Scientific Workflows and Components: Together at Last!

Kostadin Damevski, Ayla Khan, Steven Parker
Scientific Computing and Imaging Institute
University of Utah, Salt Lake City, Utah 84112, USA
{damevski,ayla,sparker}@sci.utah.edu

Abstract

Composing scientific simulations from smaller general parts has many beneficial properties such as modularity and reuse. Component software and scientific workflows are both technologies for decomposition of scientific simulation, although each of them performs this task along a separate dimension and on a different level. Scientific workflows are concerned with high-level orchestration of a time-decomposed simulation, while components are mainly used at a lower level to enable software modularity and reuse at a small performance cost. Combining these two technologies in a seamless way enables using the benefits of both of these decomposition paradigms, resulting in more flexibility for scientific application design. In this work, we describe a design of our system that communicates parameters and results to and from a Common Component Architecture (CCA) framework (e.g. SCIJump, CCaffeine) and the Kepler scientific workflow system.

1. Introduction

The complexity of modern scientific simulations has necessitated methods of decomposing the application into modular reusable parts. These interchangeable modules can be connected to create applications and subsequently executed. Scientific workflows and components are technologies aimed at better modularity and reuse of scientific software that are supported by growing user communities and growing numbers of reusable modules available on each platform. However, each of these technologies is different in its approach: components communicate through method invocation and have a spatial composition style, while workflows use dataflow to communicate and are composed temporally. Also, scientific workflows are intended to guide an application from the highest level including tasks such as communicating with outside data sources, while components are used for slightly finer-grain tasks.

We argue that scientific workflows and components are complimentary technologies and not competing ones. Each of them is better suited for a particular class of problems and a scientific programer would benefit most from the ability to combine these two decomposition schemes into one working application, leveraging the best tool for each part of the problem. Thriving user communities have produced and are producing interesting component and workflow application, and combining parts of them may yield the best approach for fast scientific discovery. In this paper we present our prototype solution that enables cooperation between the Kepler [1] workflow system and any CCA component framework resulting in a hybrid decomposition model that can use both components and workflows.

The decomposition paradigm of each of these technologies prescribes a hierarchical design for interoperability: workflows as top level application control and components as the means of implementing a part of it. In this hierarchical model, a full duplex (two-way) method of communication is necessary as a workflow may need to adapt based on (intermediate) results of a component simulation. Also, it is important to provide the user with the ability to enable both coarser and finer grain communication between the two systems. The system we architected provides a set of Kepler actors that directly communicate with a component framework and other actors that are able to directly communicate to individual components. The actor to individual components connection may only need to be used to execute a set of connected components, however, a more complex scenario where components and actors communicate at multiple points of an application is possible (e.g. to communicate intermediary results, gage progress or for computational steering). We have a prototype implementation of our design that allows this kind of hybrid application decomposition between Kepler and the SCIJump [8] component framework.

We organize the discussion of our design of interoperable software architectures as follows. In the next section we introduce some of the background behind scientific component and workflow technologies as well as some of the accompanying tools. In Section 3 we present some the re-

lated work, and in Section 4 we show a detailed view of our design and implementation of workflow and component interporability. Section 5 contains discussion of a proof of concept example of a hybrid application. Finally, we conclude and discuss the future work of this project in Section 6.

2. Background

A workflow consist of a series of execution steps represented by one or more actors that commonly execute in a dataflow fashion. The actors are loosely coupled and some of them control staging of simulation data, or other auxiliary tasks, while others manage a computation. Together the actors form a workflow the highest level of application flow and management. The Kepler scientific workflom system, based on the Ptolemy II framework for embedded software, allows scientists to design scientific workflows and execute them efficiently using emerging approaches to distributed computation.

The Common Component Architecture (CCA) specification for scientific components [3] was created through a colaboration of US Department of Energy research labs and several universities and has a large user following. Scientific components, as specified by the CCA, are usually much more fine grained than workflow actors. For instance, a series of components may be used to perform different stages of a computation. Scientific components use a "provide and use" connection paradigm and communicate via method invocation on a firmly set interface. CCA components can also support a more complex sets of interactions (such as between parallel components). A component framework is used to connect the components and start the application execution. The execution takes place without any further framework interferance.

2.1. CCA Components

The CCA model consists of a framework and an expandable set of components. The framework is a workbench for building, connecting and running components. A component is the basic unit of an application. A CCA component consists of one or more ports, and a port is a group of method-call based interfaces. There are two types of ports: **uses** and **provides**. A provides port (or callee) implements its interfaces and waits for other ports to call them. A uses port (or caller) issues method calls that can be fulfilled by a type-compatible provides port on a different component. A CCA port is represented by an interface, which is specified through the Scientific Interface Definition Language (SIDL).

The CCA component model relies on a Scientific Interface Definition Language (SIDL) to define components, ports, as well as define the component model itself. A component specification is written in SIDL and compiled into glue code that is later compiled into an executable together with the user code. The prevalent way of compiling SIDL is by using the Babel compiler [5]. Babel has the capability of compiling SIDL to bindings for several popular programming languages (C++, Java, Python, Fortran77 and Fortran95), enabling coupling of components written in any of these languages. Babel has a large and growing user community and is an important technology behind the CCA component model. Recently, the compiler was upgraded to produce bindings for distributed computing through Remote Method Invocation (RMI). The Babel RMI bindings provide a way for components existing on separate computing resources to communicate with little or no help from the user. Babel's RMI also provides a general interface within the compiler to plug in any kind of wire protocol. The wire protocol library is invoked by the Babel generated code to produce behavior that the user is expecting. Our approach leverages Babel and its wire protocol in actors containing Babel-generated stubs able to directly communicate to CCA components and services.

2.2. SCIJump

SCIJump is a framework built on SCIRun [4] infrastructure that combines CCA compatible architecture with hooks for other commercial and academic component models (see Figure 1). It provides a broad approach that will allow scientists to combine a variety of tools for solving a particular problem. The overarching design goal of SCIJump is to provide the ability for a computational scientist to use the right tool for the right job. SCIJump utilizes parallel-to-parallel remote method invocation (RMI) to connect components in a distributed memory environment, and is multi-threaded to facilitate shared memory programming. It also has an optional visual-programming interface.

2.3. Workflows

Scientific workflow systems provide a master problem solving environment that combines scientific data management, analysis, simulation, and visualization tasks in a workflow fashion. Workflows are usually more dataflow oriented and closer to signal processing and data streaming application, than they are to control-oriented business workflow applications.

The Kepler system aims to provide scientists will a workflow design package on top of the mature Ptolemy II system. Kepler reuses the GUI and the underlying dataflow model from Ptolemy, but adds several features specific to scientific dataflow applications. These workflows are based on actor-oriented design, where individual steps are represented by

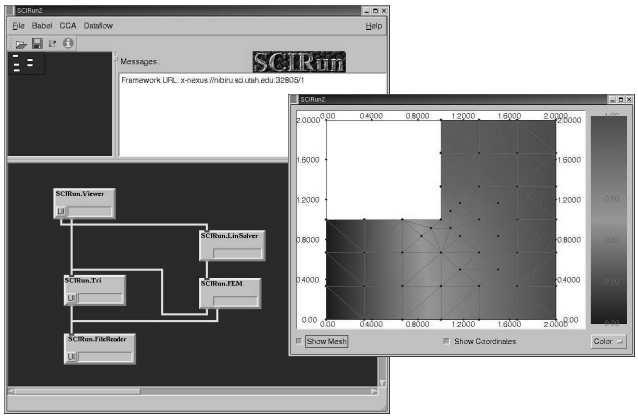

Figure 1. The SCIJump framework provides a GUI and many services to CCA component assemblies.

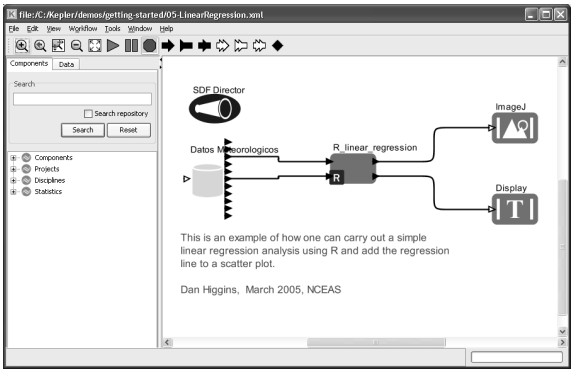

Figure 2. The Kepler system for creating scientific workflows from dataflow composed actors.

actors. Actors have multiple input and output ports which communicate tokens between two connected actors. Also, actors may define parameters which specify the behaviour of an execution step. Directors are instantiated in Kepler to specify a model of execution, such as synchronous dataflow or process networks where each actor will begin in a separate thread. A wide selection of actors is available ranging from general-purpose to ones specific to (e.g. geology, computational chemistry) applications. A simple Kepler workflow that performs linear regression is given in Figure 2.

3. Related Work

The ICENI [7] architecture provides both spatial and temporal composition in an environment that supports scientific application. Unlike our solution, these are two views of the same system and not two different systems that are connected in a hierarchical fashion.

An earlier paper by Lu et al. [6] defines a JobProxy and JobFactory web services and corresponding actors in Kepler to control the CCA framework CCAFEINE. Like their work, our approach uses workflows as higher level control of lower level component simulations. However, our goal differs significantly in that we want to provide a finer-grained approach to communication between components and workflows, while the communication that Lu et al. provide is very coarse-grained. Their approach uses a job mechanism to control the a componentized simulation and provides the ability to expose one CCA port type in Kepler as a web service to start the simulation. The job abstraction actors provided in Kepler for interoperability with components by Lu et al. partially duplicate the functionality of a similar set of general job actors available in Kepler. Our choice is to allow application designers finer level of control if they need it and allow a wider set of use scenarios.

In addition, Lu et al. choose to translate CCA's Scientific Interface Definition Language (SIDL) into the Web Service Definition Language (WSDL) that workflows use in order to use web service protocols for communication between the two frameworks. This translation presents a unnecessary level of complexity. We use a more straightforward and efficient approach that does no translation and commu-

nicates "natively" through the RMI protocol defined by the Babel SIDL compiler.

4. Design

As we have already mentioned, a Kepler workflow is very different from a CCA component assembly. Kepler's actors are composed in a dataflow fashion, representing a temporal relationship (e.g. execute actor X after actor Y finishes). Components are composed via ports and represent a spatial relationship (e.g. component X executes component Y and sends results to component Z). The spatial composition is more powerful, enabling more complex relationships, while the dataflow models is simpler, easier to debug and measure performance. Workflows are limited in expressing control-flow semantics such as webservice retry [2]. Basically, everything that has a if-statement like semantics requires a needlessly complicated workflow, which is inconvenient to write as well as read. Since CCA components are very good at control-flow, combining the two technologies may result in more expressive and better workflows.

The applications written for components are much lower level than those for workflows. The dataflow scheme of the workflows lends itself to defining a high level application flow, and the user community supports this momentum. Based on the above observation, we posit that creating workflow actors that are able to communicate to components is a reasonable way to combine these units into one application. We provide a few actors in Kepler that would enable a scientific workflow to manage and execute a Common Component Architecture (CCA) component application. For instance, an application of this kind would use preexisting Kepler actors to fetch the input data and begin the component framework, and use the "componentized" actors to setup and execute a component simulation. All the while another actor that we provide would be used to monitor the simulation's progress from Kepler, and communicate back to CCA to adjust the simulation's parameters. The task of componentizing actors is not practically difficult. To communicate to a component service that is defined by a specific interface, an actor needs to contain the client stub code. By using the Babel compiler, we are able to do this task easily, and use the distributed capabilities of RMI to communicate across machine domains.

By using this new set of "componentized" Kepler actors, one can communicate with any CCA framework that is specification compliant and exposes its framework services for RMI access. Through the interfaces provided by the services, we control various aspects of the framework (e.g. component instantiation, component connection, events reporting various messages etc.). Using a special actor that is able to gain control of the CCA framework and manage the creation and connection of components a workflow can initialize a component application. Once a set of components have been instantiated and connected, they need to be executed. For this purpose, we designed an actor that is able to bind to any component and execute methods of its interface dynamically, provided that its given the method name, parameters, port name and component name. We can monitor the state of the running applicationq by listening to events sent by the components via the framework's event service. Figure 3 gives a graphical display of the communication pattern between the Kepler actors we introduced and a CCA framework and components. We identify three actors that are required in order to achieve interesting application scenarios: one that is able to initialize the components, one able to control and begin the execution of the components, and one that is able to communicate status messages and results. Two of these communicate with the component framework's services and one of them communicates directly to an individual component's port: *CCAEventListenerActor, SCIJumpOpenNetworkActor, and CCAInvokePortMethodActor*. Other component-friendly actors may be useful to be added in the future, but these three define the core for many applications.

4.1. CCAEventListenerActor

The CCA specification defines an event service to be implemented by specification-compliant component frameworks in order to allow communication of status messages and various other information from any component to any one or more listening components. The event service uses a publish and subscribe mechanism. A topic abstraction is used to define a specific communication channel. The *CCAEventListenerActor* is able to subscribe to specific topic(s) of the CCA event service and execute when it reads any messages. The actor uses Babel RMI as the underlying protocol to directly communicate to the framework implementation. In our prototype example, we use this actor to communicate from the CCA realm back to the workflow in order to report the result of the component computation.

4.2. SCIJumpOpenNetworkActor

This actor is used to open a saved network of CCA components, ports and connections, which is accomplished through an open network service provided by the SCIJump component framework. This service is not yet standardized by the CCA. However, the *BuilderService*, a similar service which is a core part of the CCA standard, could be exposed in order to accomplish the same task as the open network service. However, using the BuilderService would be a more tedious task, requiring that each individual instantiated component and connection to be defined. The

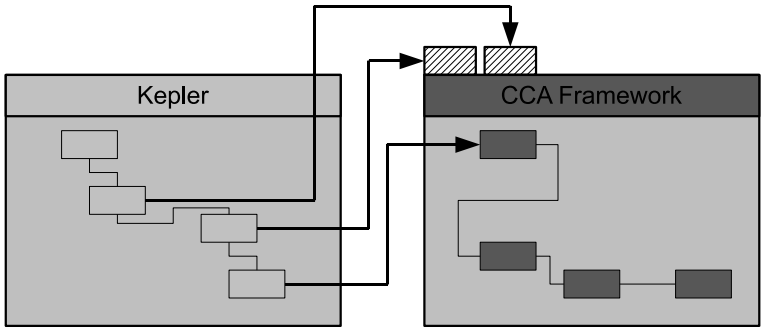

Figure 3. Our design uses "special" Kepler actors that are able to communicate with the CCA framework as well as with individual components facilitating both a course and fine grained interoperability.

SCIJumpOpenNetworkActor loads a file that defines all of this information; creating this file can be done through the SCIJump GUI.

4.3. CCAInvokePortMethodActor

This actor is different from the previous two in that it does not communicate to a service provided by the CCA framework, but directly to an instantiated CCA component, which allows finer grained communication between components and workflows. To directly communicate with an individual component, one needs to invoke a method on a component's port. In separate applications, this is usually a different port and a different method with different arguments. This is a difficult task in practice, as we are adapting to each component dynamically and not relying on predefined stubs compiled with the actor. We designed this actor with this generality in mind. It relies on reflection mechanisms heavily to discover whether ports and methods with the name the user has specified exists and then invokes them passing user specified arguments. We expect that multiple instances of this actor would exist in Kepler when interopeability with components is needed.

5. Results

Using the CCA actors in Kepler described in the previous section, we designed a proof of concept application. This hybrid application uses Kepler and the SCIJump frameworks to start and execute the CCA tutorial application. Figure 4 shows the Kepler workflow that integrates the three "componentized" actor that we defined. The tutoral application contains a few components that use Monte Carlo integration to calculate an estimate for the number PI. Pre-existing Kepler actors connect (using ssh) to a remote machine containing the SCIJump executable and start the framework. We use the *SCIJumpOpenNetworkActor* to load the components belonging to our tutorial application. After the loading has finished we use the *CCAInvokePortMethod* actor to begin the simulation by discovering the starting port. The simulation is independent after this step and we use the *CCAEventServiceListener* to communicate various status messages as well as the final result.

6. Conclusions and Future Work

This work presents a method of designing hybrid scientific applications composed of both workflow actors and components. Because each of these technologies provides a different kind of decomposition this apprach is useful for new application design. Since workflows are intended to be coarser grained then components we design a hierarchical interoperability scheme, by adding actors that are able to communicate with components. This communication is made easier by the advent of RMI in the Babel compiler used heavily in scientific component technology. We designed three actors that are necessary for a basic hybrid application: one that is able to initialize the components, one able to control and begin the execution of the components, and one that is able to communicate status messages and results.

The future of this project is to create a wider array of interesting hybrid applications and add a few more actors that may be useful to application designers. We already have several potential candidates in mind: a *CCAEventPublisherActor* would be able to send status messages from Kepler to the CCA framework, and a *CCABuilderServiceActor* would generalize the component instantiation task from SCIJump to all CCA frameworks.

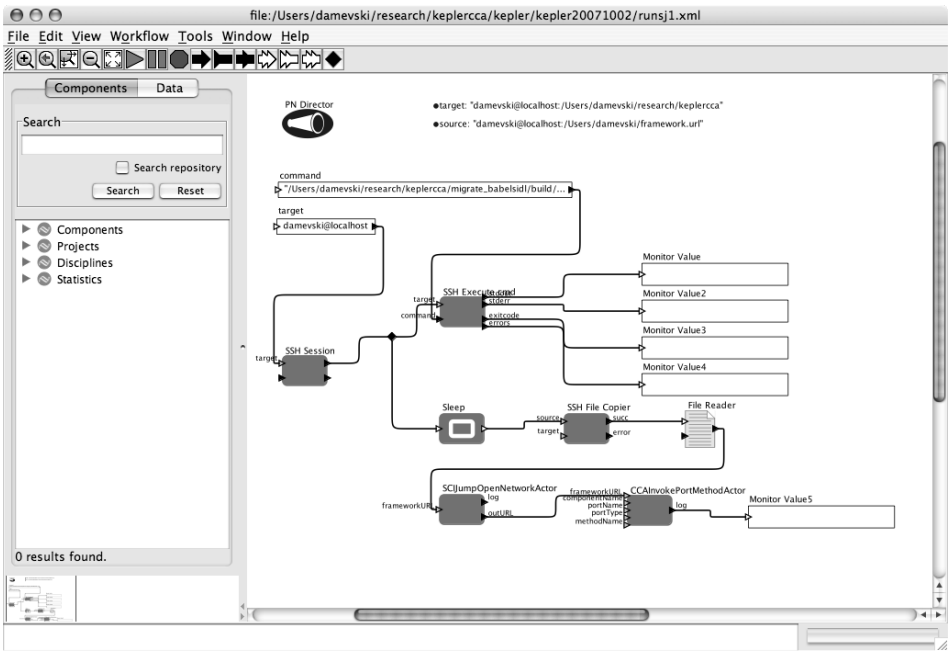

Figure 4. Our proof of concept example Kepler workflow, which invokes the CCA tutorial application running in SCIJump.

References

[1] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock. Kepler: An extensible system for design and execution of scientific workflows. *International Conference on Scientific and Statistical Database Management*, 00:423, 2004.

[2] S. Bowers, B. Ludascher, A. H. H. Ngu, and T. Critchlow. Enabling scientific workflow reuse through structured composition of dataflow and control-flow. In *ICDEW '06: Proceedings of the 22nd International Conference on Data Engineering Workshops*, page 70, 2006.

[3] Common Component Architecture Forum. see `www.cca-forum.org`.

[4] C. Johnson and S. Parker. The SCIRun Parallel Scientific Compouting Problem Solving Enviroment. In *Proceedings of the 9th SIAM Conference on Parallel Processing for Scientific Computing*, 1999.

[5] S. Kohn, G. Kumfert, J. Painter, and C. Ribbens. Divorcing language dependencies from a scientific software library. In *Proceedings of the 10th SIAM Conference on Parallel Processing*, Portsmouth, VA, March 2001.

[6] W. Lu, K. Chiu, S. Shirasuna, , and D. Gannon. A hybrid decomposition scheme for building scientific workflows. *Proceedings of High Performance Computing Symposium (HPC 2007)*, Norfolk, Virginia, March 25-29, 2007.

[7] A. Mayer, S. McGough, N. Furmento, W. Lee, S. Newhouse, and J. Darlington. Iceni dataflow and workflow: Composition and scheduling in space and time. In *UK e-Science All Hands Meeting*, Nottingham, UK, 2003.

[8] K. Zhang, K. Damevski, V. Venkatachalapathy, and S. Parker. SCIRun2: A CCA framework for high performance computing. In *Proceedings of The 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments*, April 2004.