Taverna, lessons in creating a workflow environment for the life sciences

Tom OINN[1*], Matthew ADDIS[2], Justin FERRIS[2], Darren MARVIN[2], Mark GREENWOOD[3], Tim CARVER[4], Anil WIPAT[5] and Peter LI[5]

[1]EMBL European Bioinformatics Institute, Cambridge, CB10 1SD, [2]IT Innovation Centre, University of Southampton, SO16 7NP, [3]Department of Computing Science, University of Manchester, M13 9PL, [4]UK MRC HGMP Resource Centre, Cambridge, CB19 1SB, [5]School of Computing Science, University of Newcastle, NE1 7RU.

*To whom correspondence should be addressed.

Background – myGrid & Taverna

The myGrid project is a large scale investigation into the application of grid and semantic web technologies to the scientific domain, so called eScience. As a pilot project funded by the Engineering and Physical Sciences Research Council (EPSRC) in the UK we have performed research into, and developed prototype software for, various aspects of this convergence. This paper presents some of our experiences with the design and implementation of a workflow system tailored towards the eScience community, and towards the bioinformatics domain in particular. We present some of the obstacles, both technical and sociological, that we have encountered, and also our solutions where found.

The team responsible for this aspect of myGrid's work has been expanded by significant volunteer effort both from within the myGrid project and also from other institutes who have an interest in seeing the technology achieve its potential. As a result of this, we have created a project called Taverna as a subproject of myGrid in order to allow rapid dissemination of our work and facilitate easy communication and collaboration with agents outside of our primary project framework. Taverna is an open source project licensed under the LGPL; software is implemented in Java and has proved to be portable without any major issues. Code and further documentation on the specifics of the project are available at our website: http://taverna.sf.net

Workflows in an open world

One area in which myGrid, and in particular Taverna, differ from other projects is our avoidance of a 'closed world' model for our grid and service architecture. A major deliverable of both software and theory is that they

should be able to cope with the diverse heterogeneous services, organizations and methodologies found in the real world. This aim has brought with it challenges that are generally not issues for systems where everything is under one single authority, and yet allows us, if successful, to provide a genuinely useful set of tools and best practices to our target audience, the working scientists.

User guided development

In order to ensure that our development and research efforts are aligned with the requirements of current work in the target scientific domain, we have worked closely with two groups performing research into the genetics of human disease. These groups are based in Newcastle's Centre for Life and Manchester University in the UK. They are working on the genetic basis for, respectively, Graves' disease and Williams syndrome. Both these associations have been invaluable in both providing our project with feedback on the efficacy of our work, and demonstrating that workflow and associated technologies are, while still nascent, already at a level of stability where they can provide a genuine advantage to the life science domain.

In our work we have encountered various issues, which may be broadly grouped into the categories described below. For each group we will examine the issues, their underlying causes, possible solutions (or actual solutions where we have found them) and also the likely future resolutions. The last is required because in many cases the issues or problems are a direct function of the relative lack of experience within the wider community of grid and semantic web technologies and their applications, and the corresponding paucity of available services.

Workflow design

Our intentions with regard to the construction of workflow were quite wide ranging. We believe that workflow construction should ideally be placed in the hands of the domain expert – this corresponds to the role of the researcher in, say, designing a suitable laboratory protocol for their investigation. In general, however, and particularly in the life sciences our domain experts are not intimately familiar with the concepts of service based architectures, let alone the gory details. This immediately creates a gap between, on the one hand a potentially powerful tool, and on the other the target audience for same. Our workflow design methodology therefore has the role of bridging this gap.

Creating an experimental protocol, whether eScience or bench science, can be broken down into various stages. In the first stage, the researcher determines the overall intention of the experiment. This informs a top level design, in our case this would be the overall 'shape' of the workflow, its inputs and desired outputs. This design must then be translated somehow into a concrete plan. In the lab, this translation would consist of choosing appropriate reagents, temperatures and analysis protocols. In our eScience workflow, this maps to the choice and configuration of data and analysis services.

Challenges – service selection

The most immediate problem that a user faces is therefore one of service selection. In the lab, the scientist can rely upon experience and past work to choose experimental parameters and protocols, and may also resort to looking up protocols in laboratory 'cookbooks' or consulting with colleagues. Our challenge here is to facilitate a similar process with service selection.

The first option, that of experience, is obviously only available when the scientists concerned is familiar with the task of workflow construction. In order to use this prior knowledge, we require a way in which previous uses of a service can be located and reused, along with any associated configuration.

The most common task, we believe, will be to locate a new service based on some conceptual description of the service semantics. For example, the scientist may require a component capable of performing a multiple sequence alignment (an operation whereby similarities across a group of genomic or proteomic sequences may be identified). The task therefore for us is to allow this kind of semantic querying across all known service components.

The issues associated with semantic service discovery are, we claim, significantly non technical. While there are serious research challenges to do with the construction of ontologies to adequately capture task information for a domain as diverse as bioinformatics, let alone all science, the main issue we have run into is a sociological one. As part of the myGrid project, we have enriched the UDDI registry service with the ability to store semantic metadata about the services it contains, and have experimented with searches over this store driven by reasoning engine technology. This technology is functional and theoretically capable of performing the task of locating services based on a more or less precise description of their function. The issue here is therefore one of adoption and tooling. While this technology remains in the research stage we cannot use it, and, although myGrid is a major UK project, we cannot force the world at large to deploy our version of the registry service and then register their services with appropriate metadata. We would hope that in the future this will change, and that our work in this area will feed through to drive the next generation of standards such as UDDI, and further that the 'major players' in the industry will provide appropriate tools to make semantic service registration easy enough that people will make use of it, but at the present time semantic search technology is insufficiently widespread to be used 'in anger' by our domain experts.

Locating a service based on another scientist's prior experience is a similar task to locating one based on the researcher's own experience. In order to take advantage of this, the mechanism for recording service use and configuration should allow sharing of these data with colleagues.

Some partial solutions

In Taverna we have taken the simplest possible options for service discovery. Observing that the vast majority of our target services are not registered with any kind of programmatically accessible registry system, let alone one enabled with semantic discovery, we have resorted to a system based on

simple web pages. Users can create a page accessible via HTTP containing links to WSDL documents, when pointed at this page Taverna will explore all available WSDL documents, extract services and make them available within the workbench. While crude, this works remarkably well – the gamut of available services is sufficiently small that we have not yet had to address how to filter this list, although this will clearly be an issue in the future.

In order to facilitate share and reuse of useful configured services, we plan to allow users to load a workflow definition into the service selection panel. In this mode all services in the workflow become available, if selected the service component will be cloned out of the workflow, leaving metadata and configuration information intact.

It should be stressed that we are heavily in favor of both registry and semantic search technologies, the only reason we are not using such is the lack of widespread deployment, and we look forward to a future where this is not an issue.

Challenges – service composition

Once the appropriate service components have been located, the user requires an interface allowing them to compose these services into a workflow. Thankfully this is a simpler and more tractable task than that of locating the components in the first place, and there are a variety of tools available that allow users to work with a graphical representation of the workflow, a form that has a very good fit with the underlying technology and should therefore be relatively intuitive even to a non expert user.

There is a representation issue to address here. While most if not all workflow design packages have adopted a view analogous to electric circuit layout, with services represented as ‘chips’ with pins for input and output, this arrangement can become intractable above a certain level of complexity. If the layout of service components on screen is left under the user’s control this results in an increasingly large amount of time being spent effectively doing graph layout rather than eScience, a clearly undesirable effect.

When composing workflows in an open world, we have no control over the data types used by the component services. It is entirely likely that a service identified by a scientist as being suitable does not use the same type as the preceding service in the workflow, even if the data matches at a conceptual level. The simplest approach is to only allow exact matches, but this is overly restrictive and makes it very hard to do anything useful. A serious consideration, therefore, is how a workflow system can reconcile mismatched types; how much of this can be automatically performed based on service metadata as opposed to user intervention?

An additional requirement that has become clear in the course of our work with large workflows is the ability to annotate workflows, service components and other entities within the workflow (control and data link constructs for example) with arbitrary notes. This requirement derives from the contextual nature of service functionality – the same service may perform two different

roles in the same workflow from the point of view of the scientist. This is especially the case with very general services such as, for example, regular expression based substitution, where the configuration of the service can alter its semantics.

Workflow composition in Taverna

Taverna has two main editing views. The first is a read only graphical view, for which we use the dot tool from AT&T. This provides a well laid out graph with the side effect that the view may be exported in a high quality vector format for publication. The real work of editing the workflow however is performed from a tree view, where the user can see a quick overview of all the entities in the workflow. This is not ideal, but has proved usable even with relatively large flows (over fifty individual components). We have plans in the future to take advantage of more sophisticated representations, but this lies outside the scope (and word limit) of this document.

We have addressed the issue of data typing by making a clear decision as to what we want to know about the types. Internally Taverna uses a carrier data type that is used to wrap up the real data. This carrier exposes any collection structure and allows the data to be annotated with both MIME types and full semantic markup. The exposure of the collection structure allows us to do various type coercion operations, namely implicit wrapping and implicit iteration.

We have observed that it is frequently the case that services in the bioinformatics domain will process lists of items. Often, however, we have a single item of the same type, and Taverna will recognize this disparity and wrap the single item in a list type for processing. The inverse case is also catered for, with Taverna building implicit iterators from the cross product of all service inputs. In general we have found that this behavior corresponds to what users intuitively expect to happen.

Workflow enactment

Once the workflow has been composed, it must be possible to enact the process it describes. In contrast to the design process, which must by definition be exposed to the user, we believe it is desirable to hide as much as possible of the enactment machinery. This in no way suggests that the workflow enactment should appear as a ‘black box’ process, informing the user of the progress of any given workflow enactment is a critical component, but details such as federation and failover between workflow engines, where possible, should not be exposed.

We see workflow enactment as a distinct service, whether actually implemented as such or by some software API. A critical requirement of this service approach is that workflow invocation behavior should be independent of the workflow invocation service used. This is absolutely vital if properties such as reproducibility and verifiability are to be maintained; these are critical for the scientific process, especially because they facilitate peer review of any novel results. A business workflow engine in general does not mandate this

property, as there is rarely any need to invoke a particular business workflow outside its originally envisaged context.

eScience is a highly diversified field with respect to the requirements it places on workflow enactment. At one extreme, particle physics experiments produce vast data sets and corresponding computational loads, with a corresponding requirement to deal with the machinery of classical high performance computing and networking (HPC / HPN). The bioinformatics domain, by contrast, is characterized by massive variety in terms of data types and the resources to operate on them. Workflow enactment systems in this domain must address different concerns to their HPC counterparts, with a greater emphasis on flexible service invocation, collaboration and user interaction within workflows. We claim therefore that while any given workflow solution can in theory handle any given workflow oriented problem; there is a merit in specializing the solutions to the anticipated problem domain.

Workflows in eScience may also vary hugely in terms of expected invocation duration. Workflows that we have implemented up to this point vary between two seconds and two weeks of runtime, with the longest designed (not implemented due to an absence of services) having an anticipated runtime of almost a year. The potential to invoke workflows over this kind of time scale imposes a strong requirement on any such system to keep the user informed as to the progress of their enactment, to allow the user to interact with the running workflow in terms of inspecting intermediate results, manually canceling substructures within the workflow or similar operations, and to do all this from any physical location with preferably minimal network connectivity. We would consider it ideal if these operations were possible from a wireless enabled PDA, computers are often a scarce resource in life science labs, accessing data and workflow progress from a relatively low cost hand held device may well ameliorate this factor.

Fault tolerance and resilience – challenges

Any component based architecture where the components are not under a single controlling authority is going to contain components that will, at some point, fail. It is therefore the responsibility of the aggregating system, in our case the workflow enactment engine, to handle such failures, retaining data integrity and making a reasonable ‘best effort’ to proceed with the invocation. Should this be impossible, it is critical that the reporting functionality is sufficient to inform the user of exactly why the workflow was unable to complete.

We can classify failure modes of a workflow system broadly into the following categories: failure of the enactment engine itself, failure of component services and failure of network fabric. These could be further subdivided but we will use these categories in this analysis.

Failure of the enactment engine

By introducing a single point to which workflows are submitted and from which status reports and results may be obtained, we introduce a single point of failure. This may or may not be a problem; if the workflow engine is running on some massively redundant hardware the chance of it failing might be so low as to be insignificant. The most common case, however, is that the workflow service will be either running on a workstation or some other piece of commodity hardware, and may have a significant chance of failure during the invocation of particularly long running workflows. This places a requirement on the system to be able to deal with problems varying from software failures (the enactor or other) to complete system failure (the computer running the enactor catches fire).

Possible solutions

By using a peer to peer architecture for the enactment engine service it is possible to replicate state intelligently between enactors within a peer group. This renders the enactment process almost immune to single point failures at the engine level. In our case our future implementation of this technology relies on the simple serialization of the workflow state into XML, a capability we originally built in to allow workflow checkpointing.

Failure of services

Service failure is more complex than enactor failure. For example, if a service is down because the machine it runs on has failed, it is probably worth trying again later. If the service failed because your input data was invalid it certainly isn't. While some transports can signal this distinction, in general the toolkits used to generate the services do not provide this facility.

In addition to simply retrying the service invocation, it may be possible to locate or have specified an alternate second service to try should the original fail. In an ideal world this could be inserted automatically, however, the same issues that affect semantic service discovery apply here, namely the lack of available service metadata. All we can realistically provide at the present time is a mechanism in the workflow designer application and corresponding feature in the language that allows the scientist to explicitly state that one service may act as an alternate for another.

Solutions

We believe that current technology is unable to automatically locate alternate services. We will therefore allow users to specify an alternate or alternates explicitly for any given processor. Because of the context sensitive nature of some services we believe that it may be impossible in the general case to perform this substitution automatically, and at the very least would be extremely difficult.

Standard fault tolerance techniques such as retry and exponential back out of retry times can be implemented at the enactor with very little additional work.

Failure of network fabric

This is actually similar to failure of services, but with the additional aspect that it may be possible to probe the network connectivity to a service host using standard internet protocols. This is distinct to a typical web service, which provides no facility to check whether the service is live or not. One common network fabric failure would be the case of the enactor running on a laptop which is moved away from its wireless network – the enactor is not always running on a server machine.

Reporting

Given the variety of failure modes and potential remedies, reporting on the progress of a workflow is a complex task. Various metadata about service invocation is simply unavailable in the general case, for example there is no standard interface that allows a service to define how far through a given invocation it has reached, so a progress display, something users are used to seeing on desktop application software, is either non trivial or impossible.

There is significant information available, however, and therefore a presentation issue of exactly how to show this to the user. When a workflow may contain fifty or more processing components, and each of these components can be retrying, using alternate implementations etc the complete workflow state is highly complex, and yet we require a visualization of it that allows the user to see at a glance what is happening, acquire intermediate results where appropriate and control the workflow progress manually should that be required.

Reporting in Taverna

Related to our provenance collection, the reporting mechanism consists of a stream of events for each processing entity, with these events corresponding to state transitions of the service component. For example, we emit a message when the service is first scheduled, when it has failed for the third time and is waiting to retry etc. These message streams are collated into an XML document format and the results presented to the user in tabular form.

Provenance collection

Scientists are, reasonably, interested in the results of any given process. This interest, however, goes beyond examining the results themselves and extends to the context within which those results exist. Specifically, the scientist will wish to know from where a particular result was derived, which process was used and what parameters were applied to that process. We can therefore distinguish between provenance of the data and provenance of the process, although obviously the two are linked.

The primary task for data provenance is to allow the exploration of some novel result and the determination of the derivation path for the result itself in terms of input data and intermediate results en route to the final value. This requirement motivates architecture whereby the ‘side effect’ information

generated during a workflow invocation may be stored and queried. We define side effect information as anything that could be recorded by some agent observing the workflow invocation, and implicitly or explicitly links the inputs and outputs of each operation within the workflow in some meaningful fashion.

Process provenance is somewhat simpler than data provenance, and is more similar to traditional event logging. It is complicated somewhat by the requirements for fault tolerance and the correspondingly larger range of possible events that may occur. It is critical that in the event of a failure of some kind the user can be informed in such a way that they can comprehend the failure and take appropriate action where required.

Provenance collection with semantic web and LSID

One of our active areas of investigation is the combination of the Life Science ID (LSID) scheme with semantic web technologies such as RDF to describe the provenance of data resulting from a workflow invocation. Our intention is that side effect knowledge in the form of RDF statements is collected every time a service is invoked, with the potential to enrich these statements with additional semantics based on explicit markup of the service concerned. At the base level, the statements will express that 'result a was produced by process b on inputs c, d, and e'. By marking up the service with additional semantics it would be possible to collect additional, more specialized, information such as 'result a is the predicted structure of input b'.

The sheer volume of side effect data collected in this fashion will create its own issues, there are research avenues opening in intelligent summarizing systems that could convert these data back into natural language, for example.

Acknowledgements

This work is supported by the UK e-Science programme EPSRC GR/R67743. The authors would like to acknowledge the myGrid team: Nedim Alpdemir, Rich Cawley, Neil Davis, Alvaro Fernandes, Robert Gaizaukaus, Kevin Glover, Carole Goble, Chris Greenhalgh, Yikun Guo, Anath Krishna, Phil Lord, Simon Miles, Luc Moreau, Arijit Mukherjee, Juri Papay, Savas Parastatidis, Norman Paton, Milena Radenkovic, Peter Rice, Martin Senger, Nick Sharman, Robert Stevens, Victor Tan, Paul Watson and Chris Wroe; and our industrial partners: IBM, Sun Microsystems, GlaxoSmithKline, AstraZeneca, Merck KgaA, geneticXchange, Epistemics Ltd, and Network Inference.

References

Addis,M., Ferris,J., Greenwood,M., Li,P., Marvin,D., Oinn, T. and Wipat, A. (2003) Experiences with e-Science workflow specification and enactment in bioinformatics. *Proc UK e-Science All Hands Meeting 2003*, pp. 459-466.

Booth D, Haas H, McCabe F, Newcomer E, Champion M, Ferris C, Orchard D. (2003) Web Services Architecture. *W3C* http://www.w3.org/TR/ws-arch/

Goble, C.A., Pettifer, S., Stevens, R., and Greenhalgh, C. (2003) Knowledge Integration: In silico Experiments in Bioinformatics, in *The Grid 2: Blueprint for a New Computing Infrastructure* Second Edition eds. Ian Foster and Carl Kesselman, November 2003

Lord,P., Wroe,C. Stevens,R., Goble,C. Miles,S. Moreau,L., Decker,K. Payne,T. and Papay,J. (2003) Semantic and Personalised Service Discovery. In W. K. Cheung and Y. Ye, editors, *Proceedings of Workshop on Knowledge Grid and Grid Intelligence (KGGI'03), in conjunction with 2003 IEEE/WIC International Conference on Web Intelligence/Intelligent Agent Technology*, pp. 100-107, Halifax, Canada, October 2003.
Oinn, T. (2003) Talisman - rapid application development for the grid. *Bioinformatics*, **19 (Suppl. 1)**, i212-i214.

Oinn,T., Addis,M, Ferris,J, Marvin,D, Greenwood,M, Wipat,A., Li, P. and Carver,T. (2003) Delivering Web service coordination capability to users. WWW2004. Submitted.

Rice, P., Longden, I. and Bleasby, A. (2000) EMBOSS: the European Molecular Biology Open Software Suite. *Trends Genet.,* **16**, 276-7.

Senger M (2002) Bibliographic query service. *http://industry.ebi.ac.uk/openBQS/*

Senger,M., Rice,P. and Oinn,T. (2003) SoapLab - a unified Sesame door to analysis tools. *Proc UK e-Science All Hands Meeting 2003*.

Stein L. (2002) Creating a bioinformatics nation. *Nature*, **417**, 119-120.

Stevens,R., Glover,K., Greenhalgh,C., Jennings,C., Pearce,S., Li,P., Radenkovic,M. and Wipat,A. (2003) Performing in silico experiments on the Grid: a users perspective. *Proc UK e-Science All Hands Meeting 2003*, 43-50.

Wroe C, Stevens R, Goble C, Roberts A, Greenwood M. (2003) A suite of DAML+OIL Ontologies to Describe Bioinformatics Web Services and Data. in *International Journal of Cooperative Information Systems* special issue on Bioinformatics. **12** (2), 197-224.