TEXAS STATE UNIVERSITY SAN MARCOS

Department of Computer Science
San Marcos, TX 78666

Report Number TXSTATE-CS-TR-2007-7

Flexible Scientific Workflows Using Frames and Dynamic Embedding

Anne HH. Ngu
Nicholas Haasch
Timothy McPhilips
Shawn Bowers
Bertram Ludaescher
Terence Critchlow

2007-04-01

Flexible Scientific Workflows using Frames and Dynamic Embedding

Anne H. Ngu, Nicholas Haasch
Department of Computer Science
Texas State University-San Marcos
{angu,nick}@txstate.edu

Timothy McPhilips, Shawn Bowers, Bertram Ludaescher
Department of Computer Science
University of California, Davis
{tmcphilips,sbowers,ludaesch}@ucdavis.edu

Terence Critchlow
Lawrence Livermore National Laboratory
Livermore
critchlow1@llnl.gov

Abstract

Current approach to scientific workflow design in the popular open source Kepler system is based on the actor-oriented framework where concrete actors can be hierarchically composed and orchestrated by different directors (schedulers). A common assumption in this design framework is that workflow is static and must be completely specified before orchestration. Such a static and monolithic workflow cannot response to changing runtime conditions. We present flexible scientific workflow design that allows some tasks to be partially specified via abstract actors called **Frame**. *The behavior of a frame is determined at runtime by the embedded concrete actor. We implemented the process of* **dynamic embedding** *that can tailor to different selection policies and enable automatic construction of subworkflow to execute the embedded component at runtime. Frames and dynamic embedding provide high level abstractions for specifying workflow that enables flexible execution nested to any level. Finally, we illustrate with two distinct scientific workflows from astrophysics and bioinformatics domains the benefit of frames and dynamic embedding.*

1 Introduction

Scientific workflow aims to provide a scientist with infrastructure for automating sequence of related repetitive tasks demanded by their research. SPA/KEPLER$\tilde{\text{i}}$s an open source scientific workflow system that has been used by scientists to construct and execute many complex scientific analyses [15, 1, 25]. Some of the main requirements of SPA/KEPLER Scientific Workflow system are: 1) the ability to compose complex workflows from existing workflows/components with minimal effort, 2) the ability to monitor the workflow execution and to provide automated data managements (e.g. data provenance), 3) the ability to adapt or steer the workflow execution.

In SPA/KEPLER, users develop workflows by selecting appropriate components called *actors* and placing them on a design canvas, after which they can be "wired" together to form the desired workflow graph (*cf.* Figure 1). Actors have input and output ports which provide the communication interface to other actors. Workflows can be hierarchically structured, yielding *composite actors* that encapsulate subworkflows A novel feature of SPA/KEPLER inherited from PTOLEMY II is that the overall execution and component interaction semantics of a workflow is *not* defined by the components, but is factored out into a separate component called a *director*. Taken together, workflows, actors, ports, connections, and directors represent the basic building blocks of *actor-oriented modeling and design* in SPA/KEPLER Scientific Workflow system [13].

Although actor-oriented design principle lends itself to reusable components, many existing scientific workflows in SPA/KEPLER are ”designed to fit” a specific application [25, 19]. A fundamental assumption in these design frameworks is that workflow is static and must be completely specified before orchestration. Such a static and monolithic workflow cannot response to changing runtime conditions.

During the past year, we have investigated two paradigms that allow flexible modeling and composition of scientific workflow such that it enabled re-use and re-purposing. The first approach, frame/template [5], emphasizes on decoupling of control flow from data-flow through structured embedding of control flow actors in data-oriented scientific workflow. We introduce abstract actors called *Frames* and *Templates* that can be configured at design time.

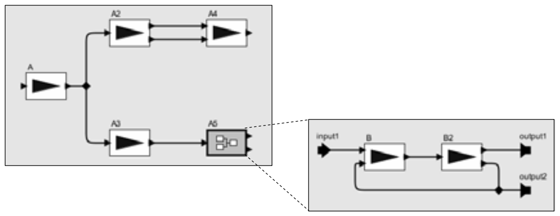


Figure 1. Actor-Oriented Modeling of Workflow

The main contribution of this particular approach is the encapsulation of complex control-flow patterns in standard templates that can be specialized by embedding different frame implementations. Frame/template approach allows workflow designers to change complex control-flow behavior by simply using different templates. This enable composition of scientific workflows that are more robust (common fault-tolerance strategies can be pre-defined by various templates) and at the same time more reusable since the embedding of frames and templates yield more structured and modular workflow designs. The frame/template approach introduces some degree of flexibility in the modeling of workflow at design time, but it does not address the adaptive execution of workflow at runtime. Once a particular template is configured at design time, it cannot be modified regardless of changing runtime conditions. The second problem with template is that, the number of states can increase quickly and become unmanageable when there is an explosive number of control-flow options. Currently, there is no working implementation of frame/template that can be executed with PTOLEMY IIś data process networks director which plays a critical role in executing scientific workflow that requires data streaming and high parallelism.

The second approach that we have experimented is the higher-order actors that allow functional approach to workflow design. These actors include collection-aware actors [18] and list comprehension actors [14]. The goal is to hide the actors that are peripheral to the goal of the workflow. This includes control-flow actors introduced for the sole purpose of routine manipulation of complex scientific data (e.g. collection or list-based input) or for various exception handlings. The collection-aware actors have shown to simplify workflow design (eliminate the use of artificial control flow actors) and allow reuse of the same workflow with varying nested complex data.

While both of the above approaches provide higher level abstractions for encapsulating complex control flows in data-oriented scientific workflow at design time, they do not address the flexible execution and dynamic composition of scientific workflow at runtime. This means it is impossible to steer and adapt workflow execution at runtime in response to availability of new resources. In the case of collection-aware actors, at design time, the designer must know how to model either the collection data or the routine control-flow information in the collection stream. It is not possible to switch or change to a different collection actor when a different type of data is received at runtime . Overall, this implies that workflow must be statically and completely defined prior to orchestration.

In our study of scientific workflows in various domains ranging from astrophysics, environment monitoring, and bioinformatics; there exists a common practice or process in conducting a specific type of scientific investigation. In the astrophysics domain, the process of running a simulation includes the generic tasks such as *submit a job*, *monitor the job*, *transfer data*, *process data*, and *visualize the data*. The entire process could take a few days to complete. There are alternative implementations for each of the above tasks depending on the nature of the experiment and the available resources for conducting the experiments. It is extremely tedious to pre-define all the possible combination of alternatives for all the tasks in this workflow. For example, if there are five possible methods to submit a job and five possible monitoring procedures available, it is necessary to establish 25 different relationships statically in order to cover all the possibilities. Moreover, the resulting workflow with all these possible relationships among all the alternative implementations will have very low re-use and hard to maintain. For example, the incorporation of every additional choice requires manual wiring of multiple ports and parameters and development of ad-hoc control-flow actors. In the case of bioinformatics workflow, the choice of which tool to use for processing the data or visualizing the data is dependent on what type of data is received and what tools are available at runtime. For example, BLAST is typically needed for processing the DNA sequence data. However, new BLAST computational tools are constantly being developed. To pre-select the specific BLAST tool at design time will not take advantage of better available tools.

Our ultimate vision for the specification and the execution of flexible workflows is a software infrastructure that will allow domain scientists to discover existing workflow that is close enough to his/her intended goal, configure it with specific contextual information, run and visualize it. Scientists do not have to be concerned with the concrete specification of every aspects of the workflow. For example, which tools to pick to accomplish a task. The concrete workflow is constructed transparently and dynamically by querying the available actors/components. The myGrid/Taverna workbench [9] is one of the first attempt to leverage semantic web for building a platform for scientific workflow discovery with reuse as the main goal. Our goal here is to provide a framework that facilitates both reuse and flexible execution. For example, when there is a plethora of tools available, a specific tool is selected during runtime to achieve optimal (with respect to specific criteria defined by the scientists) and flexible execution.

The organization and contributions of the paper are as follows: We present an approach to building flexible data-oriented scientific workflows with *Frames* whose complete specification is determined at run-time by the embedded concrete actor/component through a *dynamic embedding process*. At the conceptual level, frame provides an abstraction for encapsulating different implementations of a specific task/tool that can be reused across different scientific workflows. This simplifies the specification of a workflow that requires complex logic for handling different choices and exceptions. Frame, being an actor, can be used by a scientist just like the way a conventional actor can be used. This means frame can be nested and composed with other actors. We implemented the process of *dynamic embedding* of concrete actor/components in KEPLER. This involves automatic construction of a model required to run the embedded component as a sub-workflow at runtime without changing any core infrastructure in KEPLER. As compared to frame/template approach, dynamic embedding has the added advantage of only instantiating the needed component at runtime. A scientific workflow composed with frame actors contain partial specifications that act as placeholders for independently defined subcomponents. The independent subcomponents can be selected based on intelligent brokering, querying a particular repository, using a specific algorithm, or based on a known policies. Such a feature is particularly useful for scientific workflow that have explosive number of options, or scientific workfow that must be altered on the fly to meet changes in experimental conditions or available resources. Our dynamic embedding process is a prototype implementation of our ultimate envisioned flexible scientific workflow.

In section 2, we present the overview of the flexible scientific workflow framework, the concept of *Frames*, the *dynamic embedding process* and the implementation of the frame actor. In section 3, we demonstrate the benefit of using frames and dynamic embedding for modeling and execution of two distinct type of scientific workflows. The first one is the TSI (Terascale Supernova Initiative) workflow from astrophysics. This workflow is developed at LLNL under the Scientific Process Automation (SPA) project. The goal of the workflow is to automate the repetitive tasks involved in running a simulation at a supercomputer and transferring the resulting files onto a mass-storage and then to a local machine for data analysis and visualization. This is a control-flow intensive workflow since many ad-hoc control-flow actors with complex wiring of input and output ports are required for handling various exceptions and robust execution. We will show that *Frames* and *dynamic embedding* provide high level abstractions for specifying TSI workflow that enables flexible execution nested to any level. We also show that frames allow controlled extension of the workflow for adaptation and repurposing by changing just the selection criteria.

The second type of workflow is the NDDP (National Diversity Discovery Project) workflow from bioinformatics domain. NDDP workflow is data-flow intensive in the sense that the data set that passed through the workflow is large, complex and nested in structure. We will show that an NDDP workflow modeled with frames and dynamic embedding can be reused for different phylogenetic data while a customized NDDP workflow is required for each type of phylogenetic data without the frames and dynamic embedding.

Finally, we present our related work in section 4 and conclusion in section 5.

2 Flexible Scientific Workflow

In our current prototype implementation, a flexible scientific workflow is defined as a workflow that allows some tasks to be partially specified by *Frames* (abstract actors). The behavior of a frame is determined at runtime by the embedded concrete actor. Frames can be used by a scientist just like the way conventional actors can be used. This means frames can be reused and nested just like the conventional actors and they can be executed using all the available directors in PTOLEMY II including the process networks (PN) which comes with the benefit of data streaming and highly concurrent execution. Frame, being an actor, shares the usual properties of an actor with additional capabilities such as reasoning over selection policies, matching of ports and parameters of embedded component. We provide the formal definition of frames in the following section.

2.1 Frames

Actors in actor-oriented modeling and design framework such as KEPLER are always *concrete*: they correspond to particular implementations and can be directly executed in a workflow. For example, gridftp and sftp are tied to two different specific implementation of data transfer protocol. We extend actor-oriented modeling with a new entity called *Frame*, which is an abstraction that denotes a set of alternative actor implementations (or refinements) with similar, but not necessarily identical functionality.[1] For workflow designers, frames are placeholders for components that will be instantiated and specialized at runtime. Thus, a designer can place a frame F on the design canvas, and connect it with other workflow components, without prematurely

[1]Here the term frame symbolizes a notion akin to a picture frame—allowing different "pictures" (*i.e.*, components) so long as they conform to the constraints imposed by the "frame."

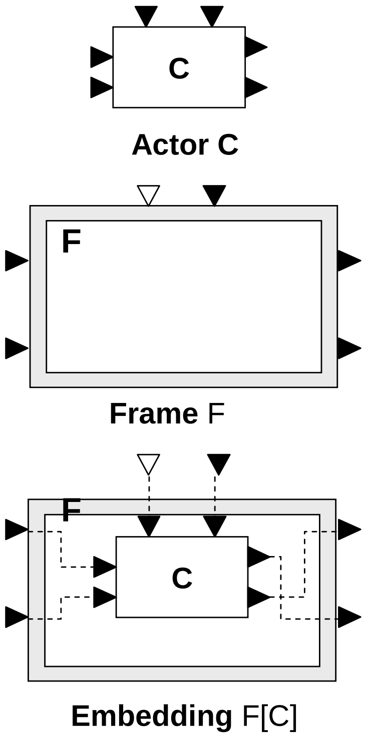


Figure 2. Embedding of an actor C in frame F

specifying which component C is to be used. For component developers, frames can be used as abstractions for a family of components with similar function. For example, we can have a DataTranfer Frame that provides abstraction for transferring of data between computers without having to worry about whether the actual implementation is provided by gripftp or sftp (secure ftp).

Formally, a frame is a named entity F that acts as a placeholder for a component C 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. Like an actor, a frame has input, output, and parameter ports, structural types, and semantic types; taken together they 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, *i.e.*, $F[C] \subseteq \mathsf{ports}(F) \times \mathsf{ports}(C)$. We indicate the wiring type of a pair $(x, y) \in F[C]$ as follows:

- $F.x \blacktriangleright C.y$; if $x \in \mathsf{in}(F), y \in \mathsf{in}(C)$ (*input*)
- $F.x \blacktriangleleft C.y$; if $x \in \mathsf{out}(F), y \in \mathsf{out}(C)$ (*output*)
- $F.x \blacktriangledown C.y$; if $x \in \mathsf{pars}(F), y \in \mathsf{pars}(C)$ (*parameter*)

The embedded component C which is also known as refinement may also introduce new ports not in $\mathsf{ports}(F)$. We denote these ports as $\triangleright C.y$, $\triangleleft C.y$, and $\triangledown C.y$ for input, output, and parameter ports y, respectively.

Similarly, an embedding $F[C]$ may not use all the ports of C. We denote these unused ports as $F.x\triangleleft$, $F.x\triangleright$, and $F.x\triangledown$ for input, output, and parameter ports x, respectively. We note that parameter ports $F.x$ can also be connected to input ports $C.y$ and vice versa. However, other connection types $(x, y) \in F[C]$ are not allowed. Here we assume that all the ports of F are used in the embedding.

An embedding $F[C]$ is *well-formed* if the input and output port directions are observed, *i.e.*, F's inputs (outputs) are wired only to inputs (outputs) of C (Figure 2). A well-formed embedding $F[C]$ is *structurally well-typed* if the structural types align, and *semantically well-typed* if the semantic types align.

2.2 Dynamic Embedding

Frames can be embedded statically or dynamically. In static embedding, a refinement to a frame is embedded during design time. A scientific workflow with static embedding requires all paths to be completely bounded before execution. At initialization time, the director is aware of the complete workflow. The frames are concrete at all stages of workflow execution.

The main problem with static embedding is that if there is a change in runtime condition, the frame cannot be reconfigured at that time to select a different refinement. In dynamic embedding, frames are embedded at runtime (during the execution of the workflow). The refinement to a frame is thus instantiated on demand. This means the system does not have to instantiate anything that will not be used at runtime. This resulted in better utilization of virtual memory and is particularly advantages when there is a need for recursive instantiation of frames.

2.2.1 Dynamic Embedding Process

The process of dynamic embedding of a frame consists of:

- Waiting for the tokens to arrive at the frame's input ports (a typical behavior of any actors).
- Choosing an embedded component using a set of selection criteria.
- Transferring of input tokens from the frame actor to the embedded component.
- Automatic construction of an internal workflow to run the embedded component.
- Running the internal workflow in any PTOLEMY II directors.

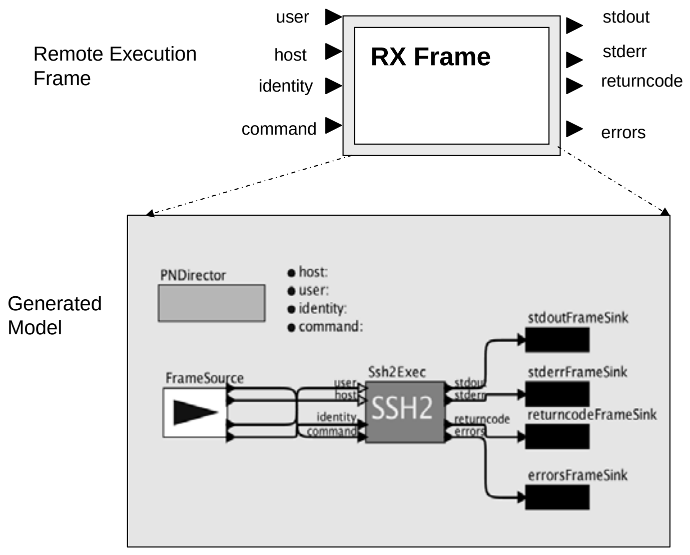


Figure 3. Automatic generated model from dynamic embedding process.

- Transfer of output tokens from the embedded component to the frame actor (another typical behavior of an actor).

Figure 3 shows a remote execution frame (*RX Frame*), a frame designed by a workflow developer to encapsulate different implementations of running a job in a remote computer. At runtime, *RX Frame* is embedded with a concrete actor. The lower pane of Figure 3 shows the automatic generated internal workfkow for that actor. The generated workflow consists of a director (PN), the selected actor (Ssh2Exec, an actor that submits a job via remote login mechanism), the source actor (FrameSource), the sink actors (xxxFrameSink) and the parameters. We discuss in the following section how the internal workflow is generated.

2.2.2 Implementation of Dynamic Frame

In this section, we use the term *Dynamic Frame* to indicate a frame actor that is embedded using the dynamic embedding process. This is to distinguish it from a frame that is being embedded statically.

The existing ModelReference actor in PTOLEMY II is a higher-order actor that can execute a given (pre-constructed) model/workflow provided through its input port [6]. The way it achieves that is by constructing an internal workflow to run the given model during the pre-fire phase. This is done by first transferring the input tokens of the ModelReference actor to the parameters of the same name in the given model (internal workflow). The internal workflow is then executed and the output tokens from the internal workflow is transferred from the parameters of the internal workflow to the the matching output ports of ModelReference actor. The ModelReference actor fits most of the requirement for dynamic embedding process except:

- User must pre-construct the given model (internal workflow is passed as an input parameter)
- The output tokens are transferred only after completion of execution of the internal workflow (i.e. outputs are transferred synchronously).

The implementation of our dynamic frames leverage the capability of ModelReference actor with two main improvements. Frame actor constructs the internal workflow automatically and the output tokens are transferred as soon as they arrive. This asynchronous transfer of output tokens is critical when large amount of scientific data needs to be streamed. Frame actor is implemented as a subclass of ModelReference actor. The additional components needed to support the Frame actor are:

- SelectActor, which implements the selection policies and returns a refinement. The refinement that is returned gets automatically embedded. Figure 4 shows a simple implementation of the SelectActor for choosing between *Web service* verses *Ssh2Exec*(Secure Shell) for the implementation of remote job execution.

```
String selectedActor;
selectActor()
    if (testWebService())
       selectedActor = ”WebServiceExec”;
       return getWebServiceActor()
    else
       selectedActor= ”ssh2Exec”;
       return getSSH2ExecActor()
```

Figure 4. RX Frame:selectActor()

We expect the selection policies will be provided by the domain scientists. In this prototype implementation, the selectActor is a set of simple if-else statements. However, in our ultimate vision of flexible scientific workflow, selectActor can be an intelligent agent that possess reasoning capabilities that will accept a query with scientists’ preferences and quality constraints and return the highest ranked component that can be embedded.

- FrameSourceActor transfers the input tokens from the frame actor to the selected actor.
- FrameSinkActor transfers the output tokens from the selected actor to the frame actor,
- PortWiring maps the ports and parameters of the selected actor to the ports and parameters of the frame

actor. Figure 5 shows the port wiring for remote execution frame that can be embedded with either *Web service* or *Ssh2Exec*.

```
getInputPortMapping()
    if (selectedActor == ”ssh2Exec”)
        return {{”hostname”, ”hostname”}
            {”command”, ”cmd”}}
    else if (selectedActor == ”WebserviceExec”
        return {{”hostname”, ”url”}
            {”command”, ”method”}}
```

Figure 5. RX Frame:portWiring()

The current prototype implementation of dynamic embedding process only performs syntactic mapping of ports which is expressed as a list containing pairs of strings. We anticipate to use KEPLER’s semantic type annotation to automatically map the ports and parameters in the future.

The above components/classes (Frame, SelectActor, FrameSourceActor, FrameSinkActor, PortWiring), form the foundation classes for dynamic frame. The development of a new frame actor is a matter of creating a subclass of Frame actor and implement the SelectActor and PortWiring classes respectively.

3 Case Studies

3.1 TSI Workflow

The goal of TSI workflow is to automate the repetitive tasks involved in running a simulation at a remote supercomputing computer, and transferring the resulting data files onto a mass-storage and to a local machine for analysis and visualization. The process consists of first logging into the supercomputer, submitting the job, monitoring the status of the submitted job. As soon as the simulation starts running and generates output files, those files must be moved to the mass-storage at regular interval to prevent overflowing of the temporary scratch space at the supercomputer. Finally, all the output files are transferred from mass-storage onto a local machine for analysis and post-processing. Currently, there are three different variations of TSI workflow as shown in Figure 6.

Each TSI workflow has an actor for *submit job, transfer files* and *post processing*. The submit job actor in workflow A is invoking a Web service to accomplish the remote job submission, the submit job actor in workflow B is using secure shell actor which requires remote login to the supercomputer and submit the job using a local command. The submit job actor in workflow C uses a wrapper around secure shell actor to pre-process the job submission command. Each of the submit job actor is coded specifically for its specific workflow. They cannot be reused across the three TSI workflows.

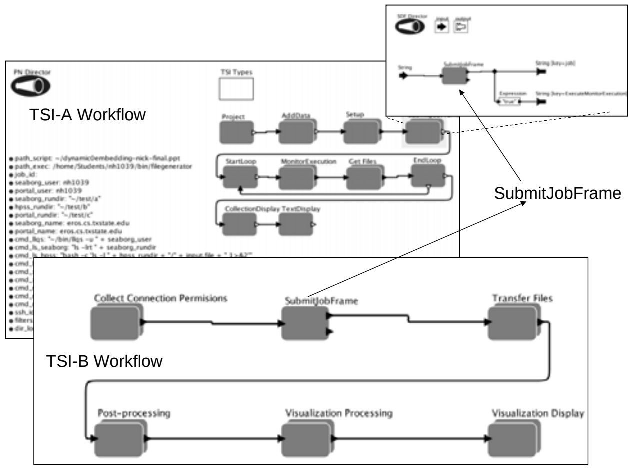


Figure 7. TSI-A and TSI-B workflows with frame actors

With frame and dynamic embedding, scientist only needs to know how to use and maintain one version of submit job frame/actor which can switch among different implementations of remote job execution.

Figure 7 shows the TSI-A workflow [2] with such a submit job frame. The same submit job frame is used in TSI-B as shown in the lower pane of the same figure. The submit job frame in turns can invoke RemoteExecution frame (cf. Figure 8) to switch among different methods for submitting a job depends on runtime conditions or established policies. For example, if at runtime, it detects that the workflow and the simulation experiment are being run on the same computer, it is possible to switch to run the job locally without having to do a remote login. Moreover, no changes need to be make to either of the two TSI workflows to enable this option. This example also demonstrates the nested invocation of *RemoteExecutionFrame* from *SubmitJobFrame*.

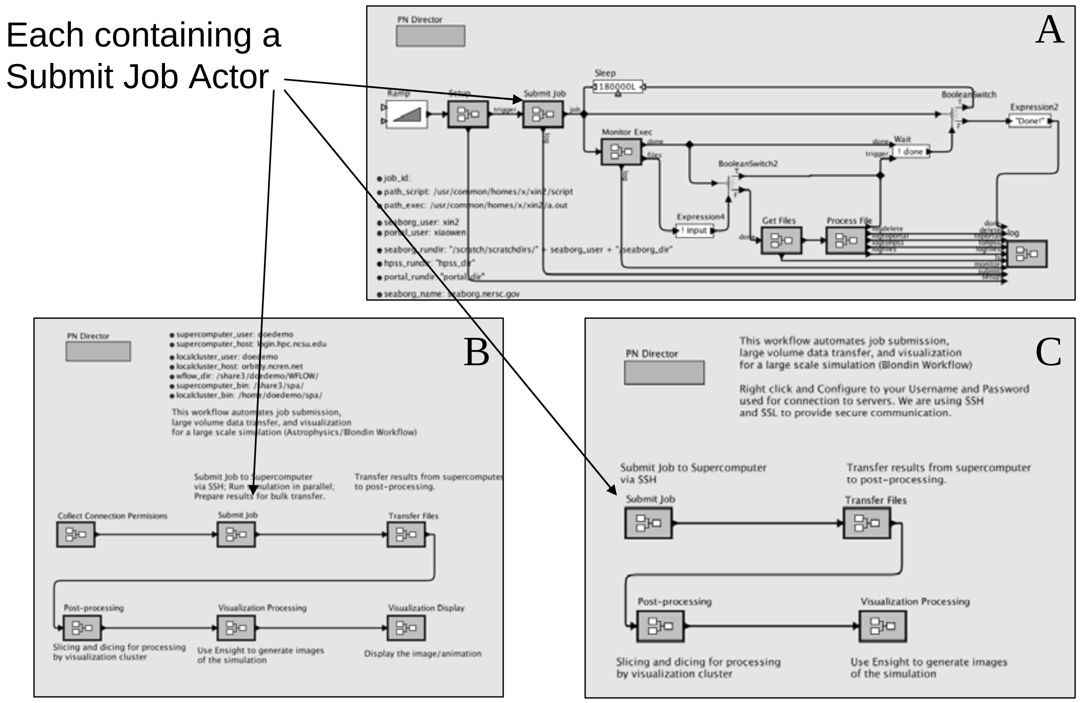
3.2 NDDP Workflow

The goal of NDDP workflow is to ..... Need to discuss the benefit of NDDP workflow with frames

4 Related Work

The need for adaptive and reusable workflows has been an active research issue in business workflow systems since the late 90’s [10, 24, 16]. To achieve adaptability, [12]

[2]Note that the new TSI-A workflow is implemented using the collection-oriented framework in KEPLER, thus it has simpler structure and used different icon shape to respresent the actor.



A:

- job_id:
- path_script: /usr/common/homes/x/xin2/script
- path_exec: /usr/common/homes/x/xin2/a.out
- seaborg_user: xin2
- portal_user: xiaowen
- seaborg_rundir: "/scratch/scratchdirs/" + seaborg_user + "/seaborg_dir"
- hpss_rundir: "hpss_dir"
- portal_rundir: "portal_dir"
- seaborg_name: seaborg.nersc.gov

B:

- supercomputer_user: doedemo
- supercomputer_host: login.hpc.ncsu.edu
- localcluster_user: doedemo
- localcluster_host: orbity.ncren.net
- wflow_dir: /share3/doedemo/WFLOW/
- supercomputer_bin: /share3/spa/
- localcluster_bin: /home/doedemo/spa/

This workflow automates job submission, large volume data transfer, and visualization for a large scale simulation (Astrophysics/Blondin Workflow)

Submit Job to Supercomputer via SSH; Run simulation in parallel; Prepare results for bulk transfer.

Transfer results from supercomputer to post-processing.

Slicing and dicing for processing by visualization cluster

Use Ensight to generate images of the simulation

Display the image/animation

C:

This workflow automates job submission, large volume data transfer, and visualization for a large scale simulation (Blondin Workflow)

Right click and Configure to your Username and Password used for connection to servers. We are using SSH and SSL to provide secure communication.

Submit Job to Supercomputer via SSH

Transfer results from supercomputer to post-processing.

Slicing and dicing for processing by visualization cluster

Use Ensight to generate images of the simulation

Figure 6. Different versions of TSI workflows.

proposed decoupling of conceptual workflow specifications (orchestration) in WSFL from the individual tasks (TSL) that make up a workflow. A task can thus be reused in different workflows. An activity in WSFL specification can choose to use a specific task implementation.

CMI [23, 21], which is an advanced business workflow prototype, introduced the concept of place holder activity. CMI's placeholder activity allows complex runtime decision to be modeled without having to exhaustively list all the potential activities within a process. The use of placeholder activities has enabled the construction of a complex and adaptive workflow at a much faster pace as demonstrated in [21]. Our Frames is analogous to the placeholder activities albeit with significant differences in the embedding process due to the use of actor-oriented modeling framework.

Process-based web service composition [8, 3, 11, 17] also support the flow specifications separately from the underling component services in an attempt to generate dynamic processes to support agile business collaboration. The main challenge is to be able to perform dynamic service aggregation at runtime to achieve a particular goal. The problem of dynamic web service composition is addressed generally using semantic web [4, 20, 2, 22] and the related inference engines. However, such an approach is yet to automatically generate a workflow that has complex control structures involving loops, nondeterminism and choices which are fundamental to control-flow intensive scientific workflow.

In PTOLEMY II there is the concept of mutation described in Chapter 8 of PTOLEMY II's volume III design document [7] that enables changes in workflow's structure at runtime. However, mutation causes the workflow to stop for an indeterminate amount of time. This introduces side effects. For example if we had an actor downloading a huge file and another actor requests a mutation, all actors in the workflow will be blocked until the mutation has completed. In PTOLEMY II's PN domain, there is no determinate point where mutations can occur other than a real deadlock. The concept of mutation is also too low level for the purpose of modeling and design of actors that is reusable across different workflows.

5 Conclusion

Most scientists are not expert programmers. Without the support of flexible scientific workflow specification, whenever an existing workflow needs to be used for a different data set or run in a different environmental setup, a new version of the same workflow must be created [9]. This resulted in an explosive number of similar workflows which is hard to manage and maintain. We aim at providing a high-level abstraction for flexible scientific workflow specification and yet still operate within a rich process model that allows hierarchical composition of actors with complex selection criteria.

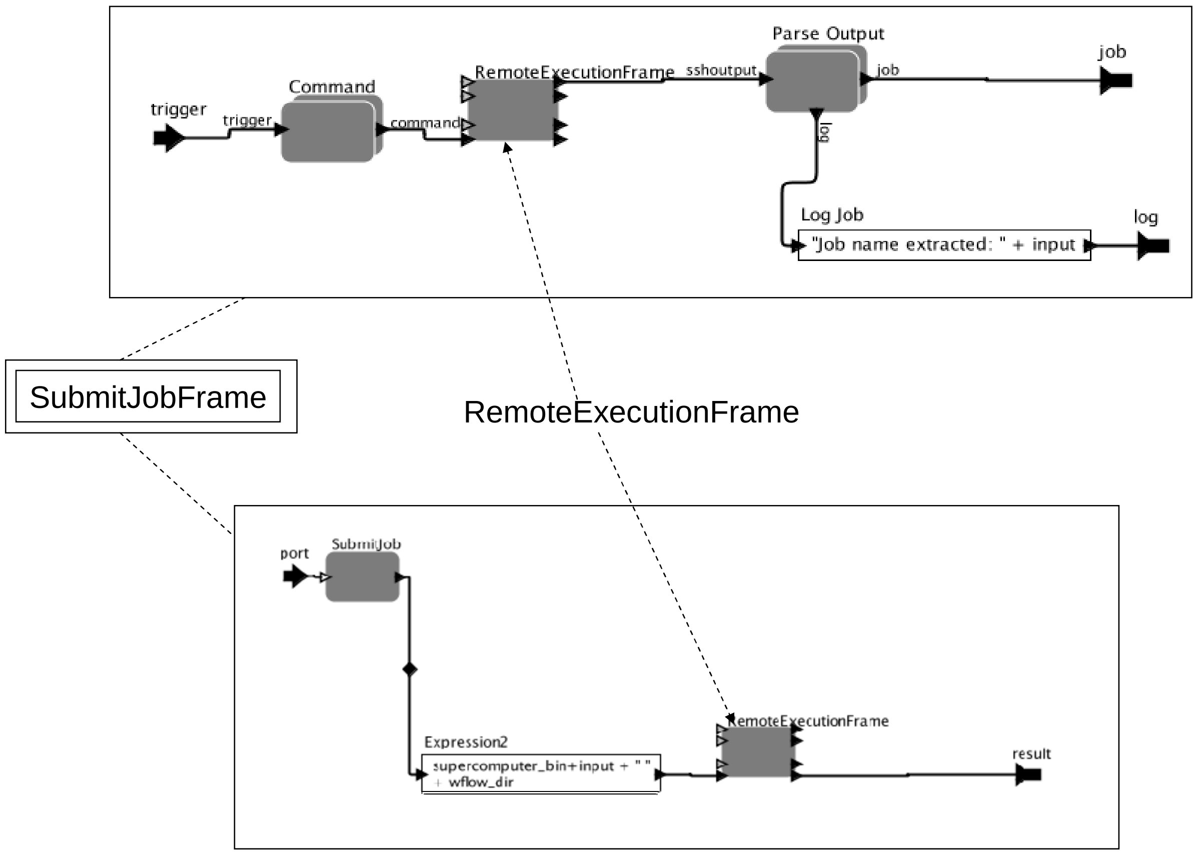


Figure 8. Nested RemoteExecutionFrame actor

We have provided an implementation of frames and dynamic embedding and used them in the modeling and design of scientific workflows from two different domains. In the case of TSI workflows, the use of frame and dynamic embedding resulted in the creation of a submit job frame and a remote execution frame that can be used in the two variant of TSI workflows without having to hard code a specific method of submitting a job in each of the workflow. In the case of NDDP workflow, only one version of NDDP workflow is needed for handling the parsing of three different kinds of data verses having to have to design three different workflows without using frames and dynamic embedding.

The current limitations of frames and dynamic embedding are 1) the inability to type check the automatic generated internal workflow before execution, 2) the additional overhead of creating a new internal workflow to execute the embedded component, 3) the need to recode and recompile the frame when there is a change in the selection process.

Our future work include coupling the dynamic embedding process with intelligent brokering. This will leverage the semantic annotation of ports and actors available in Kepler. In order to increase the usability of frames and dynamic embedding for workflow designer, there is a need to provide means for the scientists to specify configurable selection criteria.

References

[1] I. Altinas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, M. Schildhauer, and S. Mock. Kepler: An extensible system for design and execution of scientific workflows. In *Proc. of the Intl. Conf. on Scientific and Statistical Database Management (SSDBM)*, 2004.

[2] A. Ankolekar, M. Burstein, J. R. Hobbs, O. Lassila, D. McDermott, D. Martin, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara. DAML-S: Web Service Description for the Semantic Web. In *Proc. 1st Int'l Semantic Web Conf. (ISWC 02)*, 2002.

[3] B. Benatallah, M. Dumas, M. Sheng, and A. H. H. Ngu. Declarative composition and peer-to-peer provisioning of web services. In *Proceeding of 18th International Conference on Data Engineering*, 2002.

[4] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.

[5] S. Bowers, B. Ludaescher, A. Ngu, and T. Critchlow. ”Enabling Scientific Workflow Reuse through Structured Composition of Dataflow and Control-Flow. In *IEEE Workshop*

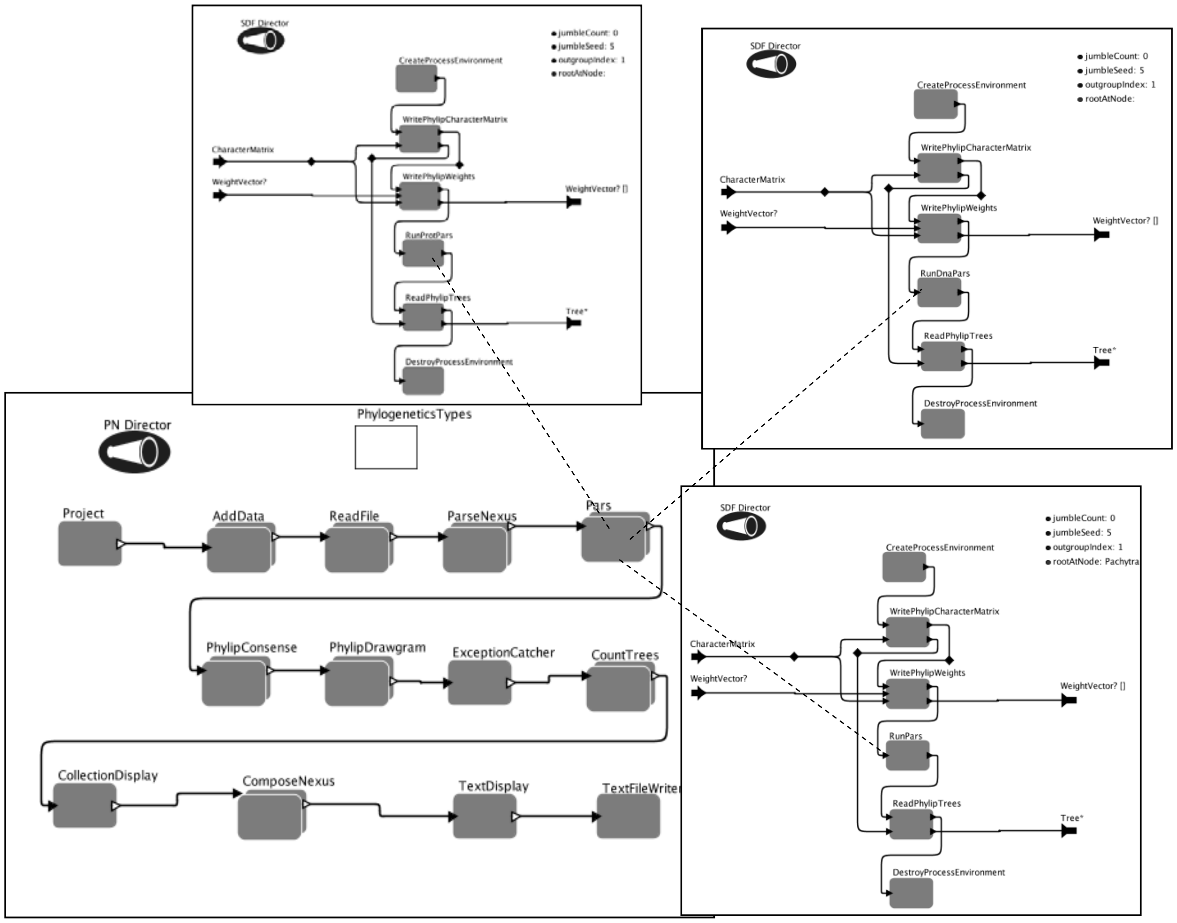


Figure 9. Three Different versions of NDDP workflows required for parsing three different types of phylogenic data.

on Workflow and Data Flow for Scientific Applications (SciFlow 2006), 2005.

[6] C. Brooks, E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng. Heterogeneous concurrent modeling and design in Java. Technical Report Technical Memorandum UCB/ERL M05/21, Univ. of California, Berkeley, 2005.

[7] C. Brooks, E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng. Heterogeneous concurrent modeling and design in Java, volume 3:ptolemy ii domains. Technical Report Technical Memorandum UCB/ERL M05/23, Univ. of California, Berkeley, 2005.

[8] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M.-C. Shan. Adaptive and Dynamic Service Composition in eFlow. In *CAiSE Conf.*, June 2000.

[9] A. Goderis, P. Li, and C. Goble. Workflow discovery: the problem, a case study from e-science and a graph-based solution. In *Proc. of the Intl. Conference on Web Services (ICWS2006)*, 2006.

[10] Y. Han, A. Sheth, and C. Bussler. A taxonomy of adaptive workflow management. In *Proc. of the Workshop Towards Adaptive Workflow Systems*, 1998.

[11] R. Hull and J. Su. Tools for composite web services: A short overview. *SIGMOD Record*, 34(2), 2005.

[12] N. Krishnakumar and A. Sheth. Managing heterogeneous multi-system tasks to support enterprise-wide operations. *Journal of Distributed and Parallel Databases*, 3(2), 1995.

[13] E. A. Lee and S. Neuendorffer. Actor-oriented models for codesign: Balancing re-use and performance. In *Formal Methods and Models for System Design*. Kluwer, 2004.

[14] B. Ludäscher and I. Altintas. On simplifying collection handling and control-flow issues in SPA/Ptolemy-II. Technical Report SciDAC-SPA-TN-2003-01, San Diego Supercomputer Center, 2003.

[15] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice & Experience*, 2006. to appear.

[16] Z. Luo, A. Sheth, J. Miller, and K. Kochut. Defeasible workflow, its computation and exception handling. In Y. Han, A. Sheth, and C. Bussler, editors, *CSCW-98 Workshop Towards Adaptive Workflow Systems*, 1998.

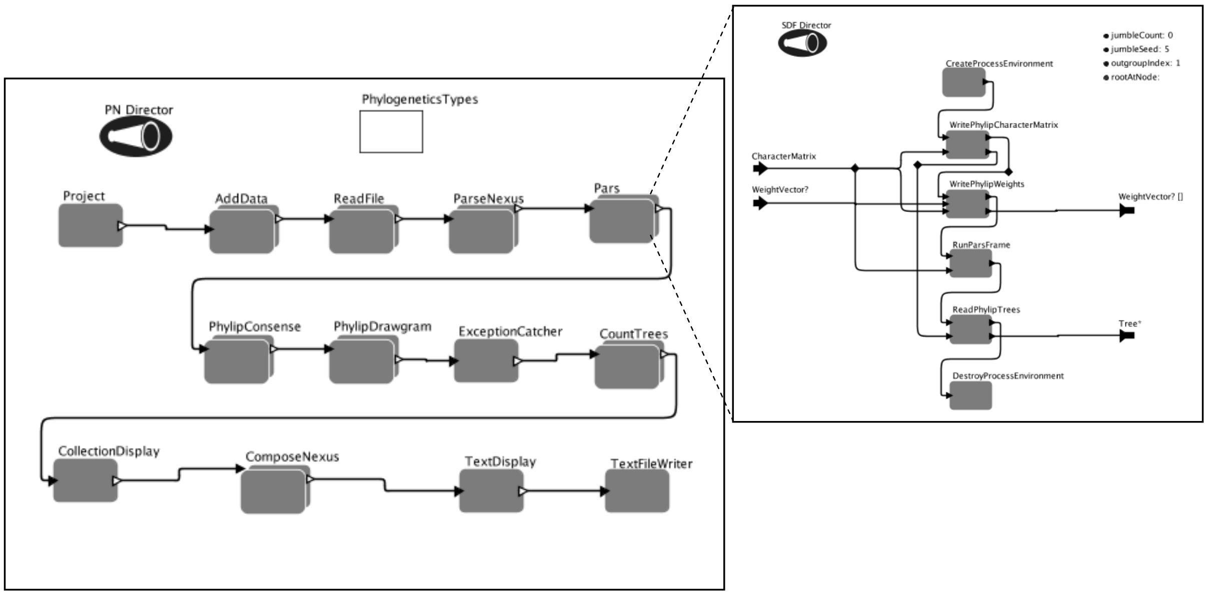


Figure 10. A single NDDP workflow with a frame actor (ParsFrame) that can parse three different types of phylogenic data.

[17] Z. LZ., B. B., L. H., N. AHH., F. D., and C. H. ”flexible composition of enterprise web services”. *International Journal of Electronic Commerce and Business Media*, 13(2), 2003.

[18] T. M. McPhilips and S. Bowers. An approach for pipelining nested collections in scientific workflows. *SIGMOD Record*, 34(2), 2005.

[19] T. McPhillips, S. Bowers, and B. Ludaescher. Collection-oriented scientific workflows for integrating and analyzing biological data. In *International Workshop in Data Integration in Life Sciences, DILS 2006, LNCS,4075*, pages 248–263, 2006.

[20] S. Narayanan and S. A. Mcllraith. Simulation, Verification and Austomated Composition f Web Services. In *Proceedings of the eleventh International Conference on World Wide Web, WWW2002*, Hawaii, USA, 2002.

[21] A. H. H. Ngu, D. Georgakopoulos, D. Baker, A. Cichocki, J. Desmarais, and P. Bates. Advanced process-based component integration in Telcordia’s cable OSS. In *Proc. of the Intl. Conf. on Data Engineering (ICDE)*, 2002.

[22] M. Paolucci and K. Sycara. Autonomous Semantic Web Services. *IEEE Internet Computing*, pages 34–41, September-October 2003.

[23] M. Rusinkiewicz and D. Georgakopoulos. From coordination of workflow and group activities to composition and management of virtual enterprises. In *Proc. of the Intl. Symposium on Database Applications in Non-Traditional Environments*, 1999.

[24] A. Sheth. From contemporary workflow process automation to adaptive and dynamic work activity coordination and collaboration. In *Proceedings of the Workshop on Workflows in Scientific and Engineering Applications*, 1997.

[25] X. Xin. Case study: Terascale supernova initiative workflow (TSI-Swesty). LLNL Technical Note, 2004. http://www-casc.llnl.gov/sdm/documentation/casestudy-tsi -s.doc.