Again, as an imperative language, BPEL offers more flexibility in handling advanced iteration strategies. For an example, BPEL can handle this issue: an activity receives two lists of inputs, needs a special kind of dot-product iteration over them, with a special "correlation" mechanism (like, classes and projects with the same developer should be combined.)

For space limitation, in Fig. 5 we only show the completed Taverna workflow for the caGrid use case in Fig. 1. There are four caGrid processors (*findProject*, *findClassesInProject*, *findSemanticMetadataForClass*, and *searchDescLogicConcept*) that represent caGrid services, and more "shim" processors for data transformation between caGrid processors.

6 Workflow result analysis

The final phase of the workflow lifecycle, namely analysis of the results, is increasingly perceived as of great importance within the e-science community [15]. The *provenance* of a piece of data produced by an arbitrary process is a complete account of how that piece of data was computed, starting from user input and taking into account intermediate results produced by the processors involved in the computation. Business and scientific workflows may differ in both their requirements and their ability to track data provenance, in particular with regards to the *precision* of provenance information.

Precision, in this case, denotes the levels of detail at which provenance can be traced, and depends on the unit of information that the workflow engine can observe during execution. When dealing with Web Services, both in BPEL and Taverna, the atomic unit of information that flows through a processor is an XML document, for instance "purchase order" for a business process, or an XML-formatted description of a protein in the case of a scientific process. The black-box nature of the Web Services that produce and consume these documents limits the ability to track its individual elements. For instance, consider a service that takes a purchase requisition request document as input, and returns a purchase order document. While it is likely that specific elements within the purchase order depend on only some of the input document elements, this fine-grained dependency is hidden within the service logic: from the point of view of provenance, the service is a black box, because the nature of the data transformation they implement is not exposed through the WSDL interface. Thus, the only data dependency that can be safely used in provenance tracking is that the entire purchase order depends on the entire purchase requisition request.

Of course, when Web Services expose message types that are simple types, for example simple strings, then provenance can be traced at the level of these types. Business applications, however, are more likely to expose complex types that are part of articulated information exchanges. Thus, in general the black-box nature of the service limits the degree of precision with which provenance of the output can be tracked: the granularity of traceable provenance is that of entire XML documents, rather than that of their composing elements.

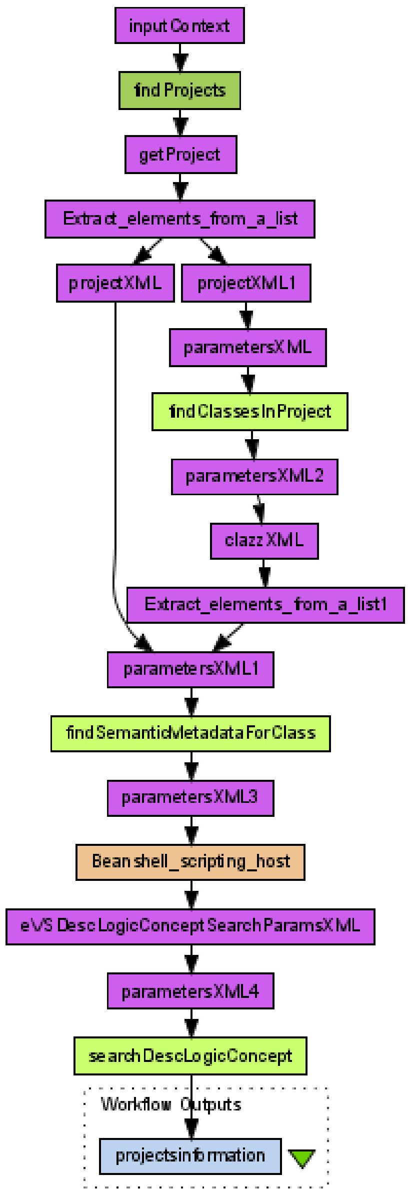


Fig. 5. Completed Taverna workflow for the caGrid use case in Fig. 1

As we mentioned, this problem affects both BPEL and Taverna. Unlike BPEL, however, Taverna is not limited to using processors that are implemented as Web Services; processors types include local Java classes, as well as *beanshells*, or small interpreted Java programs. This makes it quite natural for Taverna workflows to handle simple types, such as strings, as well as collections of elements of these types that often represent sets of scientific data products. In this case it is important to be able to track the provenance of each of these products individually. Our caGrid use case, for example, involves a one-to-many association between Projects and their UML classes, which is then used to retrieve semantic concepts associated to project classes. For provenance information to be useful, here we cannot simply state that "the collection of the concepts depends on the collection of input projects," because this is as trivially true as it is uninteresting. Instead, we must be able to determine that the presence of a specific concept in the output is due to a specific project being present in the original input.

An important example of this fine-grained data manipulation is the "packing" and "unpacking" of complex XML data, something that can be achieved automatically using XML splitters, as mentioned in Sec.5.2 and 5.3. In some cases, this may enable provenance tracking through the internal element of XML documents, for instance it may be possible to trace the *originator* element of a purchase order back to some specific workflow input, at a stage in the process prior to its use as part of the order.

In our preliminary experiments on provenance tracking in Taverna, performed within the myGrid team, we have been able to achieve high precision in many practical cases, namely when simple values are composed into collections or into complex XML messages in a way that is visible to the engine, i.e., by means of dedicated packing and unpacking processors.

As a corollary to this investigation, we have also been arguing that processors that map entire collections to new collections (i.e., without any iteration being exposed to the workflow engine) should be annotated, where possible, with an indication of properties of the mapping that help provenance tracking. A detailed discussion of the promises and limitations of this idea can be found in [16].

Other approaches to tracking provenance through Web Services involve the explicit semantic annotation of the involved services. This *semantic provenance overlays* approach is really complementary to the problem discussed in this section, and early experiment done on Taverna show promising results [17].

7 Conclusion and future work

From our experience in using both Taverna and BPEL as the candidate solutions for caGrid workflow, we have the following conclusions:

1. Taverna provides a compact set of primitives that eases the modeling of data flow. This functional-programming manner allows users to tell "what to do" instead of "how to achieve it."
2. BPEL offers a comprehensive set of primitives to model processes of all flavors (control-flow oriented, data-flow oriented, event driven, etc), with full feature (process logic, data manipulation, event and message processing, fault handling, etc). BPEL is also flexible enough to handle complex processing logic, although a little bit verbose in modeling basic data flow.
3. As a tool-suite, Taverna provides better support in the whole lifecycle of scientific workflows, including service discovery and results analysis, than the existing open-source BPEL tools do.

We do not mean to indicate that Taverna is better than BPEL, or vice versa. We would rather say that Taverna better fits the requirement of modeling a data flow, and the open source community has provided a handy workbench that consists of the modeling and the execution tools. We also acknowledge nice features of BPEL engines. For example, BPEL engines typically run inside application servers and are with persistent state storage, which offer more reliability and scalability. This is important for those long-running and computation-intensive workflows. For now the Taverna engine does not provide these capabilities.

At the same time, we suggest a promising *multi-stage modeling approach* in adapting BPEL to scientific workflow, leveraging its capability and retaining the simplicity. That is, the scientists use a model which is intuitive to them, and transform this model into a standard BPEL model automatically, through a macro-expansion procedure. This BPEL model can be orchestrated by a BPEL-compliant engine. Actually this approach has already been adopted by existing research efforts [18, 19]. In future, we also plan to investigate the possibility to provide a BPEL-centric tool set where discovery and result analysis tools are included.

Acknowledgement

We thank Taverna team, especially Mr. Stian Soiland-Reyes at University of Manchester, for the great help in using Taverna and developing plug-ins for it. We also thank the constructive comments from anonymous reviewers. This project has been funded in part with Federal funds from the National Cancer Institute, National Institutes of Health, under Contract No. N01-CO-12400.

Reference

1. Krishnan, S. and K. Bhatia. SOAs for Scientific Applications: Experiences and Challenges. Proc. IEEE International Conference on e-Science and Grid Computing. 2007.
2. Foster, I., Service-Oriented Science. Science, 2005. **308**(5723): p. 814-817.
3. Tan, W., et al., Workflow in a Service Oriented Cyberinfrastructure Environment, in Cyberinfrastructure Technologies and Applications, J. Cao, Editor. 2008, Nova Science Publishers.
4. Saltz, J., et al., caGrid: design and implementation of the core architecture of the cancer biomedical informatics grid. Bioinformatics, 2006. **22**(15): p. 1910-1916.
5. Foster, I., Globus Toolkit Version 4: Software for Service-Oriented Systems. Journal of Computer Science and Technology, 2006. **21**(4): p. 513-520.
6. Taylor, I.J., et al., Workflows for e-Science: Scientific Workflows for Grids. 2007: Springer-Verlag.
7. ICR Workflow Working Group,"Tool Reviews", http://gforge.nci.nih.gov/docman/view.php/332/7509/icr_workflow_tool_review_2007.doc, 2007.
8. Oinn, T., et al., Taverna/myGrid: aligning a workflow system with the life sciences community, in Workflows for E-science: Scientific Workflows for Grids, I.J. Taylor, et al., Editors. 2007, Springer: Guildford. p. 300–319.
9. OASIS (2007), Web Services Business Process Execution Language Version 2.0., http://docs.oasis-open.org/wsbpel/2.0/CS01/ wsbpel-v2.0-CS01.html, 2007.
10. Turi, D., et al. Taverna Workflows: Syntax and Semantics. Proc. 3rd e-Science Conference. 2007. Bangalore, India.
11. Wei Tan, et al. Orchestrating caGrid Services in Taverna. Proc. IEEE International Conference on Web Services (ICWS 2008). 2008. Beijing, China.
12. Lord, P., et al. Feta: A light-weight architecture for user oriented semantic service discovery. Proc. European Semantic Web Conference. 2005.
13. Wolstencroft, K., et al., The myGrid ontology: bioinformatics service discovery. International Journal of Bioinformatics Resesearch and Applications, 2007. **3**(3): p. 303--325.
14. Shields, M., Control- Versus Data-Driven Workflows in Workflows for E-science: Scientific Workflows for Grids, I.J. Taylor, et al., Editors. 2007, Springer-Verlag. p. 167-173.
15. Simmhan, Y., B. Plale, and D. Gannon, A survey of data provenance in e-science. SIGMOD Record, 2005. **34**(3): p. 31-36.
16. Missier, P., et al. Data lineage model for Taverna workflows with lightweight annotation requirements. Proc. Second International Provenance and Annotation Workshop. 2008. University of Utah, Salt Lake City, Utah.
17. Zhao, J., et al. Using Semantic Web Technologies for Representing e-Science Provenance. Proc. 3rd International Semantic Web Conference. 2004.
18. Tan, W., L. Fong, and N. Bobroff. BPEL4Job: A Fault-handling Design for Job Flow Management. Proc. International Conference on Service Oriented Computing (ICSOC). 2007. Vienna, Austria.
19. Wassermann, B., et al., Sedna: A BPEL-Based Environment for Visual Scientific Workflow Modeling, in Workflows for E-science: Scientific Workflows for Grids, I.J. Taylor, et al., Editors. 2007, Springer-Verlag. p. 428-449.