Flexible Scientific Workflow Modeling Using Frames, Templates, and Dynamic Embedding*

Anne H.H. Ngu[1], Shawn Bowers[2], Nicholas Haasch[1], Timothy McPhillips[2], and Terence Critchlow[3]

[1] Texas State University, San Marcos, TX
[2] Genome Center, University of California, Davis, CA
[3] Pacific Northwest National Laboratory

Abstract. While most scientific workflows systems are based on dataflow, some amount of control-flow modeling is often necessary for engineering fault-tolerant, robust, and adaptive workflows. However, control-flow modeling within dataflow often results in workflow specifications that are hard to comprehend, reuse, and maintain. We describe new modeling constructs to address these issues that provide a structured approach for modeling control-flow within scientific workflows, and discuss their implementation within the Kepler scientific workflow system.

1 Introduction

Scientific workflow systems aim to provide end-to-end frameworks for automating and simplifying data processing tasks. These tasks often include data acquisition, transformation, integration, analysis, and visualization. Many existing scientific workflow systems (e.g., KEPLER [1], Taverna [2], and SCIRun [3]) are based on *dataflow* models of computation [4], where individual components (*actors* in KEPLER) are loosely coupled, communicate via streams of data objects, and are scheduled by the workflow system according to dataflow dependencies. An advantage of this approach is that actors, which may be native to the system or wrap external software components such as web services, scripts, or external applications, can become reusable components for use within multiple workflows. Because of the emphasis on data dependencies, these systems also provide a simple and intuitive model for scientific workflow designers [5].

However, while dataflow has become a standard model of computation in scientific workflow systems, control-flow modeling is often necessary for engineering fault-tolerant, robust, and adaptive workflows. Without mechanisms to control the scheduling and execution of actors, otherwise simple workflows quickly become hard to comprehend, reuse, and maintain. In KEPLER, e.g., special-purpose control-flow actors are often introduced into workflows for this purpose, which often lead to complex workflows with many of the above problems (*e.g.*, see Figure 1).

This paper describes a *structured* approach for introducing control-flow into dataflow-oriented scientific workflows. This work is based on our experiences developing workflows for a range of domains, including astrophysics, environmental monitoring, phylogenetics, and bioinformatics. These workflows typically involve generic

* This work supported in part through NSF grants DBI-0533368, IIS-0612326, IIS-0630033, and OCI-0722079.

B. Ludäscher and Nikos Mamoulis (Eds.): SSDBM 2008, LNCS 5069, pp. 566–572, 2008.
© Springer-Verlag Berlin Heidelberg 2008

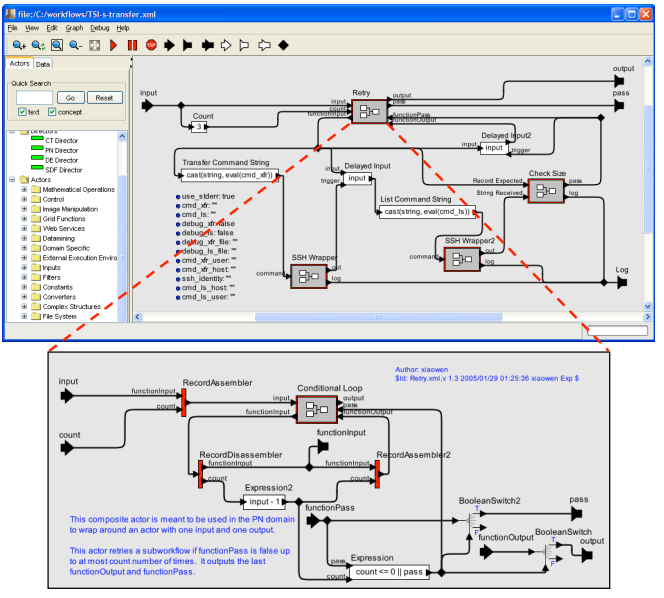

Fig. 1. Control-flow intensive workflow in KEPLER. "Retry," a composite actor for fault-tolerant data transfer (top), contains a subworkflow (bottom), which itself contains a "ConditionalLoop" subworkflow (not shown). Complex feedback loops and low-level actors demonstrate complexity of modeling control-flow using dataflow constructs.

processing steps related to submitting and running external jobs, transferring and processing data, and visualizing results. Each generic processing step typically has many different concrete implementations, where the desired implementation depends on the data input to the workflow, on the type of analysis being performed, or even on the state of the workflow environment at the time of execution.

Our approach is based on workflow *frames* and *templates* [6], which provide an abstraction for modeling control-flow issues surrounding the selection of concrete actor implementations among multiple alternatives. An advantage of our approach is that dataflow remains the primary model of scientific workflows, while allowing complex control-flow specifications to be embedded as subtasks or wrappers around existing actors. We also present an approach (extending [6]) in which the complete specification of frames is determined at workflow execution time through *dynamic embedding*.

2 The KEPLER Scientific Workflow System

KEPLER provides support for designing and executing scientific workflows. KEPLER workflows are created by selecting actors and wiring them together on a design canvas to form the desired workflow graph (Figure 1). Actors have input and output *ports* for communicating with other actors. Data from one actor is streamed asynchronously to another actor via data *tokens*. Composite actors encapsulate subworkflows, e.g., allowing one workflow to be reused in another. Actors typically have *parameter* ports

for configuring default behavior. The overall execution of a workflow is *not* defined by actors within KEPLER, but is factored out into a separate component called a *director* [7]. Thus, different execution models may be used for a workflow, where a different director may be used at different hierarchical composition levels.

The primary mechanisms currently provided by KEPLER for managing control-flow include: (a) allowing actors to have multiple ports, which provides a mechanism for passing control-tokens between actors; (b) low-level and specialized actors for handling control tasks, *e.g.*, to disassemble and reassemble complex data types (such as records), and "Boolean-switch" actors to fork token streams; and (c) allowing complex workflow graph structures that contain cycles, multiple paths, etc. As described in [6,8] (and demonstrated in Figure 1), modeling control-flow using these constructs involves inserting and linking low-level and specialized actors alongside dataflow actors, increasing the complexity of the original workflow and making it difficult to distinguish dataflow (or "scientific" tasks) from control-flow (since they are "entangled").

3 Enabling Flexible Scientific Workflows

Frames and templates decouple control-flow from dataflow by introducing new abstractions: A *frame* wraps a set of alternative actor implementations, and a *template* specifies a subworkflow with "holes" that can be filled in at design time or runtime with actors or additional templates. We adopt and modify the finite-state transducer framework of [7] for specifying templates, which provides an intuitive language for encapsulating control-flow behavior within KEPLER. This approach allows workflow designers to change control-flow behavior by selecting and applying different templates and frames.

In [6], the specification of control-flow behavior using frames and templates is performed exclusively at workflow design time. For example, once a particular template is configured, it is not modified again regardless of changing runtime conditions (although the template may be specified to react to pre-specified runtime conditions). Further, once a frame or template has been embedded, different implementations cannot be re-selected for embedding at run time, unless this behavior is explicitly encoded within the template. *Dynamic embedding* extends the approach by allowing actors and control-flow behavior to be selected at workflow runtime. The rest of this section describes frames, templates, and dynamic embedding.

Actor Frames. Actors in KEPLER are always *concrete* in that they correspond to particular implementations that are directly executed in a workflow. As a simple example, a `gridftp` and a `sftp` (secure ftp) actor are tied to two different data-transfer implementations. A *frame* denotes a set of alternative actor implementations (or refinements) with similar, but not necessarily identical functionality. For workflow designers, frames are placeholders for actors that will be instantiated and specialized at runtime. Thus, a designer can place a frame on the design canvas, and connect it with other workflow components, without prematurely specifying which component is to be used. For actor developers, frames can be used as abstractions for a family of components with similar function, e.g., a `DataTransfer` frame can generalize the transfer of data without specifying whether the implementation is provided by `gridftp` or `sftp`.

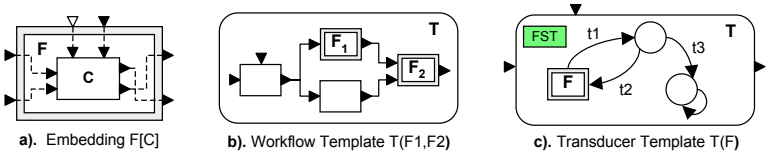

Fig. 2. (a) Embedding of component C in frame F; (b) worfklow template $T(F_1, F_2)$; (c) finite state transducer template $T(F)$

A frame is a named entity F that acts as a placeholder for a component to be "plugged into" F (see Figure 2). When devising a frame F, a family of components $\mathbf{C}_F$ is envisioned, with each $C \in \mathbf{C}_F$ being a possible alternative for embedding into F. As with actors, frames have ports and structural types, which together form the *frame signature* Σ_F. This signature represents the common API of the family $\mathbf{C}_F$ of components that F abstracts. An *embedding* $F[C]$ of a component C into a frame F is a set of pairs associating (or "wiring") ports of C with ports of F. The embedded component C may also introduce new ports not in F; and an embedding $F[C]$ may not use all the ports of C. Parameter ports of F can also be connected to input ports of C and vice versa; however, other connection types are generally not allowed.

Workflow Templates. A frame F imposes constraints on its set of components $\mathbf{C}_F$ such that embeddings $F[C]$ should be well-formed and well-typed for any $C \in \mathbf{C}_F$. However, no assumptions can be made about the "inner workings" of C. A *workflow template* T provides a similar level of abstraction for a set of workflows $\mathbf{W}_T$. Unlike a frame, a template T (partially) specifies the behavior of the workflows it represents. In addition, a template includes an "inner" workflow graph W_T, where some of the components of W_T are not concrete actors, but frames (Figure 2b). Let $F_1, \ldots, F_n$ be the frames that occur in W_T, either directly, or indirectly through nested templates. We can view T as a partial workflow specification $T(F_1, \ldots, F_n)$, whose frames F_i can be independently specialized by embedded components (actors or templates) C_i. The resulting embedding $T(F_1[C_1], \ldots, F_n[C_n])$ is a concrete, executable workflow if no C_i has a frame; otherwise the embedding is a (more refined) template. The left diagram in Figure 3 shows an example data transfer frame embedded with a "retry" transducer and example state implementation.

Dynamic Embedding. Because frame and template embeddings are given at design time, they require all paths within a workflow to be completely bound and loaded prior to execution. In *dynamic embedding*, frames are embedded and instantiated on demand during workflow execution, which unlike static embedding, does not require the workflow system to load or manage the alternative implementations of a frame. Instead, alternatives are dynamically selected according to rules employed by the frame itself. Dynamic embedding is used when selection rules change often or are complex, and when deploying large workflows with many frames and alternative implementations.

Our implementation of dynamic embedding consists of: (1) waiting for data tokens to arrive at the frame's input ports, (2) selecting an embedded component using a set of selection criteria (given by the workflow designer), (3) transferring input tokens from the

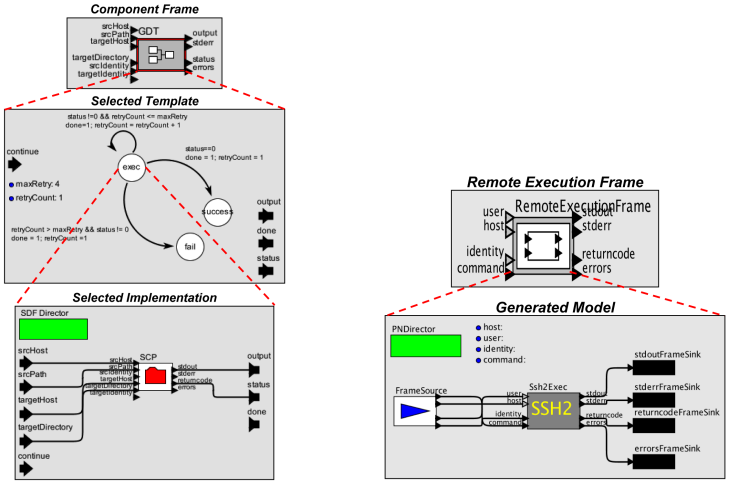

Fig. 3. A generic data transfer component statically embedded with a specific template and underlying state frame implementation (left), and a frame after being embedded dynamically (right)

frame to the embedded component, (4) automatic construction of an internal workflow to run the embedded component (with the appropriate director), (5) executing the constructed workflow, and (6) transferring output tokens from the embedded component to the frame actor. The right of Figure 3 shows a remote job-execution frame implemented via dynamic embedding. As the workflow executes, the frame selects the appropriate concrete actor based on given selection criteria. Also shown in Figure 3 (bottom, right) is one of the automatically generated internal workflows, which consists of a dataflow director, the selected actor, the source and sink actors for controlling input and output, and the workflow parameters.

Frames that employ dynamic embedding are implemented via higher-order actors, which invoke actors or subworkflows given as input. These frames configure subworkflows according to selection criteria and use higher-order actors to control their execution. The frames also use additional actors to mediate the flow of tokens to the underlying frame embeddings. The *select actor* implements the selection policies of the frame (given as rules based on runtime conditions and input tokens), and supplies the embedding. The *source actors* transfer input tokens from the frame actor to the selected actor. The *sink actors* transfer output tokens from the selected actor to the frame actor. Finally, a *port wiring* component is used to map ports and parameters of the selected actor to ports and parameters of the frame.

Implementation. We have developed a prototype implementation of dynamic embedding within KEPLER and have applied the approach to example workflows, including the Terascale Supernova Initiative [9]. This workflow, in particular, was developed to automate the repetitive and complex data transfer and monitoring tasks involved in running a supercomputer simulation from a user's local computer. We have also developed examples of frame-based workflows for inferring phylogenetic trees that select the

appropriate implementations of actors (*e.g.*, for file conversion and tree inference) based on the type of input data provided to the workflow.

4 Summary and Related Work

Frames, templates, and dynamic embedding enable workflow designers to more easily specify control-flow tasks needed for fault-tolerant, reusable, and adaptive scientific workflows, while still supporting dataflow as the primary model of computation. Most scientific workflow systems are based on dataflow, as opposed to business workflow systems [10] and associated approaches (e.g., workflow patterns [11] and web-service composition [12,13]) that use control-based models such as Petri nets. A significant challenge in both fields is to seamlessly integrate control-flow *and* dataflow within a single model. Frames and templates are inspired by hierarchical finite state machines [14] and the nesting of heterogeneous computation models [15]. Our approach also extends adaptive workflow modeling [16] by supporting complex data and control structures. In future work, we will further explore dynamic embedding for designing and analyzing workflows, e.g., to automatically compose specifications of dynamically-embedded components and discover suitable actors for use within frames and templates.

References

1. Ludäscher, B., et al.: Scientific workflow management and the Kepler system. Concurrency and Computation: Practice & Experience (2006)
2. Oinn, T., et al.: Taverna: A tool for the composition and enactment of bioinformatics workflows. Bioinformatics 20 (2004)
3. Parker, S.G., Miller, M., Hansen, C.D., Johnson, C.R.: An integrated problem solving environment: The SCIRun computational steering system. In: HICSS (1998)
4. Lee, E.A., Parks, T.M.: Dataflow process networks. Proc. of the IEEE 83, 773–801 (1995)
5. Bowers, S., Ludäscher, B.: Actor-oriented design of scientific workflows. In: ER (2005)
6. Bowers, S., Ludäscher, B., Ngu, A.H.H., Critchlow, T.: Enabling scientific workflow reuse through structured composition of dataflow and control flow. In: IEEE SciFlow (2006)
7. Eker, J., et al.: Taming heterogeneity—The Ptolemy approach. Proc. of the IEEE 91 (2003)
8. Goderis, A., Goble, C., Sattler, U., Lord, P.: Seven bottlenecks to workflow reuse and repurposing. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 323–337. Springer, Heidelberg (2005)
9. Xin, X.: Case study: Terascale supernova initiative workflow (TSI-Swesty). LLNL Technical Note (2004)
10. Alonso, G., Mohan, C.: Workflow management systems: The next generation of distributed processing tools. In: Advanced Transaction Models and Architectures (1997)
11. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases 14 (2003)
12. Curbera, F., et al.: Business Process Execution Language for Web Services (BPEL4WS), Version 1.0 (2002)
13. Martin, D.L., et al.: Bringing semantics to web services: The OWL-S approach. In: Intl. Workshop on Semantic Web Services and Web Process Composition (2004)

14. Girault, A., Lee, B., Lee, E.A.: Hierarchical finite state machines with multiple concurrency models. IEEE Transactions on CAD 18 (1999)
15. Lee, E.A., Neuendorffer, S.: Actor-oriented models for codesign: Balancing re-use and performance. In: Formal Methods and Models for System Design. Kluwer, Dordrecht (2004)
16. Ngu, A.H.H., et al.: Advanced process-based component integration in Telcordia's cable OSS. In: ICDE (2002)