Heterogeneous Workflows in Scientific Workflow Systems

Vasa Curcin, Moustafa Ghanem, Patrick Wendel, and Yike Guo

Department of Computing, Imperial College London

Abstract. Workflow systems are used to model a range of scientific and business applications, each requiring a different set of capabilities. We analyze how these heterogeneous approaches can be resolved, look at how existing workflow systems address this and present the solution in Discovery Net, which combines three levels of workflows, control, data and grid, at different levels of abstraction.

1 Introduction

A disctinction is often drawn between workflow systems based on their intended aim. Typically, the two types isolated are dubbed business and scientific workflows, with the former concentrating on increasing efficiency within an organization, while the latter are concerned with the fostering of innovation in scientific environments. While the two are not neccessarily in direct conflict and even share some common areas of interest such as automation, provenance tracking and collaboration, they do branch into different fields. The business workflow field revolves around issues such as messaging protocols, process optimization, and sophisticated patterns involved in component interaction. Scientific workflows, on the other hand include technologies which help bring the usability of the workflow abstraction in scientific research up to the level of a research scientist, by investigating interactive analysis creation, process knowledge capture, dissemination and reusability, and harnessing complex computational infrastructures within simple graphical metaphors. In addition to this separation, there are several perspectives from which a workflow model can be observed [2]. *Control flows* describe the flow of execution control between tasks, *data flows* represent a functional programming perspective, in which tasks are data transformations, while some other perspectives, such as *resource* and *operational* are also possible.

Workflow technologies have also been used in the context of Grid computing, mainly automating the submission and execution of user tasks on distributed computing resources. The level of details captured in such workflows varies between different systems. For example, within the Globus GT3 [3] project, GridAnt provides a workflow language with conditional, sequential, and parallel constructs that are used to describe the detailed steps required for submission and execution of user tasks on high performance resources. In contrast, within the Pegasus [4] system, the VDL language is used to define workflows as dependency graphs between user tasks, and the Pegasus system then automatically

Y. Shi et al. (Eds.): ICCS 2007, Part III, LNCS 4489, pp. 204–211, 2007.

maps the abstract workflows down to executable ones based on the available grid resources.

The Discovery Net system [1] was designed primarily to support the analysis of scientific data based on a distributed Web Service/Grid Service composition methodology. Within this framework, computational services executing at remote locations can be treated as black boxes with known input and output interfaces. Such services are connected together into acyclic graphs or workflows defined as sequences of operations where the outputs of one service act as inputs for other services. To initiate an execution, the user defines the termination node in the workflow and data is pulled through the preceding nodes in a demand driven fashion. Data passed between the nodes can be either atomic or a collection (e.g. a table of values), and in the latter case, each node will implicitly iterate over the input contents. That original paradigm is best suited to capture the semantics of a data flow model of computation. Although the system allowed the definition of parallel branches within a workflow to be formed (e.g. through the availability of multiple output ports in a service), it supported no explicit control flow constructs (e.g. conditionals) or explicit iteration operations. Despite these restrictions, the system has been extremely successful in enabling end users to create complex data analysis applications in various fields, including Life Sciences [6], Environmental Monitoring [5] and Geo-hazard Modelling [7]. Some of the key features that enabled its wide uptake were rooted in its support for a fully interactive model of workflow construction and deployment.

In this paper we describe the introduction of new extended features for supporting control flow semantics within the Discovery Net system. The paper is organized is follows. Section 2 reviews the main features of the scientific workflow systems that were described in the literature since the design of the original Discovery Net system. In Section 3, we present and describe the notion of workflow embedding as a generic solution to heterogeneous workflow composition, demonstrating it in Discovery Net through two new layers: a control flow layer and Grid control layer. Section 4 presents the conclusion and lays out the future research directions.

2 Heterogeneity in Existing Workflow Systems

There are numerous criteria which we can use to characterize the workflow execution semantics: atomic vs. streaming, push vs. pull, single vs. multiple starting/ending points, implicit or explicit iterations over data sets etc. The important thing to note is that *all* of these have their uses in some workflow scenarios. However, we argue that the correct way of modelling systems with multiple requirements is by implementing the heterogeneity in different semantic levels, rather than providing all of the functionality as different components in the same environment. In this section, we will investigate how some popular workflow systems address this issue.

2.1 Taverna

Taverna [8] is a graphical workflow authoring and execution environment, using SCUFL as its workflow language, and Freefluo as its enactment engine. SCUFL (Simple Conceptual Unified Flow Language) was developed for the purposes of the Taverna project, and can integrate any Java executable code as a component. The basic unit of execution in SCUFL is a processor, which has associated with it input and output ports. The processor may be regarded as a function from input data to output data, with possible side effects on the execution environment.

Data links between the components have a source processor and output port name, a sink processor and an input port name. This type of link ensures the basic consistency in pure dataflow execution. However, due to possible effects on the execution environment which are separate from inputs and outputs, explicit ordering constraints are introduced, which are not based on the data dependency between processors. These control constraints are useful when execution ordering must be imposed between two processors but when there is no dataflow between them. The execution of a SCUFL workflow will start from the nominated starting points, *sources*, and finish when all end points, *sinks*, have either produced their outputs or failed. Whether the workflow can execute partially (ie. if one sink failed, should the others complete) is configurable by the user.

In addition to control constraints and data links SCUFL also has an indirect conditional construct, which corresponds to *if/else* or *case* structure in procedural programming languages. This construct consists of a single input being passed to multiple nodes, which are all connected downstream to the same component. The user has to ensure that only one of them will succeed and continue the execution. If, however, for some reason multiple nodes execute and produce output, the downstream node will only process the first input received. The generic iteration construct is not supported, but one special case is the *implicit iteration*. This form of execution happens when a single-item processor receives a data collection, then the processor is executed once for each item in the collection. This behaviour is equivalent to the *map* construct in functional programming languages.

2.2 Triana

Triana [9] is a visual workflow-based problem solving environment, developed at Cardiff University. The functional component in Triana is called a *unit*. Units are connected through *cables* to form workflows. The notion of hierarchical workflows is supported through grouping connected components into a higher-level group unit. The created group unit implicitly has an associated control structure, another unit, which can coordinate its executional behaviour.

In addition to numerous functional components, Triana provides control flow units that operate on the same level as the data flow, and can be freely combined with them. The looping is achieved through a dedicated Loop component, that has two output ports. One typically connects to a functional unit, while the other provides the final output once the iteration is over. The Loop component

receives input data, evaluates its exit condition, and then passes the data either to the exit port or to its functional unit. Also, Triana posseses a number of *trigger* units, which can be used to send a signal on user action, at a fixed time or after a certain delay. Streaming execution is achieved using dedicated units for blocking, merging, splitting, pausing, etc. There is no explicit distinction between data and control components, placing them all on the same level. Semantics are left for the user to design, using the large number of specialized units provided. This design is opposed to languages such as YAWL and Kepler, which try to minimize the number of control nodes, the former through formalizing them into a minimum neccessary set, and the latter by separating control from the nodes altogether.

2.3 YAWL

YAWL (*Yet Another Workflow Language*) has its origins in theoretical work on workflow patterns [2] comparing a number of (mostly business) workflow systems. and was not developed for the purposes of an application project. Realizing that all the patterns described can be implemented in high-level Petri nets, albeit with some difficulty in cases of patterns with multiple process instances, advanced synchronization and cancellation, YAWL was based on high-level Petri nets, in order to preserve their benefits, but with extensions to help direct support of the special cases mentioned. YAWL's formal semantics are in contrast with other languages which either have no formal semantics (Triana, XPDL) or they constructed post hoc (Taverna). While the control flow perspective of YAWL is the most investigated one, the language also supports the data and resource perspective by allowing input and output parameters in the components to be connected to global variables.

The core component concept in YAWL is derived from the Petri Nets, with the workflow being a set of tasks, conditions and flows between them. Unlike Petri Nets, tasks can be connected to each other directly, without a condition inbetween (or an implicit condition that automatically succeeds). A task in a workflow can be either atomic or composite, with composite tasks containing another workflow (*extended workflow net* in YAWL terminology) within them. This corresponds exactly to the grouping concept present in Discovery Net, Taverna, Kepler, Triana and some other systems. There are six explicit branching constructs: three splits (AND, XOR and OR) and three joins (AND, XOR and OR), which model every legal data routing through the workflow. Due to the nature of splits the execution path through the workflow is determined dynamically at runtime, as opposed to being apriori statically determined. Both looping and conditional constructs are achieved using explicit conditions which evaluate the state of the workflow and direct the execution accordingly. So, there are no *if/else* or *while* components but the structure of the graph can create such behaviour.

The data flow in YAWL is achieved via variables, and can be split into internal and external data transfers. Internal transfers are always performed between the tasks and their workflows, since all variables inside tasks are internal to that task. So, in order to communicate some data between tasks A and B, task A

has to register its variable as the output parameter, and pass it to some global workflow variable N, which task B will take as its input parameter. External transfers occur between global variable and the user or an external component, such as a web or Grid service.

2.4 Kepler

Kepler [11] is a scientific workflow construction, composition, and orchestration engine, focusing on data analysis and modelling. This focus influenced the design in that it is suitable for modelling a wide variety of scientific domains, from physics via ecosystems to bioinformatics web services. Instead of trying to provide a generic semantic for all possible types of processes encountered in these domains, Kepler externalizes the execution engine from the workflow model, and assigns one *director* to each model, that then coordinates the model execution.

The workflow components in Kepler are represented by *actors* with ports that can be input, output, or mixed. Tokens are basic data containers, and they are passed from the output port of one actor to another through the relation connections. The number of tokens consumed and produced depends on the node used. *Directors* are the key concept in Kepler. While actors and relations together constitute a workflow model, the directors form the execution model, or the model of computation. In this setup, actors' intelligence stretches as far as knowing its inputs, the operation to be performed on them and what outputs to produce. The decision when to schedule the execution of each actor is left to the director. Therefore, depending on the director used, the actors may have separate threads of control, or they may have their executions triggered by the availability of new input, in a more conventional dataflow manner. The architecture in which components are agnostic to the manner in which they are executed is formalized as *behavioural polymorphism*.

Kepler supports four director types. **SDF - Synchronous Dataflow** is characterized by fixed token production and consumption rates per firing. The actor is invoked as soon as all inputs have data, which is possible to know since all actors have to declare their token production before the execution. Therefore, the order of execution is statically determined from the model, and components cannot change the routing of tokens during execution. **PN - Process Network** is a derestricted variant of SDF, in that the actor is invoked when the data arrives. However, there is no requirement that *all* data has to be present, which results in a more dynamic environment, where actors are executing in parallel and sending each other data when and if needed. The tokens are created on output ports whenever input tokens for an actor are available and the outputs can be calculated. The output tokens are then passed to connected actors where they are held in a buffer until that next actor can fire. The workflow is thus driven by data availability. **CT - Continuous Time** introduces the notion of time that is affixed to tokens in order to perform system simulations. The system is typically based on differential equations, and start conditions, which are then used to predict the state at some specified future time. The data tokens that are passing through the system then have a timestamp that the director is using

to determine the step and the stop condition. **DE - Discrete Event** director is working with timestamps, however, they are not used to approximate functions and schedule executions, but to measure average wait times and occurrence rates.

3 Discovery Net Workflow Embedding

The key new feature of Discovery Net 3.0 is the layered approach for definition, embedding and execution of scientific workflows. The top, control flow, layer is introducing new control operators for the coordination of the traditional data flow operations. This layer enables the execution of distributed applications with scheduling dependencies more advanced than the simple data availability criteria, and where the data passing between components need to be restricted due to volume or costs associated with transfers. The middle layer corresponds to the traditional Discovery Net data flow layer enabling data integration, transformation and processing using distributed services. The bottom layer, Grid Control, enables the access and control of Grid resources.

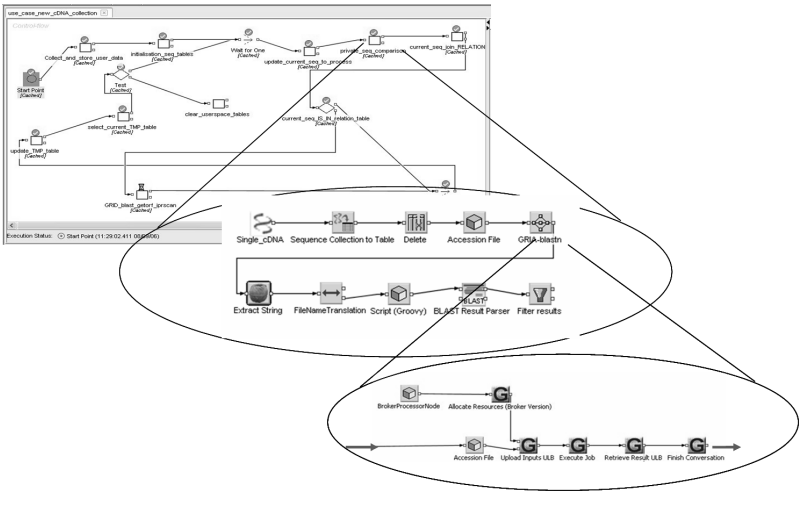

Fig. 1. Hierarchical workflow composition

3.1 Control Flow Layer

This layer includes a range of control flow elements such as branching synchronisation, conditional branching and looping. The control flow mechanisms use a different execution scheme to the standard workflow execution engine; hence the control flow components cannot be connected to data flow components, but rather they orchestrate data flows, specifying which will be the next data flow

to run based on the control decision. The key new control flow operations introduced in Discovery Net include:

- **Test** Generic condition construct, where the user specifies the condition, and indicates which branch is to be executed, depending on the outcome.
- **Wait for All** Provides synchronisation allowing waiting for all arriving tasks to be completed before a workflow can proceed.
- **Wait for One** Similar to Wait for All, except that it executes the node, once the first of the arriving tasks is complete.
- **For** Explicit looping construct, which loops through activities a specified number of times - a syntactic shortcut for a generic loop implementable using the former three components.
- **While** Loops through activities while a variable is true, also a syntactic shortcut.

Firstly, a notion of *token* was introduced, in which each component may receive a token, which will cause an execution of its instance. Multiple tokens may exist in the workflow at any one time, and even in a single component. Secondly, the execution switched from the pull-based model to a push-based one, with a dynamic flow of control – nodes making runtime decisions about the direction which the execution will take. Furthermore, tokens have access to the component that produced them, thereby giving a link to the result of a previous execution, if needed.

Interestingly enough, even though the control flow does not fit into the workflow-as-service model, so strongly upheld on the dataflow level, it utilizes it in order to have a uniform execution paradigm for its components. Namely, the execution node in the control flow is the only component which can perform an action outside of the scope of the flow being executed. The way it operates is by executing an internal dataflow, with a nominated output, which is used to produce and fire a token once the execution is complete. The output is nominated by declaring it as the output of the service created from the inner workflow. So, the atomic unit of execution inside the control flow is actually a dataflow-based service.

3.2 Grid Control Layer

The role of the bottom layer is to enable access and control of remote Grid computing resources. Specifically, within this layer, sub-workflows are used to control selection of resources as well as authentication, job submission, job invocation, collection of results as well as session management. An example implementation of these operations for the GRIA middleware is described in details in [10].

4 Summary

This paper discussed the need for heterogeneity in workflow systems, reviewed the approaches different scientific workflow systems take, and introduced the hierarchical approach to combining control and data structures in Discovery Net.

Discovery Net adopts a conservative approach to its dataflow semantic, insisting on a single-output static execution model, corresponding to a function call, with no non-determinism in the execution flow. All non-deterministic elements, such as conditional and looping structures, are implemented in a separate level, as well as strategies for GRID execution which involve scheduling and service discovery. This is in sharp contrast to other systems analyzed, which are either enriching a dynamic control flow structure with global variables (YAWL), introducing dataflow-flavoured control elements (Taverna), placing all the combinations of functionality/semantics into separate nodes (Triana). The Kepler approach of letting the user determine the hierarchical composition of different semantics, is the closest to the one presented here, but it allows for some distinctly non-workflow systems to be built. In the future work, we plan to analyze how multiple execution semantics interact in a hierarchical model, by looking at error handling, user interaction and process provenance. The complete picture of the Discovery Net model will then allow us to compare and contrast workflow executions with other scientific workflow systems, producing formal notions of equivalences between these systems.

Acknowledgement. The authors would like to thank Dr. Yong Zhang and Mr. Nabeel Azam for the helpful discussions on the embedding implementation.

References

1. AlSairafi et al: The Design of Discovery Net: Towards Open Grid Services for Knowledge Discovery, High Performance Computing Applications, vol 17, no 3, (2003) 297–315
2. Aalst van der et al: Advanced Workflow Patterns, 7th International Conference on Cooperative Information Systems, vol 1901, Lecture Notes in Computer Science (2000) 18–29
3. Ian T. Foster: Globus Toolkit Version 4: Software for Service-Oriented Systems, NPC (2005) 2–13
4. Deelman, E. et al: Pegasus: Mapping Scientific Workflows onto the Grid, Lecture Notes in Computer Science : Grid Computing, (2004) 11–20
5. Richards et al: Grid-based analysis of air pollution data, Ecological Modelling, vol 194, no 1–3 (2006) 274–286
6. Rowe et al: The discovery net system for high throughput bioinformatics, Bioinformatics, vol 19, no 90001, (2003) 225–231
7. Guo et al: Bridging the Macro and Micro: A Computing Intensive Earthquake Study Using Discovery Net, Proceedings of SC2005 ACM/IEEE (2005)
8. Hull et al: Taverna: A tool for building and running workflows of services, Nucleic Acids Research, Web Server Issue, vol 34, (2006) W729–W732
9. Taylor, Ian et al: Visual Grid Workflow in Triana Journal of Grid Computing, vol 3, no 3-4, (2005) 153–169,
10. Ghanem, M. et al.: Grid-enabled workflows for industrial product design, 2nd IEEE International Conference on e-Science and Grid Computing (2006)
11. Ludaescher, B. et al.:Scientific Workflow Management and the Kepler System, Concurr. Comput. : Pract. Exper., vol 18, no 10, (2006) 1039–1065