Programming Paradigms for Scientific Problem Solving Environments

Dennis Gannon, Marcus Christie, Suresh Marru, Satoshi Shirasuna, Aleksander Slominski

Department of Compute Science, School of Informatics,
Indiana University,
Bloomington, IN 47401
gannon@cs.indiana.edu

Summary. Scientific problem solving environments (PSEs) are software platforms that allow a community of scientific users the ability to easily solve computational problems within a specific domain. They are designed to hide the details of general purpose programming by allowing the problem to be expressed, as much as possible, in the scientific language of the discipline. In many areas of science, the nature of computational problems has evolved from simple desktop calculations to complex, multidisciplinary activities that require the monitoring and analysis of remote data streams, database and web search and large ensembles of supercomputer-hosted simulations. In this paper we will look at the class of PSE that have evolved for these "Grid based" systems and we will consider the associated programming models they support. It will be argued that a hybrid of three standard models provides the right programming support to handle the majority of the applications of these PSEs.

1 Introduction

Domain specific problem solving environments have a long history in computing and there are several examples of widely used tools that are also commercial successes. For example Mathematica [1] provides a platform for doing symbolic mathematics and related visualization tasks using a programming language that is designed with mathematical primitives as a basic component of the type system. Another example is Matlab [2], which is widely used in the scientific community to study problems requiring matrix manipulations or other linear algebra operations. In the area of computer graphics PSE like AVS and Explorer [3] pioneered the use of programming by component composition to build visualization pipelines. This same approach is used in SciRun [4] and many of the other systems described below.

In recent years, we have seen a shift in the nature of the problems scientists are trying to solve and this is changing the way we think about the design of PSEs. Specifically, many contemporary computational science applications require the integration of resources that go beyond the desktop. Remote data sources including online instruments and databases and high-end supercomputing platforms are among

the standard tools of modern science. In addition multidisciplinary collaborations involving a distributed team of researchers are becoming a very common model of scientific discovery. Grid computing was invented to make it easier for applications and research teams to pool resources to do science in such a distributed setting. Grids are defined as a service oriented architecture that allows a group of collaborators, known as a virtual organization (VO), to share access to a set of distributed resources. There are three primary classes of core services that Grids provide that make it easier to build PSEs that use distributed systems. These services are:

- Security - authentication and authorization
- Virtualization of Data Storage
- Virtualization of Computation

The PSE that is built on top of a Grid service framework is often called a science gateway, because it provides a portal for a community to access a collection of resources without requiring them to be trained in the distributed systems and security technology that the Grid is built upon. As illustrated in Figure 1, the user's desktop interaction is through a browser and other tools which can be started with a mouse click in the browser. A remote server mediates the user's interaction with the Grid security services, the virtual data storage and metadata catalogs and application resources. The user's programs are represented as workflows that are executed by a remote execution engine.

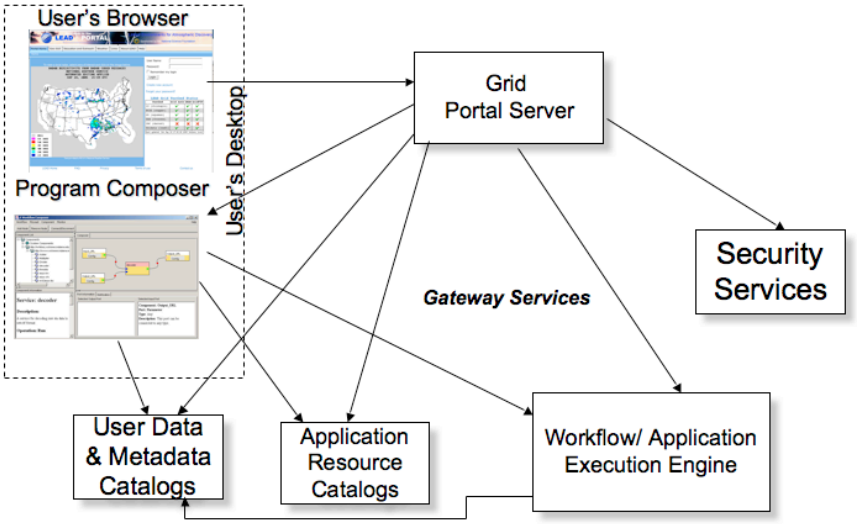

Figure 1. The organization of top level services in a science gateway PSE.

In this paper we will look at the problem solving programming model that is evolving for these Grid science gateway PSEs and suggest ways in which it can be extended in the future.

2 Programming in a Science Gateway PSE

The access point to a science gateway is usually based on a web portal that allows users access to the collective computational and data management resources of the underlying Grid. There are many examples of these gateway portals currently in use. The TeraGrid web site (http://www.teragrid.org) has links to many of these. They include

- The National Virtual Observatory (NVO), a gateway for astronomical sciences.
- Linked Environments for Atmospheric Discovery (LEAD), a PSE portal for mesoscale weather prediction.
- Network for Earthquake Engineering Simulation (NEES), a gateway for earthquake hazard mitigation.
- The GEOsciences Network (GEON), a geophysics gateway.
- Network for Computational Nanotechnology and nanoHUB, a PSE for access to nanotechnology tools.
- The Earth System Grid (ESG), a portal for global atmospheric research.
- The National Biomedical Computation Resource (NBCR), a gateway focused on integrative biology and neuroscience
- The Virtual Laboratory for Earth and Planetary Materials (VLAB), which focuses on materials research.
- The Biology and Biomedicine Science Gateway (The Renci Bioportal) which provides resources and tools for molecular biosciences.
- The Telescience Project, a gateway for neuroscience biology.

This is only a small sample. There are many other significant gateway projects in the U.S., Europe and Japan. While there are many unique features supported by these gateways, they also share many common attributes. Perhaps the most important feature they all share are mechanisms that provide access to community data. Science has become more data driven. The Scientists and engineers need to be able to search for, discover, analyze and visualize the data produced by instruments and computational experiments. They need to have mechanisms to discover new data based on searchers of metadata catalogs and they need tools to extract this data and save it in a gateway workspace for later use.

Once a scientist has collected the data (or identified the required data sources), he or she must begin the process of analyzing it. The data is frequently used as the input to a large simulation, a data mining computation or other analysis tool. A simple approach to the design of a PSE is to wrap up all the important application components and present a web portal user-interface page to the user for each one. For example, a simulation program may require one or two standard input files and a desired name for the output file or files. These may be exactly what is required to run the simulation program from the command line. The advantage of providing the input parameters in a portal web page is that we can transfer the complexity of selecting the best computer to run the application and establishing all the needed libraries and environment variables to the back-end Grid system. The user need only identify the input and output data set names.

While a simple web interface to individual applications is useful, life is seldom this simple. Specifically, the data is seldom in exactly the form that the analysis tools expect, so transformations must be applied to make it fit. These transformations may be format conversions, data sub-sampling, or interpolation. The task may also

involve data assimilation, where multiple data sources must be merged or aligned in a particular way to meet the requirements of the simulation task. There may be many such preprocessing tasks and the analysis/simulation part of the activity may require the use of more than one package. Finally, there may be post-processing to create visualizations or other reports. And, as with most scientific experiments, the sequence of transformations, data analysis, mining, simulation, and post processing must be repeated in exactly the same way for many different input data samples. Programming the sequence of steps required to do such an analysis scenario is known as workflow design and this term is now in common use in the e-Science community.

The second most important feature of any science gateway PSE is to provide a mechanism for users to create workflow scripts that can be saved and later bound to input files and executed automatically using the remote grid resources. A recent study [5] has identified a dozen popular workflow tools used by these PSEs. The four most commonly used tools are Kepler [6], which is used in a variety of application domains, Taverna [7], a common tool for life-science workflows, Pegasus [8], used by many large physics applications, and BPEL [9], the industry standard for web service orchestration. In a later section of this paper we describe how BPEL has been integrated into the LEAD science gateway.

3 Compositional Programming Models in e-Science

To see how these tools work we need to look at the semantics of their graphical composition. Within the e-Science community, the primary model of workflow composition is based largely on macro-dataflow concepts. The idea is very simple. Scientific analysis is based upon transformation of data. An experiment begins with raw data. These data are often derived from experimental measurement, such as from a collection of instruments. The data must be pre-processed or "assimilated" into a coherent set of inputs to analysis or simulation packages. The output is then routed to final analysis or visualization tools. This is "programmed" by using a graphical tool which uses icons to represent the individual tasks as components in a workflow. As illustrated in Figure 2, each workflow component has one or more inputs and one or more outputs. Each input represents a data object or "message" that is required to enact the component and each output represents a result data message. The data

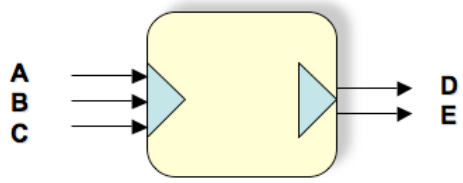

Figure 2. Each icon represents a process "component" with one or more required inputs and one or more output data objects.

object may be a numerical value, a string or the URI of a file. In some systems the data object may be a continuous stream of data to be processed. As illustrated in Figure 3, two components may be composed if the output of one component can serve as a valid input to another component. "Unbound" inputs represent the data sources for the workflow and the unbound outputs are the final data products.

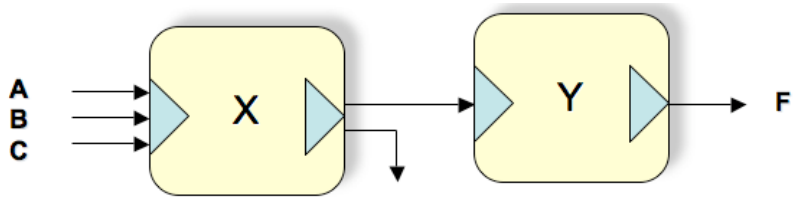

Figure 3. The components may be composed. In this case one result from component X is used as an input to component Y. A, B, and C are unbound inputs which must be supplied by the user at runtime.

In the typical system based on this model, the programmer drags icons onto a pallet and wires together the dataflow for the experiment. Figure 4 illustrates the interface to the XBaya system used in the Linked Environments for Atmospheric Discovery (LEAD) project [10, 11].

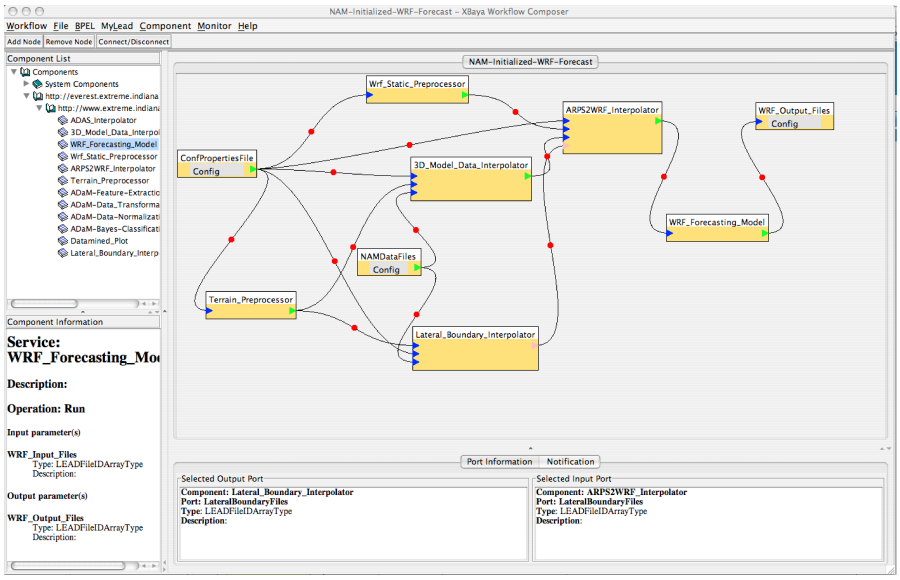

Figure 4. The XBaya workflow composition tool used to build a storm forecasting workflow.

Unfortunately, there are several problems with this basic model of dataflow driven workflow as described above. The first problem relates to the way components are connected. When is the output of one component suitable as an input to another component? Clearly, if they have conflicting simple types, such as providing a String as an input to something that is expecting a Float, then it is easy for a rudimentary type system to detect the error. But most problems are due to subtle semantic differences between the content of the message that is passed. For example, in large systems, the message often only contains the URI of a data file that is stored on a remote resource. How do we know if the data file has the right format or content to be used by the destination component? The solution to this problem lies in providing complete information about the exact semantics and format of each input and output. This metadata needs to be attached to the component and some form of metadata analysis would be required to check compatibility. Without a common metadata schema, a component provided by one group of researchers cannot be used by another group. Consequently, it is up to the scientist composing the workflow to understand this issue.

The second problem with this simple model is that it does not take into account the control dependencies that a typical computer program uses. For example, conditionals and iteration are difficult to express in a language where the only operation is the composition of directed acyclic graphs. However, it is not difficult to overlay additional control operations over the dataflow. For example, a conditional can be expressed by a component (Figure 5) that takes two inputs, a value message, and a conditional predicate that the message must satisfy. There are two outputs. If the predicate evaluates to true, then the value is forwarded out one output. If the value is false, the other message is generated.

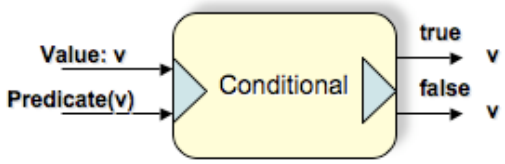

Figure 5. A simple conditional element with two inputs: a value and a predicate. Based on the predicate value one of the output messages is generated.

Another essential component of any complete e-science workflow programming tool is the expression of iteration. There are two cases to consider. The first is the classical case of a "while" loop. As illustrated in Figure 6, the input is a predicate, an initial iterate value and a set of data values. The predicate is applied to the iterate value and if the result is true, the iterate value and data values are passed to a subgraph. The subgraph transforms both the data values and applies some function to the iterate value. These are fed back to the while control node and the test is repeated.

The second form is a parallel "for each" that can be used when you wish to execute a subgraph for each element of a set of data values. In this case the subgraph

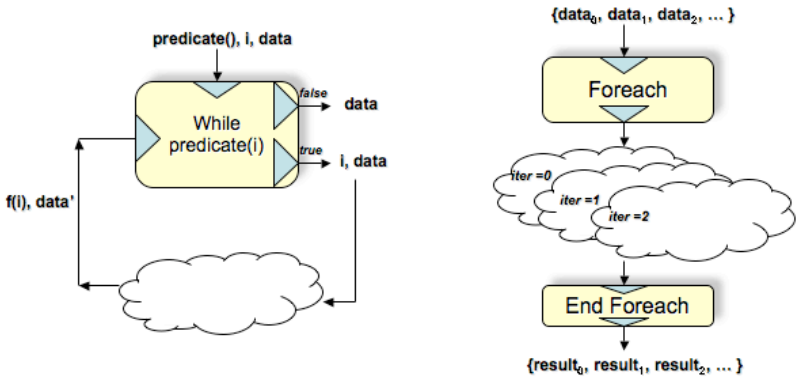

Figure 6. A "while" loop and a parallel "for each" element.

is also supplied with an additional "iteration" index so the different invocations of the subgraph can be uniquely identified. This additional index is important when the subgraph must create a side-effect outside the body of the workflow. For example, when an element of the workflow creates a file, it must be distinguished from the file generated by the other instances. However, the exact semantics of how such an iteration index is propagated to the body of a "for each" loop is non-trivial and not a topic for this paper.

There are other standard features of workflow composition tools in this category. For example, it is important to be able to encapsulate any valid composed workflow as a component which can be used in other workflows.

Finally a topic that is always overlooked by e-Science workflow systems is that of exceptions. An exception occurs when a specific component realizes that it cannot correctly process an incoming message. As with any modern programming language, it is essential that the system have a mechanism to capture these runtime exceptions and deal with them. The model often used in programming languages, where a block of code is encapsulated in a "try" block which is followed by a "catch" block which is responsible for handling the exceptional conditions, can be used in graphical dataflow-based systems. In the graphical case we can simply identify a subgraph that may throw an exception and provide a description for a replacement "catch" subgraph. The exceptions that are the most frequent are those that are related to access to remote resources. For example, a remote service that fails to respond because of a network or other resource failure. In these cases it is often better to handle the problem at a lower, resource allocation level than at the abstract workflow graph level. A situation that may be handled at the graph level could be one where a request to an application component is simply too large or, for some other reason, too difficult to process. In these cases, the workflow designer may know that an alternative service exists that can be used in special cases like this.

4 The Service Architecture of a Science Gateway PSE.

The science gateway PSE programming model we have described so far is based on building applications by composing application services. The LEAD gateway is like many others in that the components services are implemented as Web services. This allows us to use standard robust middleware concepts and tooling that is widely used in the commercial sector. However, large-scale computational science is still the domain of big Fortran applications that run from the command line. To use these applications in a Web service based workflow we need to encapsulate them as services. To accomplish this we use an Application Factory Service [12], which when given a description of an application deployment and execution shell script, automatically generates a web service that can run the application. As illustrated in Figure 7, the service takes as input command-line parameters and the URLs of any needed input files. The service automatically fetches the files and stages

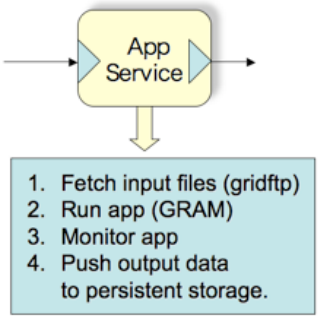

Figure 7. The application services provide a mechanism to execute applications on behalf of the user on remote resources.

them in a subdirectory on the machine where the application is to run. It then uses a remote job execution tool (Globus GRAM [13]) to run and monitor the application. Finally the output files are pushed to the data storage facilities. During the invocation of the service the progress of the data transfers and the monitoring of the application are published as "events" to a message notification bus. The bus relays the messages to listening processes including the user's private application metadata catalog. This allows the user to consult the catalog from the portal to see the status of the execution.

To tie this all together we need to fill out a more complete service oriented architecture (SOA). The portal and workflow composer are only one piece of the system. One important component is the workflow engine. While most e-Science workflow tools also double as the execution engine, the XBaya system is actually a compiler. It can either directly execute the workflow or it can compile a python program which, when run, does the execution, or it can generate a BPEL document. BPEL is

the industry standard for web service orchestration and many commercial and open source execution engines exist. The importance of having an execution engine that is separate from the composition tool cannot be understated. Science workflows can take a very long time to execute. This is especially true in the case where a workflow is driven by data from instruments where an event from the instrument may not come for months! The execution engine must be able to retain the state of the workflow in persistent storage so that it can survive substantial system failures. Even the workflow engine may need rebooting. Figure 8 illustrates the parts of the SOA that are directly involved in the execution of the workflows. The only detail of the

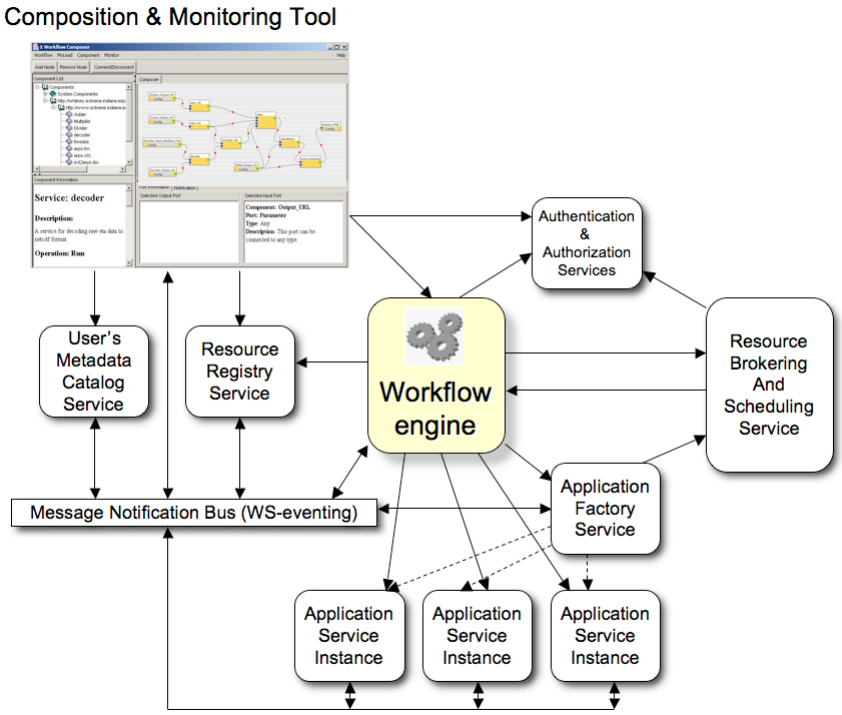

Figure 8. The organization of services in a science gateway PSE.

SOA workflow execution we have not discussed is the process of resource allocation and brokering. When a workflow is composed it is in an abstract form: the specific application services used in the graph are not bound to specific instances of services ready to run the application on specific hosts. The application factory service is responsible for instantiating the application services, but the specific instances are selected by a resource brokering and workflow configuration service. There are many ways to do resource brokering and this topic is far beyond the scope of this paper.

It should also be noted that we have not described the complete picture of the the SOA for an e-Science PSE. A major component not discussed here is the data

subsystem. e-Science revolves around data. The workflow system only transforms the data. This topic is treated in another paper in this workshop and elsewhere [14, 15].

5 Event Bus based PSE organization.

There are other approaches to building a PSE programming system that are often overlooked because the dataflow graph model is so intuitive for scientists. Rather than thinking in terms of composing applications as explicit dataflow/control flow graphs, we can consider the possibility of program components that respond to their own environment in productive ways. The concept is based on an information bus as illustrated in Figure 9. In this model a component "subscribes" to messages of some type or "topic" or containing certain content. Any component may "publish" messages on some topics for others to hear. To understand this, we should consider an example. Data from an instrument is gathered and published by an instrument component sitting on the bus. The user inserts data filters onto the bus which captures the data events and transforms them and republishes them. These events are captured by a data analysis component, which publishes results. The results are captured by different rendering tools. This type of system, which resembles a blackboard model [11], is extremely flexible and dynamic.

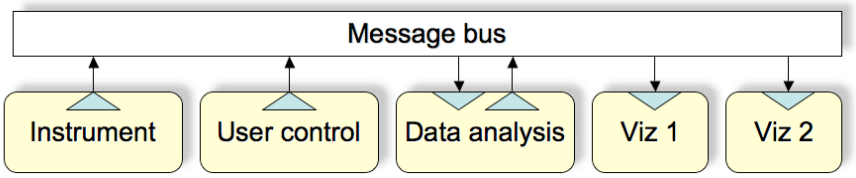

Figure 9. The message bus architecture allow a more dynamic organization than the fixed dataflow model of execution.

This information bus model is the most flexible for integrating user interaction into the system. Future systems will likely contain a combination of bus-based and dataflow approaches.

6 Discussion.

As part of this workshop a series of questions were posed to the authors from other participants. In the spirit of the workshop we will devote our conclusion to a discussion of the points they raised.

- Q1: Anne Trefethen. You mentioned MATLAB as one of the classic PSEs. Have you looked at MATLAB Simulink, SimBiology, or SimEvents, which seem to have the same kind of graphical interface? Have they solved any of the issues you raise?

Yes. These tools all use a graphical interface similar to the ones we have discussed here. There are many more examples. This model of programming is certainly not new. Many domain specific composition tools are able to reduce the complexity of the problem by simplifying the semantic space. SImBiology is an excellent example. However, most of these systems are not designed to operate in the wide area as web service workflow engines. However MATLAB does have support for Web service integration, so it is possible to integrate web services into a MATLAB-based application framework.

- Q2: Tom Jackson. How do you deal with the problem of integrating legacy user code into portals (which are typically non-Java), particularly for visualization?

As discussed in Section 4 of this paper, legacy application integration is accomplished by wrapping the application as a web service. This is a semi-automatic process. In the case of visualization, it is possible to wrap an off-line rendering system as a web service and we have done that. A more complex problem is to invoke a "live" desktop application as part of a workflow. This is a general problem many systems have with inserting a human action into the workflow. The best solution is to combine the dataflow model with the event-bus model described above.

- Q3: Gabrielle Allen. How do you deal with resource allocation and/or resource scheduling in these scenarios?

As mentioned in section 5, resource allocation and scheduling is handled by a "call-out" to a resource allocation service from the workflow configuration service. This use of late binding of the resources with the workflow script allows for very great flexibility. If the workflow engine is also able to catch exceptions and listen to the event notification bus, it is possible to change the resource allocation while the workflow execution is continuing.

- Q4: Tom Jackson. Where you referring to Enterprise Service Bus architectures when you discussed message bus solutions?

Yes. Although Enterprise Service Bus is often associated with a specific technology such as an EJB/JMS solution. However the concept is identical.

- Q5: Gabrielle Allen. What is the difference between event-driven and data-driven architectures, and can you integrate these with a centralized component which allows decision making and control to only need to be implemented in one place?

A workflow or computation can be data-driven and implemented with an event-driven bus framework or with a dataflow framework. There is a big difference between dataflow (as described here) and an event-driven bus. In the case of dataflow the workflow designer implements control based on a graph of dependencies that must be satisfied. Messages from one service are explicitly routed to the graphically connected services. In the even-driven bus case each service can hear all messages and respond to any of them. We control chaos by selecting services that only respond to messages that are of the appropriate topic.

- Q6: Bill Gropp. Have formal methods for verifying correctness been applied to graphical workflows?

Yes and No. There is ample work in the theoretical literature about the semantics and correctness of these graphical models, but we know of no system in use that implements any of these idea in practice.

- Q7. Richard Hanson. Libraries of software routines are well established as a programming model and tool. What do you visualize as an execution model for grid computing and workflows?

In many ways, what we have described here is a way to deploy application software libraries in a distributed context. But there is an important and subtle difference between software components and traditional software libraries. Most software libraries are not well encapsulated: they rely on the runtime environment of the program invoking them and they often operate by side-effecting common data structures. The behavior of component systems is completely defined by the interfaces they present to their clients.

- Q8: (Mo Mu) What do you think is the role of APIs in the composition of workflows as a mechanism/standards to ensure the proper fitting of components/services?

In a Web service oriented system, interfaces are defined by the Web Service Definition Language. This provides a programming language neutral way to describe the messages sent to a service and the types of messages that are returned. Also, Web service systems have evolved considerably from the days of remote procedure calls. The stand now is message oriented, where the message is an XML document defined by an XML schema. The reply is defined similarly. By using WSDL and XML schemas, the services become completely programming language neutral. Services built from Java or C++ or .Net or Perl or Python can all interoperate. This was not possible with programming language based APIs because they all have different type systems.

- Q9 : (Keith Jackson) What role dows semantic information play in a component architecture? What kinds of semantic information should a service expose?

Semantics are critical. Current service models do not provide enough semantics about the content of messages and responses. As discussed above this is one of the greatest challenges to making a truly interoperable system of service components.

- Q10: (Anne Trefethen) How do we get community agreement on the semantics?

This is perhaps the most important question. The first step is to get a community to agree upon an ontology. This is starting to happen in many scientific domains. Once there is a common ontology, one can start defining common scientific metadata. Again this is happening in atmospheric science, oceanography, physics, geology, and many more areas. But there is a long way to go. Once you have a common ontology and common scientific metadata, then wrapping community codes to work as services in general e-Science PSE frameworks is relatively easy.

References

1. S. Wolfram, Mathematica: a system for doing mathematics by computer, 1991, Adison Wesley Co.
2. D. Hanselman, B. Littlefield, Mastering MATLAB 5: A Comprehensive Tutorial and Reference, 1997 - Prentice Hall PTR Upper Saddle River, NJ, USA

3. C. Upson , T. Faulhaber, Jr. , D. Kamins , D. H. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, A. van Dam, The Application Visualization System: A Computational Environment for Scientific Visualization, IEEE Computer Graphics and Applications archive Vol. 9 , no. 4, July 1989, pp. 30 - 42
4. S. Parker, C. Johnson, SCIRun: a scientific programming environment for computational steering, Proceedings of the 1995 ACM/IEEE conference on Supercomputing, San Diego, California, United States Article No. 52, 1995.
5. I. Taylor, E. Deelman, D. Gannon, M. Shields (Eds.) , Workflows for e-Science Scientific Workflows for Grids, Springer, 2007.
6. D. Pennington, D. Higgins, A. Townsend Peterson, M. Jones, B. Ludascher, S. Bowers, Ecological Niche Modeling Using the Kepler Workflow System. in Workflows for e-Science Scientific Workflows for Grids, Springer, 2007.
7. T. Oinn, P. Li, D. Kel l, C. Goble, A. Goderis, M. Greenwood, D. Hul l, R. Stevens, D. Turi and J. Z hao, Taverna / myGrid: aligning a workflow system with the life sciences community, in Workflows for e-Science Scientific Workflows for Grids, Springer, 2007.
8. E. Deelman, G. Mehta, G. Singh, M-H. Su, K. Vahi, Pegasus: Mapping Large-Scale Workflows to Distributed Resources, in Workflows for e-Science Scientific Workflows for Grids, Springer, 2007.
9. A.Slominski, Adapting BPEL to Scientific Workflows, in Workflows for e-Science Scientific Workflows for Grids, Springer, 2007.
10. K. Droegemeier, D. Gannon, D. Reed, B. Plale, J. Alameda, T. Baltzer, K. Brewster, R. Clark, B. Domenico, S. Graves, E. Joseph, D. Murray, R. Ramachandran, M. Ramamurthy, L. Ramakkrisshnan, J. Rushing, D. Webeer, R. Wilhelmson, A. Wilson, M. Xue, S. Yalda, Service-Oriented Environments for Dynamically Interacting with Mesoscale Weather, CiSE, Computing in Science & Engineering – November 2005, vol. 7, no. 6, pp. 12-29.
11. B. Plale, D. Gannon, J. Brotzge, K. Droegemeier, J. Kurose, D. McLaughlin, R. Wilhelmson, S. Graves, M. Ramamurthy, R. Clark, S. Yalda, D. Reed, E. Joseph, V. Chandrasekar, CASA and LEAD: Adaptive Cyberinfrastructure for Real-Time Multiscale Weather Forecasting, IEEE Computer, November 2006 (Vol. 39, No. 11) pp. 56-64
12. Gopi Kandaswamy, Dennis Gannon, Liang Fang, Yi Huang, Satoshi Shirasuna, Suresh Marru, Building Web Services for Scientific Applications, IBM Journal of Research and Development, Vol 50, No. 2/3 March/May 2006.
13. I Foster, C Kesselman, Globus: A metacomputing infrastructure toolkit, International Journal of Supercomputer Applications, 1997
14. Y. Simmhan, S. Lee Pallickara, N. Vijayakumar, and B. Plale, Data Management in Dynamic Environment-driven Computational Science, IFIP Working Conference on Grid-Based Problem Solving Environments (WoCo9) August 2006, to appear as Springer-Verlag Lecture Notes in Computer Science (LNCS).
15. Beth Plale, Dennis Gannon, Yi Huang, Gopi Kandaswamy, Sangmi Lee Pallickara, and Aleksander Slominski, Cooperating Services for Data-Driven Computational Experimentation", CiSE, Computing in Science & Engineering – September 2005 vol. 7 issue 5, pp. 34-43