CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2006; **18**:1115–1140
Published online 13 December 2005 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.988

Grid-Flow: a Grid-enabled scientific workflow system with a Petri-net-based interface

Zhijie Guan*,†, Francisco Hernandez, Purushotham Bangalore, Jeff Gray, Anthony Skjellum, Vijay Velusamy and Yin Liu

Department of Computer and Information Sciences, University of Alabama at Birmingham, 115A Campbell Hall, 1300 University Boulevard, Birmingham, AL 35294-1170, U.S.A.

SUMMARY

Advances in computer technologies have enabled scientists to explore research issues in their respective domains at scales greater and finer than ever before. The availability of efficient data collection and analysis tools presents researchers with vast opportunities to process heterogeneous data within a distributed environment. To support the opportunities enabled by massive computation, a suitable scientific workflow system is needed to help the users to manage data and programs, and to design reusable procedures of scientific experimental tasks. In this paper, the design and prototype implementation of a scientific workflow infrastructure, called Grid-Flow, is presented. Grid-Flow assists researchers in specifying scientific experiments using a Petri-net-based interface. The Grid-Flow infrastructure is designed as a Service Oriented Architecture with multi-layer component models. The contributions of Grid-Flow are as follows: (1) a new, lightweight, programmable Grid workflow language, Grid-Flow Description Language, is provided to describe the workflow process in a Grid environment; (2) a Petri-net-based user interface, based on the Generic Modeling Environment, is demonstrated to help the user design the workflow process with a Petri-net model; and (3) a program integration component of the Grid-Flow system is presented to integrate all possible programs into the system. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: workflow management system; Grid computing; Grid workflow system; Petri-net model

1. INTRODUCTION

Advances of computing technologies have enabled scientists to explore research issues in their areas at time and space scales both greater and finer than ever before. Faced with a growing set of emerging powerful and effective data analysis tools enabled by new technologies, researchers in many scientific fields (for example, systematic biology, cheminformatics, climatology, and astronomy) require a

*Correspondence to: Zhijie Guan, Department of Computer and Information Sciences, University of Alabama at Birmingham, 115A Campbell Hall, 1300 University Boulevard, Birmingham, AL 35294-1170, U.S.A.
†E-mail: zguan@uab.edu

Received 2 June 2004
Revised 15 March 2005
Accepted 27 April 2005

problem solving environment that can utilize the distributed data and computing resources, orchestrate the data analysis tools crossing various platforms, and support collaborative efforts between and among each participant of data/computation intensive applications. To satisfy such demands, two emerging computer technologies, workflow management [1] and Grid computing [2], cooperate to provide a promising solution of a distributed, collaborative workflow system, which is denoted here by 'Grid workflow system'.

According to the definition from the Workflow Management Coalition (WfMC) [1], a workflow is '*The automation of a business process, in whole or parts, where documents, information or tasks are passed from one participant to another to be processed, according to a set of procedural rules*'. The definition of workflow in a scientific area may differ slightly from the above because of a greater emphasis on scientific data and analysis processes, as compared to the business process influence of the quoted definition. However, the essential characteristics of a computer workflow process are common across both definitions; that is, a computer workflow is a procedure that applies a specific computation (process) on selected data according to certain rules. Because the data and computing may be dispersed in a physically distributed environment, the workflow management system needs one or more mechanisms to handle the data transfer issues and invoke the computational tools over a distributed, heterogeneous platform. Grid computing technology precisely satisfies those requirements by providing a new computing infrastructure for large-scale resource sharing and distributed system integration. A Grid, according to the definition in [3], is a system that '*coordinates resources that are not subject to centralized control using standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of services*'. Based on the services provided by Grid computing, a Grid workflow system can support workflow definition (including process, resource, program defining), workflow execution, and workflow monitoring or administration over a set of dispersed data and computing resources.

Although a large set of candidate features may serve the purpose of differentiating between any two Grid workflow systems, this paper focuses on three major issues of a Grid workflow system to identify the system and to determine its performance. These three issues are as follows: the model used to describe a workflow process; the language conveying the process definition; and the mechanism to integrate data and computing resources in a distributed environment. Most of the current Grid workflow systems, including UNICORE [4], Symphony [5], Kepler [6], GridAnt [7], Pegasus [8], GridFlow [9], and Triana [10], use directed acyclic graphs (DAGs) as a modeling language to describe the workflow process in a graphical model. Although a DAG is intuitive for constructing process descriptions and relatively easy to comprehend, it has several drawbacks. First, it has limitations with respect to modeling power. As there is no cyclical circuit in a DAG, it is hard to express the loop patterns in a DAG model explicitly. Second, a DAG model cannot express process state information. Owing to these limitations, Grid workflow users will have difficulty monitoring the execution of a process with a DAG model. To avoid such drawbacks, the Petri-net model [11] has been applied in several Grid workflow systems, such as the workflow systems presented in [12] and [13]. Compared with a DAG, a Petri-net model has more powerful modeling capabilities. A Petri-net model employs 'places' to represent the status of a workflow process, which facilitate users' monitoring of the process execution. Furthermore, some extended Petri-net models have equivalent computational power to a Turing Machine [11].

Although a graphical model greatly eases the description of a complex workflow process for users, it can hardly be accepted as a complete process definition by the Grid workflow engine, which is the executor of the workflow process. The reasons for this incapability can be summarized by the following two points. First, compared with its ability on describing control flows, a graphical model lacks the capability to elaborate the meta information about data and computing resources. Such 'meta information' are crucial for the workflow engine to construct the references to the resources. Second, typical workflow engines are usually designed to process text-based languages instead of graphical models, since apparently the designers of workflow engines have limited exposure to the knowledge of the emerging Model-Integrated Computing (MIC) technology [14]. Thus, most workflow engines, including Triana [10], Taverna [15], and Kepler [6], have architectures similar to a traditional compiler, which is not suitable to directly process graphical models.

A workflow language is consequently designed to bridge the gap between the graphical model and the Grid workflow engine. The major functions of workflow language are as follows: (1) recording the meta information about the data and computing resources; and (2) conveying the workflow process definition. An interpreter is usually provided in the Grid workflow system to transform the graphical model into a specific workflow language. Compared with a graphical model, a workflow language is more efficient for workflow domain-specific experts.

Two workflow languages, GSFL [16] and BPEL4WS [17], are popular choices in current Grid workflow systems. Both of these languages are capable of describing complex data entries and workflow processes. Each language also adopts XML [18] as a persistent storage notation. The sophisticated description capabilities, however, also afford disadvantages to both these languages; that is, they are too complicated to be mastered by casual users of the workflow system. Furthermore, it is hard, if not impossible, for an experienced user to describe even a non-trivial workflow process and its data and computing resources directly with any of these two languages without the help of assistant tools. Although these two languages are widely accepted as a reference model and interface for workflow languages [6,10,15], they are seldom fully implemented for public access.

Compared with these two languages, a lightweight Grid workflow language that addresses major workflow functions would be more preferable for expert users. This language must be powerful enough to support general workflow patterns, as well as sufficiently agile in order to let the expert users directly write workflow definitions. This language should also be compatible or translatable to currently popular workflow languages. Furthermore, this language should be able to exploit full support from both the graphical user interface and the workflow engine.

After application design by an end user, a Grid workflow process is usually submitted to a Grid workflow engine, where the process will be executed. The workflow engine is responsible for parsing the process description and performing the corresponding computation on the data according to the schedule embedded in the description. Certain of the current Grid workflow systems, such as McRunjob [19] and ScyFlow [20], base their computations on Grid services provided by the Grid infrastructure, which means that only predefined grid services can be used in the workflow process. This design may limit users who may want to integrate their own tools into the workflow system. Yet, current workflow systems seldom provide users with a useful mechanism to integrate their data analysis tools. Thus, a Grid workflow system that can integrate all possible tools in an improvizational manner is preferred, no matter where the tools are physically located (e.g., local computer, network environment, or even the Internet).

Considering these critical issues, this paper presents the design and prototype implementation of Grid-Flow, a scientific Grid workflow infrastructure, which is based on the Service Oriented Architecture (SOA) [2]. The contributions of this paper are as follows.

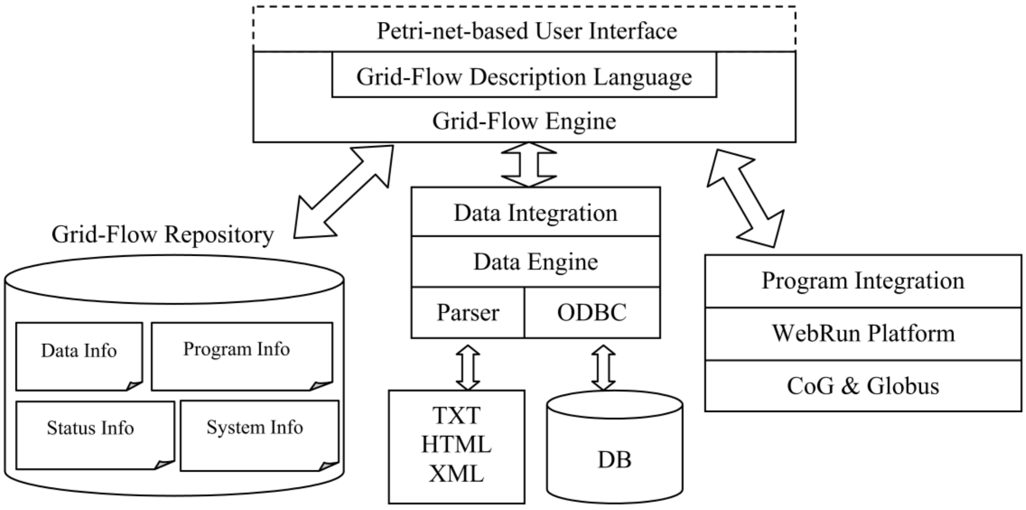


Figure 1. The Grid-Flow system architecture.

1. A lightweight, programmable Grid workflow language, denoted the Grid-Flow Description Language (GFDL), is provided to describe the workflow process in a Grid environment. This language is powerful enough to describe all types of workflow patterns, and is intended to be easily mastered by scientific domain experts.
2. A Petri-net-based user interface, based on the Generic Modeling Environment (GME) [14], is demonstrated to help the user design a workflow process with a Petri-net model. A model interpreter (see Section 2.4) is implemented to translate the Petri-net workflow model into GFDL.
3. A program integration component of the Grid-Flow system is presented to integrate most types of tools into the system across distributed and heterogeneous platforms. The integration component can access Grid services through a unified platform (such as WebRun [21]), as well as introduce the programs in the local operating system into the workflow process as atomic activities.

Figure 1 illustrates the architecture of the Grid-Flow system. The Petri-net-based user interface provides users with an integrated design environment. GFDL conveys the workflow process definition to the Grid-Flow engine. The orchestration of distributed programs is supported by the data and program integration components. Further details are provided in Section 3.

The remainder of this paper is organized as follows. Section 2 offers a brief survey of current Grid workflow systems, their modeling approaches, and their workflow languages. The design and implementation issues of the Grid-Flow system are discussed in Section 3. Section 4 presents an illustrative example of a Grid-Flow system, including the design and execution details of the bioinformatics workflow process. Section 5 offers conclusions and describes potential future research and development of the Grid-Flow system.

2. RELATED WORK

In this section, background on Grid workflow systems, modeling approaches, Grid workflow languages, and the generic modeling environment is provided. The origin of Grid-Flow is also described.

2.1. Grid workflow system

Although Grid workflow systems lack a long history when compared to other technologies in the Grid computing community [13], there is, in the authors' experience, quite a bit of related work on orchestrating tools in a Grid environment that has been proposed and used by researchers. The basic idea of orchestrating programs in a distributed environment can be found in two early projects, WebFlow [22] and the Common Component Architecture (CCA) [23]. The purpose of the WebFlow system is to provide a Web-based visual programming environment for distributed computing software. It was developed as a coordination model and programming paradigm for Java-based Web applications. Compared with WebFlow's emphasis on integrating applications, CCA focuses more on composing disparate and coarse-grain, high-performance components into a running application [23]. This strategy of component composition has been inherited by many current Grid workflow systems, such as GridAnt [7], Triana [10], Symphony [5], XCAT [24], GridFlow [9], and Kepler [6]. The strategy has been extended to applications of composition of the software modules and Web/Grid services [5–7,9,10,24].

Almost all of the Grid workflow systems mentioned above have a common characteristic; that is, they are derived from some existing flow control tools or systems and are applied to orchestrate Grid-enabled programs or services. For example, GridAnt [7] was developed from Ant [25], while XCAT was based on CCA [7]. Another common feature for these Grid workflow systems is that they all provide a graphical user interface and a script-like language for users to organize and describe the workflow process. Some other Grid workflow systems focus on integration of programs/applications instead of components/services. These workflow systems include P-GRADE [26], ScyFlow [20], McRunjob [19], Taverna [15], GriPhyN [27], and Kepler [6]. From these descriptions [6,15,19,20,26,27], it is evident that these Grid workflow systems intend to provide interoperability among various programs/applications across heterogeneous computational platforms. Since programs/applications have more diverse interfaces than components/services, these program-based workflow systems are more domain-specific than component-based workflow systems in order to limit the diversity of integrating objects in a manageable scope. Some other resource management system and process planning/scheduling system could also provide workflow control functionality in a Grid environment. UNICORE [4] and the Directed Acyclic Graph Manager (DAGMan) [28] are two good examples of such systems.

2.2. Modeling approach

The majority of Grid workflow systems adopt the DAG and its variants as their approaches for workflow process modeling [4–6,10]. For example, UNICORE [4] and Symphony [5] use DAGs to describe the workflow process, while Kepler [6] and Triana [10] use Directed Cyclic Graphs (DCG) as their modeling methods. DAGs, as well as DCGs, have limitations on their modeling abilities.

Since a DAG has no cycles in its model, it is not applicable to explicitly express loops. Because of this drawback, the Triana system [10] does not explicitly support control constructs. All of the loop controls are handled by a specific loop component in Triana. A more powerful modeling approach, Petri nets [11], is gradually being introduced into Grid workflow systems to model workflow process [12,13]. A Petri-net modeling method is applied in [12] to describe user tools and workflow schemes developed in the Fraunhofer Resource Grid (FhRG) [29]. An extension of the Petri net [11], D-Petri net, is also employed in [13] to capture the characteristics of dynamism of Grid job scheduling.

2.3. Grid workflow language

In addition to the above modeling approaches, the Grid workflow description can usually be achieved by taking advantage of scripting Grid workflow languages. Web service software industrial providers were first to present several flow control languages for Web services; these languages include the Web Services Flow Language (WSFL) [30], XLANG [31], the Web Services Conversation Language (WSCL) [32], and the Web Services Choreography Interface (WSCI) [33]. With the development of Web services, WSFL and XLANG converged and this has led to a new generation of specification language for business interaction, the Business Process Execution Language for Web Services (BPEL4WS) [17]. BPEL4WS [17] expands the Web Service interaction model and make it applicable to depict business transactions. With the emergence of Grid services, the Grid Services Flow Language (GSFL) [16] was proposed to address the issue of integrating Grid services across distributed and heterogeneous platforms within the Open Grid Services Architecture (OGSA) [34]. Both BPEL4WS and GSFL adopt eXtensible Markup Language (XML) [18] as their basic language format [16,17]. Most of the Grid workflow systems also choose XML-based language as their workflow process language. For instance, Triana [10] uses an XML-based language similar to the Web Services Description Language (WSDL) [35]. At the same time, Triana can use any languages that are compatible with BPEL4WS [10]. Since no standards have been accommodated for Grid services, the research area of Grid workflow language is still undergoing a rapid change and development. For example, a new Grid Workflow Execution Language (GWEL) has been proposed to reuse ideas from BPEL4WS on describing interactions between services defined with WSDL in [36].

The Grid workflow language (GFDL) proposed in this paper was designed with the concern of being compatible and translatable to most of the scripting Grid workflow languages. A translator could be easily constructed by interpreting GFDL's grammar to the counterpart of another workflow language with the help of the GFDL parser and appropriate generator of the corresponding language. Since more and more workflow languages are emerging to adapt Grid services, the implementation of translators between GFDL and those languages are intentionally deferred until a standard of Grid workflow language matures. Once the standard is consolidated, a translator between GFDL and the standard language will be implemented to facilitate the collaboration among various workflow systems.

2.4. Generic modeling environment

From a modeling perspective, the expressive power in software specification is often gained from using notation and abstractions that are aligned to a specific problem domain. This can be further enhanced when graphical representations are provided to model the domain abstractions. In domain-specific modeling, a design engineer describes a system by constructing a visual model using the terminology and concepts from a specific domain. Analysis can then be performed on the model, or the model can be synthesized into an implementation [37]. Model-Integrated Computing (MIC) has been refined over the past decade at Vanderbilt University to assist in the creation and synthesis of complex computer-based systems [14]. A key application area for MIC is in those systems that have a tight integration between the computational structure of a system and its physical configuration (e.g. embedded systems) [38]. In such systems, MIC has been shown to be a powerful tool for providing adaptability in evolving environments [39]. The GME [14] is a domain-specific modeling tool that realizes the principles of MIC. The GME provides meta-modeling capabilities that can be configured and adapted from meta-level specifications (representing the modeling paradigm) that describe the domain.

Meta-models in the GME can be instantiated to provide a domain-specific modeling language (DSML) that is customized to the visual representation and semantics appropriate for that domain. A DSML may have multiple interpreters associated with it that permit synthesis of different types of artifacts. For example, one interpretation may synthesize to C++, whereas a different interpretation may synthesize to a simulation engine or analysis tool [37]. A DSML raises the level of abstraction to highlight the key concerns of the domain in a manner that is intuitive to a subject matter expert or systems engineer, who may not be familiar with lower-level technologies in the solution space (such as conventional general-purpose programming languages, operating systems, and middleware platforms).

As an example of domain-specific modeling, Figure 2 illustrates a simplified Petri-net domain as implemented in GME. The top part of the figure defines a basic meta-model to represent Petri nets. This meta-model is described in UML [40] and OCL [41] (not shown) and defines places and transitions of a Petri net, as well as various semantic and visualization attributes. From this meta-model, a new Petri-net modeling environment is generated (bootstrapped from within GME). The middle of the figure defines an instance of the meta-model that represents a solution to the dining philosopher's problem as specified in the Petri-net modeling language.

GME permits model interpreters to be associated with specific domains as tool plug-ins. A model interpreter traverses the internal structure of a model and generates various artifacts during the interpretation. GME provides an API for accessing the internal model structure to permit interpreters to be written in C++ or Java. The bottom of Figure 2 symbolically represents a generated artifact from the interpretation of the dining philosopher's model; this artifact could be source code, an XML representation of the model, or some other translation. A comparison can be made between the meta-model and its instance to that of a programming language definition and a particular program written in that language. The meta-model defines the schema definition for expressing the correct syntax and semantics of instantiations; similarly, a grammar of a programming language defines the correct semantics of a program in that language. Furthermore, a model interpreter is akin to a compiler that generates machine code from a general programming language. Although the example in Figure 2 was chosen for simplicity, GME has been used to create very rich modeling environments containing thousands of modeling components [42].

2.5. BioFlow synopsis

The BioFlow system [43] is a previous work of this paper's first author in cooperation with others. Its initiation dates back to the early stage of the development of Internet/Grid computing. With the support of Internet computing technology, BioFlow provides a workflow platform for the development of high-level bioinformatics applications using both local and online resources.

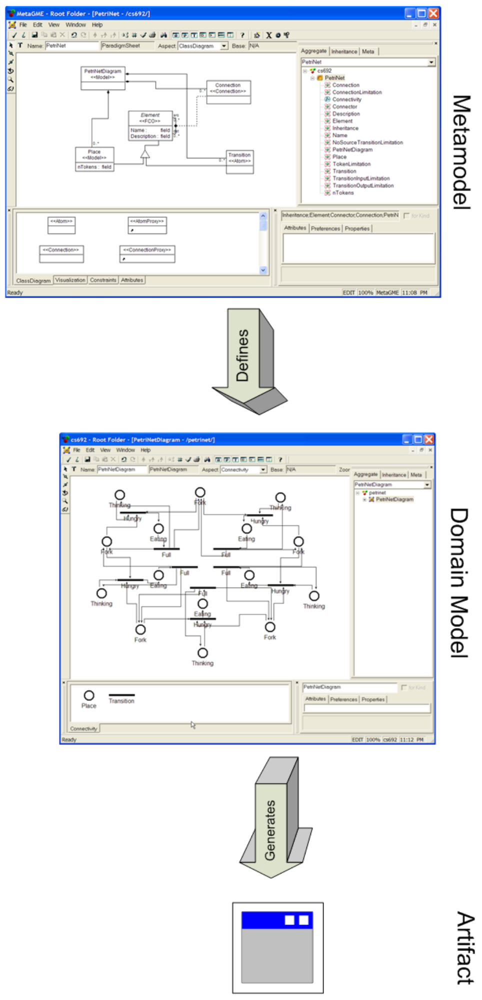


Figure 2. A Petri-net domain in GME.

Using HTQL [44], BioFlow can invoke Internet programs, query information from Web pages, as well as shield the distributive and heterogeneous details of the online resources from the users. In addition, a BioFlow [43] language records the 'meta information' of online resources and describes the flow control of *ad hoc* high-level applications. From the viewpoint of Grid computing, however, BioFlow has the following limitations:

1. the prototype of the BioFlow GUI can only support simple workflow processes instead of complicated workflow patterns that arise from real-world applications;

2. BioFlow directly accesses Web pages for submitting jobs to the Internet programs, rather than employing Web or Grid services;
3. BioFlow was not capable of tailoring itself in response to the rapid change of the Internet programs, thus the quality of the target Internet programs cannot be guaranteed.

On the other hand, the architecture and the BioFlow language [43] are flexible enough to accommodate the development of Grid computing technology. Hence, the Grid-Flow system adapts and extends BioFlow's architecture; the syntax and grammar of the BioFlow language are reused to develop GFDL in the Grid-Flow system. Based on the BioFlow system, the Grid-Flow system makes the following improvements:

1. the Grid-Flow system will provide an interface designed to be user-friendly and based on the Petri-net modeling approach;
2. the Grid-Flow system adapts the program integration component to support Web and Grid services. The program integration component is based on WebRun [21], which is a unified platform supporting Grid computing [2] environments.

3. GRID-FLOW SYSTEM

This section introduces the overall Grid-Flow system, including descriptions of the Grid-Flow system architecture, the Grid-Flow language, the graphical user interface, the Grid-Flow engine, and the grid-enabled data and program integration sub-system.

3.1. System architecture

The Grid-Flow architecture (as shown in Figure 1) is adapted and extended based on the BioFlow system architecture presented in [43]. This architecture can be categorized into three layers: the Petri-net-based user interface, the Grid-Flow engine, and the Grid-enabled data and program integration framework. The Petri-net-based user interface, implemented within a graphical modeling environment, can help users design the workflow via a graphical editor, translate the workflow specification into the GFDL, and monitor the execution of the workflow process. The GFDL, which conveys the specifications of workflow processes, acts as a bridge connecting the Petri-net models with the Grid-Flow engine. The Grid-Flow engine layer is responsible for interpreting user-defined workflow processes and responding to users' monitoring. More importantly, it coordinates the activities and execution of user workflow using appropriate services provided by the layer of Grid-enabled data and program integration. The data and program integration layer of the system infrastructure plays a critical role of interconnecting distributed computing resources in the whole system. To simplify its descriptions, the data and program integration layer can be divided into two components. The data integration component can access various data sources with heterogeneous data formats and perform the transformation. The program integration component, the other part in this layer, is based on the WebRun system [21], a unified platform that invokes remote programs via Grid computing technology.

```
Register Data data_name As Text | Table;
Set data_source= 'data_source_string';
Set data_format= 'data_format';
. . . . . .
End;
```

Figure 3. Syntax of data registration.

3.2. Grid-Flow Description Language

The GFDL is a declarative workflow language that is used to describe the data, programs, and processes in the Grid-Flow system. GFDL supports two principal types of sentences: resource registration and process description. To achieve the advantages of a declarative language that is both simple and easy to master, GFDL was defined by features borrowed from a well-known declarative language, the Structured Query Language (SQL) [45]. Resource registration can be thought of as similar to the SQL Data Definition Language (DDL), and the process descriptions closely resemble SQL's Data Manipulation Language (DML). For a successful compilation of a process description, all resources and processes referred to in the process description must also be defined and compiled successfully.

As a workflow definition language, GFDL was designed to provide three main functions: data registration, program registration, and process description.

3.2.1. Data registration

Most workflow processes would involve a set of operations on various data from distributed resources. The data upon which the Grid-Flow system is to operate must first be registered. Registration implies recording 'meta information' about the data into the system repository. Generally, the registration program needs to know two features of the data: the source and the format. The source of the data informs the system of where to get the data, and the format of the data tells the system how to use the data in the corresponding processes. The syntax in Figure 3 is used to register data resources.

From this SQL-like [45] syntax, it can be observed that data could be registered as a *Text* file (including plain text, HTML, and XML) or a *Table* (as in a relational database). The data source and format features are assigned following the data name declaration. More features about the data, if needed, can be added into this template.

The Grid-Flow system simplifies the data registration procedure by providing users with a 'wizard' to guide users' step-by-step input of required information. After retrieving all of the related information, the registration wizard automatically generates data registering sentences and submits them to the Grid-Flow engine. The Grid-Flow engine responds by processing data registering sentences and storing the meta information about the data into the Grid-Flow system. For example, when a user registers the data *TMPredInput*, the wizard generates the sentences in Figure 4 and sends them to the Grid-Flow engine.

```
Register Data TMPredInput As Text;
Set data_source='http://www.ch.embnet.org/software/...';
Set data_format='HTML'
Set data_ddf='File:///c:\BioFlow\DDF\TMPredInput.ddf';
End;
```

Figure 4. Registration sentences of data TMPredInput.

```
Register Program program_name As Internal_Program | OS_Program | Grid-Flow_Program;
Input input_parameters_list;
Output output_parameters_list;
// Program features description goes here
......
End;
```

Figure 5. Syntax of program registration.

3.2.2. *Program registration*

A program is also an important concept in GFDL. A program is defined to be an atomic execution unit that cannot be subdivided in the Grid-Flow system. A task is an instance of a program once scheduled or to be scheduled. A program could be invoked by the Program Integration component, access some data through the Data Integration component, and accomplish certain functions. For example, when the Program Integration component plans to invoke the program *mpiblastp* [46] on a remote resource to compare a protein sequence against a large protein database, it must retrieve the analyzing sequence through the Data Integration component, send it to a remote resource, call the program *mpiblastp* via the Globus Toolkit [47], and fetch the results from the remote side after the execution.

In the Grid-Flow system, three types of programs can be used. One type of program is implemented in the Grid-Flow system, such as basic mathematical operations (for example, less than, larger than, equal to) and fundamental logical operations (for example, and, or, not). This kind of program is called an *Internal Program*. Another type of program is provided directly by the operating system, either from local computers or from the distributed environment, such as PAUP [48] and ClustalW [49]. These are called *OS Programs*. The third type of program is designed by users who utilize the Grid-Flow language to plan their processes. After the design process of data analysis with the Grid-Flow language, users can save their blueprints into Grid-Flow description files for future use. A saved Grid-Flow process can be used to construct complex processes as a single unit, and be reused as a simple process definition. Such kinds of reusable programs are named *Grid-Flow Programs*. Actually, the Grid-Flow Programs are a kind of reusable sub-process. The syntax for registering programs is listed in Figure 5.

In this syntax, the program name and type must be declared followed by a list of input and output parameters. The input parameter list defines what data should be fed into the program when the program starts. The output parameter list defines what data should be transferred out to the Grid-Flow system

```
Register Program TMPred As OS_Program;
Input TMPredInput;
Output TMPredOutput;
Set program_source='http://ardra.hpcl.cis.uab.edu:8080/webservices/...';
End;
```

Figure 6. Registration sentences for the Program *TMPred*.

after the program ends. The remaining syntax follows with different contents according to different types. For an Internal Program, it is not necessary to provide any additional information. For an OS Program, additional information is provided about the program, such as the program location, operating system supporting the program, and what privileges are needed to execute the program. For a subprocess program (that is, a Grid-Flow Program), the process description segment is added after the program definition head. All of the information about the program will be registered into the Grid-Flow repository.

As an example, the following sentences in Figure 6 will generate a program record for the program named *TMPred*.

3.2.3. Process description

The major functionality of GFDL is to model all kinds of processes in the workflow system. A workflow process is a flexible combination of a set of activities. An activity is an atomic operation that cannot be divided further. A workflow process is responsible for organizing and scheduling activities in a workflow model, and accomplishing a particular function. The workflow model is described as a process description, which is a list of GFDL sentences that could be executed sequentially. Each GFDL sentence contains one or more expressions. The expression is a logical connection of a set of registered programs and data. More detailed information about the general syntax of the Grid-Flow process description language is listed in [43], since the GFDL is derived from the BioFlow language.

According to the workflow models provided in [50], there are four major structures (routings) used for organizing activities. They are *sequence structure*, *parallel structure*, *choice structure*, and *loop structure*. In sequence structure, the activities are carried out step-by-step in the order they appear. GFDL applies a program-driven structure to facilitate the design and execution of sequence structures. In the program-driven structure, registered programs interact with each other through their input/output parameters. That means one program's output is transferred to another program as an input parameter. All of the programs in an expression construct a streamlining structure with the output/input connections. For example, a set of programs with the relationships as shown in the left part of Figure 7 can be described as the expression on the right of the figure with GFDL.

In Figure 7, *ProgA, ProgB, ProgC*, and *ProgD* are all registered programs in the Grid-Flow repository. This program-driven structure means the output of *ProgA* will be transferred to *ProgB* as an input parameter. The outputs of *ProgB* and *ProgC* will be transferred to *ProgD* as input parameters. Consequently, the result of *ProgD* will be saved in the Grid-Flow repository as the final result of this sequence structure. The advantage of this program-driven structure is that it arranges the activities in a

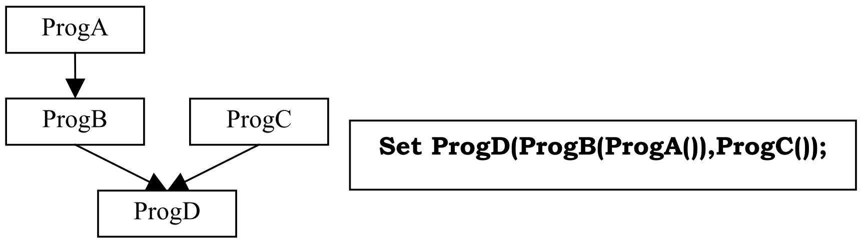


Figure 7. Example of sequence structure.

```
Set x=ProgA();
Set y=ProgB(x);
Set z=ProgC();
Set ProgD(y,z);
```

Figure 8. Extended expression of sequence structure.

natural sequence, in which the corresponding tools are used for real workflow processes. This makes it simple and intuitive for users to model their workflow procedure with GFDL.

Tasks in the Grid-Flow description can be organized into a nested structure as well as an extended structure. For example, the sequence of sentences in Figure 8 captures the same meaning as the single sentence in Figure 7 expresses via assignment statements.

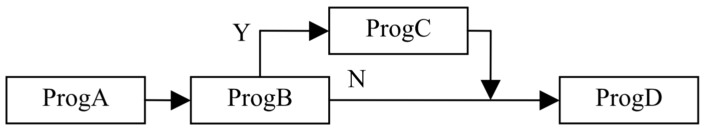
Sequence structure has an explicit dependency relationship between its activities. This dependence relationship not only relies on the input/output connection, but also reveals the execution order between activities. The execution order is dependent on the input/output connection. If no input/output connection exists between a set of activities, those activities can be executed in any order. A parallel structure is used to describe this unordered relationship between activities. Figure 7 provides a good example for the parallel structure. *ProgB* and *ProgC* have no input/output connection—they could be executed in any order. In the GFDL sentence, *ProgB* and *ProgC* are parallel and separated by a comma. The Grid-Flow engine will analyze the sentence and arrange it to be executed logically.

In the choice structure, one particular activity is selected out of two or more possible activities according to the running conditions set in the process. The following syntax describes the choice structure:

Set expression1 **When** expression2;

This syntax means *expression1* will only be executed once if the value of *expression2* is true. Otherwise, the Grid-Flow engine will skip *expression1* and execute the sentences following.

Every expression in the Grid-Flow language returns an integer value. Similar to C-like languages, a non-zero positive value of the expression implies true. The Grid-Flow engine controls the execution flow of the *Set–When* construct according to the expression value of expression2. For instance, in Figure 9, if *ProgB (ProgA())* returns true, *ProgC* will be executed before *ProgD*, otherwise only *ProgD*



```
Set ProgC() When ProgB(ProgA());
Set ProgD();
```

Figure 9. Example of the choice structure.

will be executed after the execution of *ProgB* and *ProgA*. The corresponding GFDL sentences are provided in the lower part of Figure 9.

The execution of activities in the loop structure also depends on the running conditions set at the beginning or end of the loop. Whenever the running conditions are true, the activities in the loop will be executed repeatedly. The different subtypes of loop structure are distinguished by the position where the running conditions are defined. If the running conditions are defined at the beginning of the loop, the workflow process first checks the value of the conditions, and then decides whether the activities in the loop should be executed. This kind of subtype is called a 'begin-controlled loop'. On the other hand, if the running conditions are defined at the end of the loop, the loop is called an 'end-controlled loop'. In the end-controlled loops, the activities included in the loop will be executed at least once, regardless of whether the running condition is true. The workflow process checks the value of running conditions after each execution and decides whether the activities should execute again. The GFDL supports two kinds of syntax, which conform to functionalities of the two subtype loops, respectively. These syntaxes are described as follows:

Set expression1 **While** expression2; (1)

This syntax is applied to the subtype 'begin-controlled loop', which means *expression1* will be executed if and only if the value of *expression2* is true. When this sentence is executed in the Grid-Flow engine, the value of *expression2* will be first checked. If true, the *expression1* could then be executed. If false, the engine will skip *expression1* and execute the following sentences.

Set expression1 **Until** expression2; (2)

This syntax is applied to the subtype 'end-controlled loop', which means *expression1* is guaranteed to execute once, and then the Grid-Flow engine will evaluate *expression2* and decide whether to continue executing *expression1*.

3.3. Graphical user interface

Although practically speaking, we are convinced that GFDL is powerful enough to describe most current existing workflow processes accurately. However, it has a key drawback, namely, it requires understanding and a proficient grasp of programming languages that is beyond the skill sets of many users. To reduce the gap between required proficiency of GFDL and users' capabilities, a graphical user interface is indispensable. Users specify a workflow process with tools that automatically translate the graphical model into the workflow description language using techniques of generative programming [51]. Thus, a visual model should be carefully selected to describe the workflow processes. It must be powerful with respect to description and comprehensible by users of a specific domain.

A workflow process is usually described as a Petri net by intuitively mapping programs as transitions, status as places, and data as tokens. Although the concept of a Petri net has been acknowledged to be one of the most powerful tools to describe multi-task procedures, especially for asynchronous and concurrent tasks, they are seldom used in practical workflow systems because they are not user-friendly. The work described in this paper employs Petri nets because it can describe data flow and control flow with a unique format. Petri nets can also construct the workflow hierarchy (that is, treat a sub-workflow as an entity of a large, complex workflow process) easily. To build up a user-friendly interface with Petri nets, the GME [14] is used in our system as the foundation of the graphical editor.

To model the workflow process with a Petri net in the GME, a meta-model is created (shown in Figure 10) to describe the basic modeling blocks of a Petri net, such as places, transitions, tokens, and connections [11]. Each part in the model has its own visualization and attributes. With these basic modeling blocks, users can construct their own Petri-net models by mapping their concept of a workflow process into combinations of building blocks. The meta-model is a correlated result of the Grid-Flow system.

One of the advantages of the GME is that it can automatically generate GFDL based on the Petri-net models, which lift the burden for the users of having to write code for any workflow process. The generation of GFDL from Petri-net models is implemented by a GME interpreter. From a Petri-net model, the GME interpreter first extracts the relationships between places and transitions, and then maps the relationships into GFDL control structures. Each pattern in a Petri-net model corresponds to a GFDL structure, as shown in Figure 11. This figure uses the concepts *AndJoin*, *AndSplit*, *OrJoin*, and *OrSplit* [50].

3.4. Grid-Flow engine

The Grid-Flow engine is the key component of the Grid-Flow system. When a workflow process is submitted by the user, the Grid-Flow engine checks the types of each data for the programs. This mission is accomplished by retrieving the metadata and provenance of the data, and matching these with the requirements of the programs. After type checking, the Grid-Flow engine drives forward the control flow by invoking programs through a program integration component, and feeding programs with corresponding input files. The way to invoke a set of programs (i.e. sequential or concurrent invocation of programs) is chosen by the Grid-Flow engine based on the data and programs interdependency among those programs. The communication between programs is based on file transfer. Each file acts as a token in the Petri net, whose arrival at the places triggers the execution of multiple ready-to-run programs, after all preconditions are met. The approach of reading and writing files acts as the inter-program communication mechanism since it is an easy way to implement it and a practical method to check the data dependency. Advanced models of pipelining data between programs in grid environments, including data streaming [52,53], will be applied to the Grid-Flow

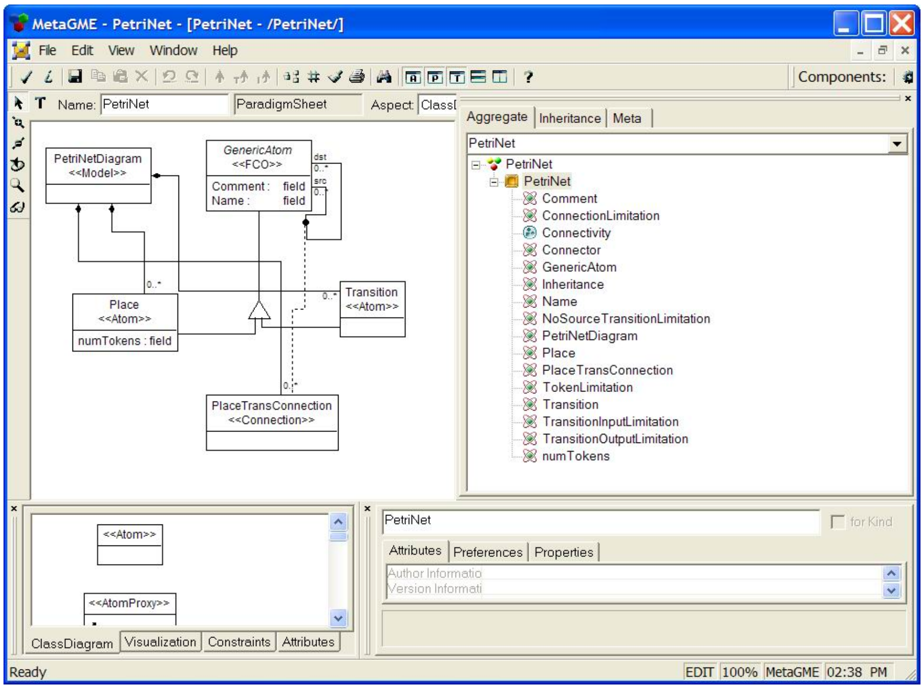
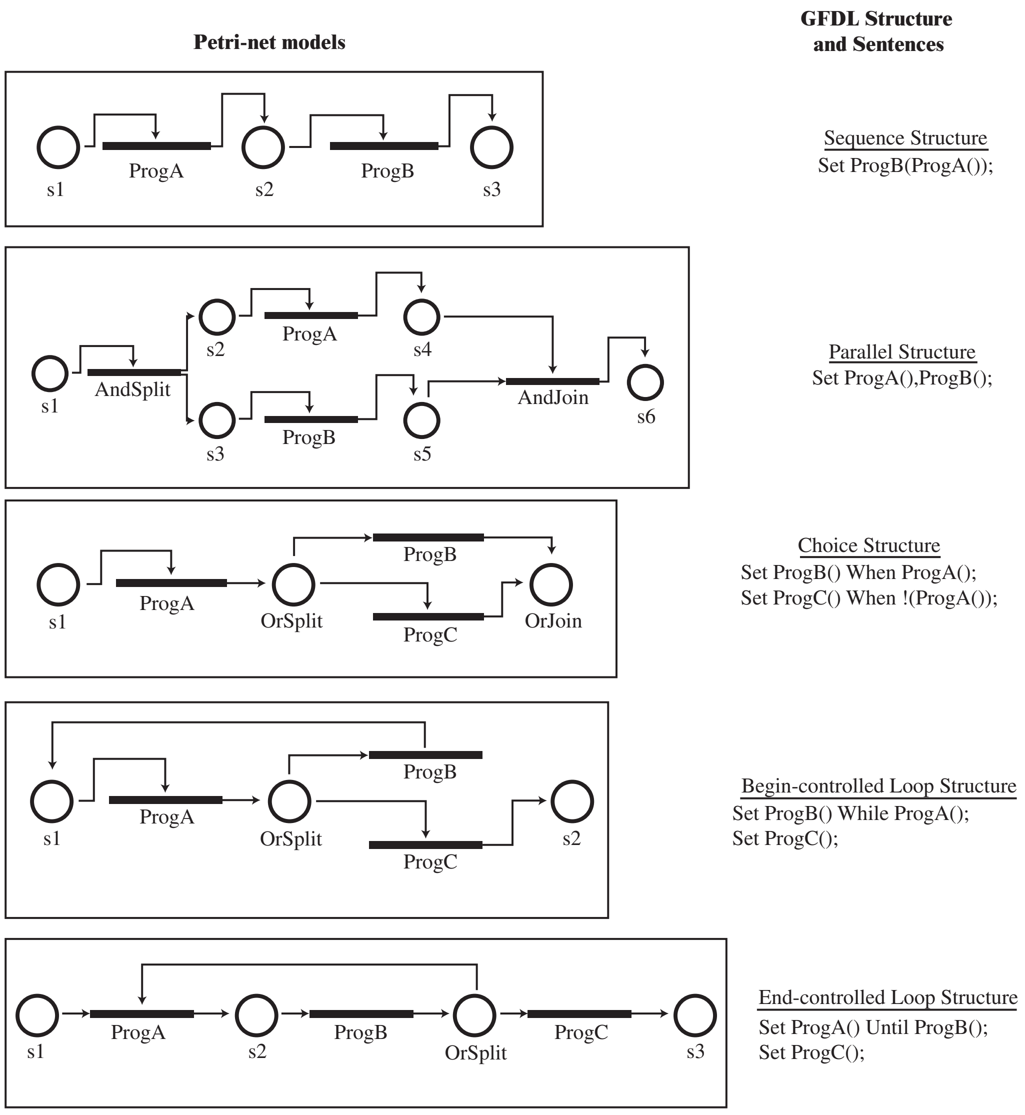


Figure 10. Meta-model of Petri net in the GME.

system with the improvement over time of ubiquitous Grid file systems. The Grid-Flow engine controls the scheduling and monitoring of the workflow process execution. The Grid-Flow engine is derived in terms of source code from the previous 'BioFlow' system [43], software originally authored by the first author of this paper.

The detailed inner structure of the Grid-Flow engine is described in Figure 12. The engine has the following five components, paralleling the BioFlow engine structure [43]:

1. *Syntax Analyzer* parses and compiles a workflow definition conveyed by GFDL to generate executable tasks;
2. *Repository Interface* maintains data and program information by interacting with the Grid-Flow repository;
3. *Data Manager* maintains application data through the Data Integration component;
4. *Program Manager* invokes the program tasks through the Program Integration module, while coordinating the acquisition and organization of local and remote site responses;
5. *Administrative Module* is responsible for the marshalling of the user applications, execution, management of the program flow, and task allocation and the coordination for the above-mentioned components. A Grid resource broker and scheduler [54] is employed by this module



Sequence Structure
Set ProgB(ProgA());

Parallel Structure
Set ProgA(),ProgB();

Choice Structure
Set ProgB() When ProgA();
Set ProgC() When !(ProgA());

Begin-controlled Loop Structure
Set ProgB() While ProgA();
Set ProgC();

End-controlled Loop Structure
Set ProgA() Until ProgB();
Set ProgC();

Figure 11. Mapping between Petri-net models and GFDL.

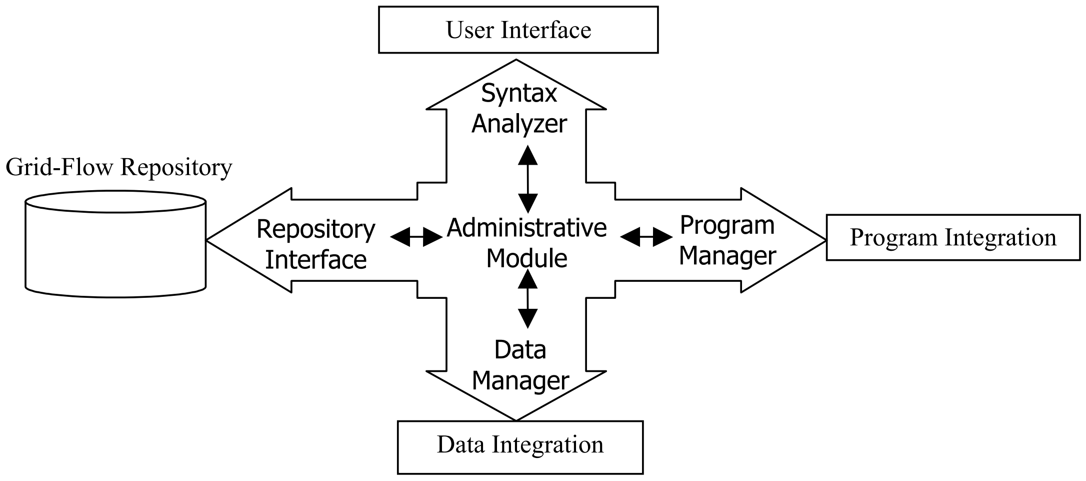


Figure 12. Architecture of the Grid-Flow engine (adapted from [43]).

to map the programs registered in the Grid-Flow repository to the real-world Grid services, and schedule the invocation of those Grid services via the Program Manager component. The design and implementation issues of this Grid resource broker and scheduler are discussed in [54].

In the Grid-Flow system, each component provides particular services to human users, or other components. At the same time, each component uses the services provided by the other components to fulfill its own function. Components need to communicate with each other through well-designed interfaces. The Grid-Flow engine component follows this mechanism and coordinates the execution of tasks using appropriate services provided by the subsystems described below.

3.5. Grid-Flow repository

This component assists in storing the information of registered data and program, process variables, and numerous system metadata necessary for application executions. It also maps all data and program registration information into system tables for use by the Grid-Flow engine at runtime. The Grid-Flow data repository is implemented on a relational database management system that supports the Open Database Connectivity (ODBC) and the SQL. Four primary system tables are maintained (similar to [43]).

- *System Information Table* records the metadata of the Grid-Flow system, including the execution path of the system, the location of the Grid-Flow repository, and all kinds of configurations of Grid-Flow processes.
- *Process Status Table* preserves all of the information about the current-running Grid-Flow processes. Each running Grid-Flow process places a record in this table, which records the ID

and name of the Grid-Flow process, the status (started, running, paused, waiting, and complete), the list of currently occupied resources, and the waiting list of resources. The Grid-Flow engine is responsible for the maintenance of this table.

- *Registered Data Table* holds all of the information about registered data. This information is collected from the registration procedure when the data are registered. Information that is persistently stored includes data name, data type, data source, and data access method. When the Grid-Flow process is running, some intermediate or temporary data information are also stored in the registered data table. The Grid-Flow engine will be charged with the job cleanup of this table after the execution of each workflow process.
- *Registered Program Table* holds all of the information about a registered program. This information is collected from the registration procedure when a program is registered. Stored information includes program name, program type, program path, and the program calling method. This information will be used when the Grid-Flow engine invokes the programs from the WebRun platform.

The Grid-Flow process descriptions, however, are stored in the local file system instead of the Grid-Flow repository. The process description in a file format is more portable than a record in the Grid-Flow repository table. Users can transfer their design of a workflow process by copying description files from one system to another. This design consideration facilitates the design and execution issues of a workflow process in a distributed computing environment.

3.6. Data/program integration

This section introduces the Data and Program Integration components of the Grid-Flow system.

3.6.1. Data integration

To handle the various formats of the different data sources, a Data Integration component is introduced into the Grid-Flow system. Data Integration works by accessing the data using self-describing information and then explicitly transforming the data into the desired data format according to the requirement. Working with the Data Integration component, the Grid-Flow engine can access the data without being concerned about the issues of data format. Thus, the Grid-Flow engine can interconnect the execution of programs by implicitly transforming the data between them.

When the Grid-Flow engine wants to access data, it first searches the registration information about that data in the Grid-Flow repository. After getting the required information, the Grid-Flow engine provides the Data Integration component with the registration information and its desired data format. Data Integration gets the data with its own data engine, and then transforms the data into the desired format with the Hyper Text Query Language (HTQL) [44]. Data with various data types are treated differently in the data engine. For data in the database, the data engine accesses the data using SQL. For data in plain text/HTML/XML format, the data engine accesses them through corresponding parsers. Data could come from different types of data sources, located either locally or on remote systems. The Data Integration component uses GridFTP (included in the Globus Toolkit [55]) to transfer data files when needed. Finally, the Data Integration component returns the data to the data engine and accomplishes its data access service.

3.6.2. Program integration

The Grid-Flow engine treats every Grid-Flow task as an executable program, which gets some data as input, processes the data, and outputs the results. Programs can be categorized into three types: internal programs, Grid-Flow programs, and OS programs. Internal programs and Grid-Flow programs are executed by the Grid-Flow engine. OS programs are executed by the Program Integration component. An OS program could be a program located on the local machine, or a distributed or parallel program located in a distributed environment, or even an Internet program accessible only with Web services. Therefore, a unified platform that can cover the diverse environmental details from the users is desired to enable the invocation of various programs on distributed and heterogeneous computing resources. The WebRun [21] system is employed in the Grid-Flow system in the role of a unified program execution platform. With WebRun, the Program Integration component provides the Grid-Flow system with a service to interact with any kind of program.

WebRun provides users with both a Web interface and a programmatic interface to command-line programs located on remote computing resources. The Web interface, a browser and servlet implementation in WebRun, supports the finding, starting, controlling, and utilization of command-line application programs. The programmatic interface enabled by the WASP (Web Applications and Services Platform) [56] client/service model provides a Java interface for accessing the programs stored on the remote resources that are otherwise not accessible. WebRun adopts a hierarchical architecture including three main components, the Program server, the Wasp server, and the Web server, as shown in Figure 13.

Program servers are distributed among all the hosts that contain programs participating in the WebRun system. Each program server will maintain a program repository, which resides at the backend and represents all available computing resources within that hosting environment. For each submitted job, the program server will be responsible for the maintenance of its running environment created in terms of its accompanying Resource Specification Language (RSL) [57].

The WASP server is initially centralized on one machine. It will maintain a program description store, which holds the property file for each program joining the program participation. Each program profile is accessed as responses to an invocation request, which in turn may be used to construct a job-submission form or a RSL associated with a given job. In essence, for each program description, Web services will be correspondingly generated and registered with the WASP server. The Web services provide a programmatic interface for servlets or programmers. Based on the Web service interfaces, dynamic Web pages may be developed via servlets, which are in turn accessible by users via Web browsers. This facilitates remote job submission and monitoring for a chosen program. Since OGSA [34] is still under development, the current implementation of WebRun creates Grid services using Java CoG-based Grid functionalities instead of exploiting OGSA-based Grid services directly. Work now is under way to map these services directly onto OGSA-based Grid services.

The WebRun system maintains a program repository that records all the programs available with the current users' credentials. The Grid-Flow engine, as a Web service client, utilizes the functionality of each program provided by the WebRun system. Thus, the Grid-Flow engine can access any program through a unified Web services interface. To facilitate the creation of Web services and the usage of the Globus Toolkit 3.2 (GT3.2) [47], the domain experts are provided with a graphical modeling environment for automated generation of Web services [58].

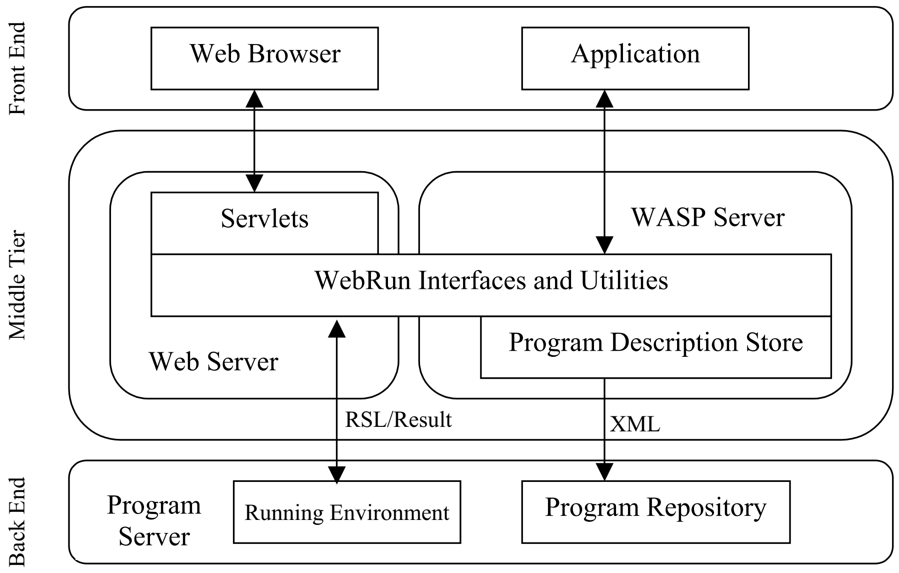


Figure 13. WebRun architecture.

4. AN ILLUSTRATIVE EXAMPLE

For illustrative purposes, this section presents an example to describe how to design and execute an application of biological data analysis within the Grid-Flow system. A broad overview of this example is provided first, followed by the details of each design and implementation aspect of the analysis.

A biologist interested in bio-data analysis needs to predict the functionality of a given protein sequence after completing the sequencing process that identifies each residue of that protein sequence [59]. The method of prediction [59] essentially relies on several aspects of related information, such as the functionality of similar protein sequences, the trans-membrane regions of the given sequence, and the promoter regions on the corresponding DNA sequence. The trans-membrane region identification is one of the most important steps that must be performed during the work of protein functionality prediction. A protein with several potential trans-membrane helices is likely to be an integral membrane protein that functions in transport or polymerization of molecules, while proteins with no trans-membrane helices are likely to be cytosolic and function in synthesis of molecular building blocks [59]. An overview of the process design on prediction of trans-membrane regions is illustrated in Figure 14. In this figure, rectangular boxes represent the beginning or end of the process, a set of document-style boxes describe the data transferred among the tasks, and rectangular boxes with rounded corners stand for the programs/tasks that process the data.

The process operates in the following way. When a biologist plans to analyze the trans-membrane regions of the given protein sequence, he or she initializes an instance of the workflow model of

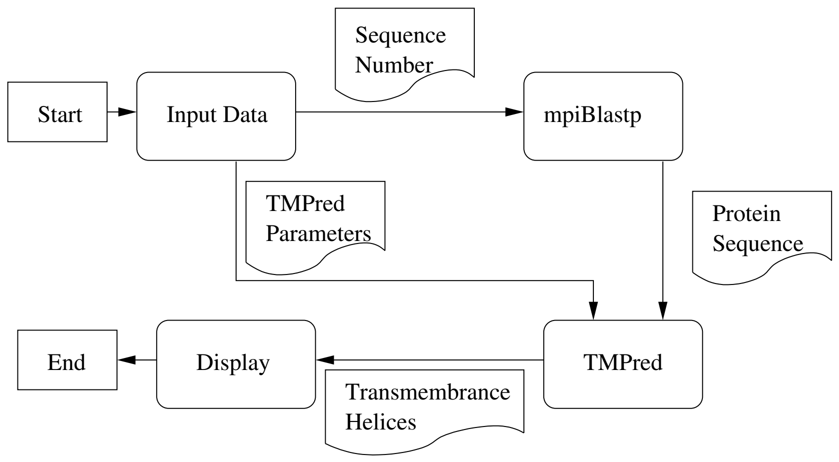


Figure 14. An example of analysis of trans-membrane regions (adapted from [59]).

predicting trans-membrane regions. This process first asks the biologist to provide the protein sequence accession number and decide the analysis parameters of the *TMPred* task [60], which is a program that can predict the trans-membrane regions against the protein sequences. The task *Input Data* is used to label this first step in the whole process. After this step, there are two data values returned: *Sequence Number* and *TMPred Parameters*. In the second task, the workflow system will automatically submit the *Sequence Number* to program *mpiblastp* [46] and search the protein sequence in the protein database located on the same resource as *mpiblastp*. The program *mpiblastp* is a parallel computing tool that can compare a protein sequence against a large protein database distributed on a cluster of computers. After using *mpiblastp*, the workflow system generates the data *Protein Sequence* and then submits this sequence to the third task, *TMPred*. The *TMPred* task uses two input data: one is the data *Protein Sequence* generated by *mpiblastp*; the other is the data *TMPred Parameters* that served as input in the first task. The *TMPred* task is responsible for predicting the trans-membrane regions according to the input protein sequence and parameters. Once the *TMPred* task is executed, it predicts the possible trans-membrane helices and displays the result to the end-user through the task *Display*. After the execution of *Display* task, the instance of the process goes to the end-state and its status is changed to ‘finish’.

To model this workflow process, a user may intuitively specify the Petri-net model as shown in Figure 15. From the structure viewpoint, the Petri-net model is similar to the data and program model described in Figure 14. The *Start* and *End* places indicate the beginning and end of the process. Each transition corresponds to a program in the data and program model. The data is represented by the token, and places as data and token containers connecting transitions. With this Petri-net model, the graphical user interface can automatically generate the GFDL as the following:

```
Set Display(TMPred(mpiBlastp(Input Data()),Input Data()));
```

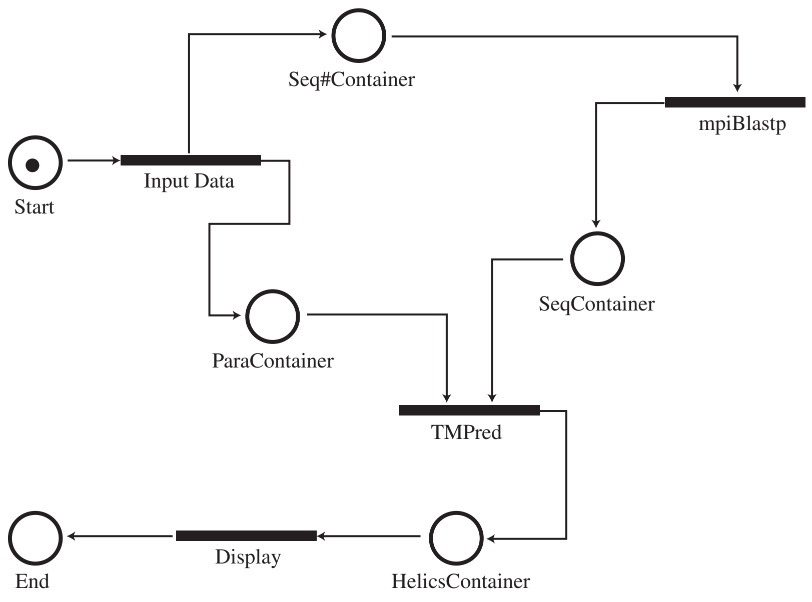


Figure 15. A Petri-net model of analysis of trans-membrane regions.

In this GFDL sentence, program *Display*, *TMPred*, *mpiBlastp*, and *Input Data* have already been registered with the Program Registration component before the design of the workflow process. Program *Display* and *Input Data* are programs on a local computer. Program *TMPred* is located on a program server within the same local network. Program *mpiBlastp* is running on a cluster whose head node shares the same network with the local computer. The data is input by the user through the program *Input Data*. Because the programs involved in this process are located on various computing resources distributed on the local network, the Grid-Flow engine should invoke the programs through the WebRun platform. The input data and temporary intermediate results are transferred by GridFTP [55]. The successful execution of this workflow process suggests that the Grid-Flow system can help the users' design and implement workflow processes with a user-friendly interface.

5. CONCLUSIONS AND FUTURE WORK

This paper analyzes three critical issues that distinguish the utility of Grid workflow systems and provides a new solution to address these in the Grid-Flow system. A brief survey is first presented on current Grid workflow systems, their modeling approaches, and their workflow languages. Based on this investigation, the drawbacks of existing systems are noted with current modeling approaches and workflow languages. In view of these drawbacks, a novel Grid workflow system is presented, which employs a Petri-net modeling approach based on a user-friendly graphical interface (GME), describes the workflow process definitions with a lightweight workflow language (GFDL), and integrates heterogeneous data and distributed analysis tools through a unified platform (WebRun). The meta-model to describe the Petri-net-based workflow process in GME is a novel achievement of this work. To the best of our knowledge, no similar meta-model has been developed in GME to address the modeling problem of workflow processes. Furthermore, we describe in detail the architecture of the Grid-Flow system, its workflow language GFDL, its graphical user interface, its workflow engine, its system repository, and its data and program integration mechanism. Finally, with the help of a case study, we illustrated the feasibility of the proposal of the Grid-Flow system and demonstrate the advantages of using our system.

The Grid-Flow system is an on-going project. It needs more future work to adapt itself to the ever-changing Grid computing environment. Possible future work may focus on the Petri-net modeling approach. As a modeling approach, Petri nets have their own limitations. As reported in [12], the size of the Petri net will increase dramatically when the modeled workflow process becomes more complicated. Although currently one can use hierarchical Petri-net models to describe the workflow process in a hierarchical structure, more elegant and sophisticated models are needed to organize the process in an abstract manner. Another concern about the future development of the system is to adapt the Grid-Flow language to business-oriented applications, such as banking, finance, and data mining. Based on the recognition that the essential characteristics of scientific and industrial workflows are similar, the authors are convinced that the Grid-Flow language with appropriate adjustments is capable of describing business-oriented workflow processes. The Grid-Flow language will be friendlier in the near future for designing and carrying out business operational patterns and protocols. Finally, the optimization and fault management features should also be considered in the future design of the Grid workflow system.

ACKNOWLEDGEMENTS

The Grid-Flow system is adapted and extended from the BioFlow system [43]. The BioFlow system was developed in the department of Computer Science and Engineering of Mississippi State University. The BioFlow project was guided by Dr Hasan M. Jamil, and received much help from Dr Liangyou Chen. The first author also acknowledges the enlightening discussion with Ms Ilkay Altintas about the Kepler system during the 10th Annual Summer Institute for Science Researchers in San Diego Supercomputer Center.

The first and second authors acknowledge the Department of Computer and Information Sciences at the University of Alabama at Birmingham for additional support of their research.

REFERENCES

1. Workflow Management Coalition. http://www.wfmc.org [31 May 2004].
2. Berman F, Hey A, Fox G (eds.). *Grid Computing: Making The Global Infrastructure a Reality*. Wiley: New York, 2003.
3. Foster I. What is the Grid? A Three Point Checklist, Daily News and Information for the Global Grid Community, 22 July 2002. http://www.gridtoday.com/02/0722/100136.html [31 May 2004].
4. Erwin DW, Snelling DF. *UNICORE: A Grid Computing Environment* (*Lecture Notes in Computer Science* vol. 2150). Springer: Berlin, 2001; 825–834.
5. Lorch M, Kafura D. Symphony—a Java-based composition and manipulation framework for computational Grids. *Proceedings of 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2002)*, Berlin, Germany, 21–24 May 2002. IEEE Computer Society Press: Los Alamitos, CA, 2002.
6. Altintas I, Berkley C, Jaeger E, Jones M, Ludaescher B, Mock S. Kepler: Towards a Grid-enabled system for scientific workflows. *Proceedings of Workflow in Grid Systems Workshop in GGF10*, Berlin, Germany, March 2004. GGF: Berlin, 2004.

7. Laszewski Gv, Amin K, Hategan M, Zaluzec NJ, Hampton S, Rossi A. GridAnt: A client-controllable Grid workflow system. *Proceedings of 37th Hawaii International Conference on System Science*, Island of Hawaii, Big Island, 5–8 January 2004. IEEE Computer Society Press: Los Alamitos, CA, 2004.
8. Deelman E, Blythe J, Gil Y, Kesselman C, Mehta G, Patil S, Su M-H, Vahi K, Livny M. Pegasus: Mapping scientific workflows onto the Grid. *Proceedings of The 2nd European Across Grids Conference*, Nicosia, Cyprus, 28–30 January 2004. Springer: Berlin, 2004.
9. Cao J, Jarvis SA, Saini S, Nudd GR. GridFlow: Workflow management for Grid computing. *Proceedings of 3rd International Symposium on Cluster Computing and the Grid*, Tokyo, Japan, 12–15 May 2003. IEEE Computer Society Press: Los Alamitos, CA, 2003; 198–205.
10. Shields M, Taylor I. Programming scientific and distributed workflow with Triana services. *Proceedings of Workflow in Grid Systems Workshop in GGF10*, Berlin, Germany, March 2004. GGF: Berlin, 2004.
11. Peterson JL. Petri Nets. *ACM Computing Surveys* 1977; **9**(3):223–252.
12. Hoheisel A. User tools and languages for graph-based Grid workflows. *Proceedings of Workflow in Grid Systems Workshop in GGF10*, Berlin, Germany, March 2004. GGF: Berlin, 2004.
13. Zhang S, Gu N, Li S. Grid workflow based on dynamic modeling and scheduling. *Proceedings of International Conference on Information Technology: Coding and Computing (ITCC2004)*, Las Vegas, NV, 5–7 April 2004, vol. 2. IEEE Computer Society Press: Los Alamitos, CA, 2004; 35–39.
14. Ledeczi A, Bakay A, Maroti M, Volgyesi P, Nordstrom G, Sprinkle J, Gabor K. Composing domain-specific design environments. *IEEE Computer* 2001; **34**(11):44–51.
15. Oinn T, Addis M, Ferris J, Marvin D, Greenwood M, Carver T, Wipat A, Li P. Taverna, lessons in creating a workflow environment for the life sciences. *Proceedings of Workflow in Grid Systems Workshop in GGF10*, Berlin, Germany, March 2004. GGF: Berlin, 2004.
16. Krishnan S, Wagstrom P, Laszewski Gv. GSFL: A Workflow Framework for Grid Services [Draft], 19 July, 2002. http://www.globus.org/cog/papers/gsfl-paper.pdf [31 May 2004].
17. Andrews T *et al*. Specification: Business Process Execution Language for Web Services Version 1.1, 5 May, 2003. http://www-106.ibm.com/developerworks/library/ws-bpel/ [31 May 2004].
18. Extensible Markup Language (XML). http://www.w3.org/XML/ [31 May 2004].
19. Graham GE, Evans D, Bertram I. McRunjob: A high energy physics workflow planner for Grid production processing. *Proceedings of Computing in High Energy and Nuclear Physics*, La Jolla, CA, 24–28 March 2003.
20. McCann KM, Yarrow M, DeVivo A, Mehrotra P. ScyFlow: An environment for the visual specification and execution of scientific workflows. *Proceedings of Workflow in Grid Systems Workshop in GGF10*, Berlin, Germany, March 2004. GGF: Berlin, 2004.
21. Guan Z, Liu Y, Velusamy V, Bangalore PV. WebRun: A unified platform supporting Grid computing environment. *Technical Report UABCIS-TR-2004-1404-1*, Department of Computer and Information Sciences, University of Alabama at Birmingham, 2004.
22. Bhatia D, Burzevski V, Camuseva M, Fox G, Furmanski W, Premchandran G. WebFlow—a visual programming paradigm for Web/Java based coarse grain distributed computing. *Concurrency: Practice and Experience* 1997; **9**(6):555–577.
23. Common Component Architecture. http://www.cca-forum.org [31 May 2004].
24. Krishnan S, Bramley R, Gannon D, Govindaraju M, Alameda J, Alkire R, Drews T, Webb E. The XCAT science portal. *Proceedings of Supercomputing (SC2001)*, Denver, CO, 10–16 November 2001. ACM Press: New York, 2001.
25. Ant—A Java-based build tool. http://ant.apache.org [31 May 2004].
26. Dozsa G, Kacsuk P, Lovas R, Podhorszki N, Drotos D. P-GRADE: A graphical environment to create and execute workflows in various Grids. *Proceedings of Workflow in Grid Systems Workshop in GGF10*, Berlin, Germany, March 2004. GGF: Berlin, 2004.
27. Deelman E, Blythe J, Gil Y, Kesselman C. Workflow management in GriPhyN. *Grid Resource Management: State of the Art and Future Trends*, Nabrzyski J, Schopf JM, Weglarz J (eds.). Kluwer Academic: Norwell, MA, 2003.
28. DAGMan (Directed Acyclic Graph Manager), 2002. http://www.cs.wisc.edu/condor/dagman [31 May 2004].
29. Fraunhofer Resource Grid. http://www.fhrg.fhg.de [31 May 2004].
30. Leyman F. Web Services Flow Language (WSFL 1.0), 2001. http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf [31 May 2004].
31. Thatte S. XLANG: Web Services for Business Process Design, 2001. http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm [31 May 2004].
32. Web Services Conversation Language (WSCL 1.0), The Hewlett-Packard Company, 2002. http://www.w3.org/TR/wscl10/ [31 May 2004].
33. Arkin A *et al*. Web Services Choreography Interface (WSCI) 1.0 Specification, 2002. http://wwws.sun.com/software/xml/developers/wsci/index.html [31 May 2004].

34. Foster I, Kesselman C, Nick J, Tuecke S. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, 2002. http://www.globus.org/research/papers/ogsa.pdf [31 May 2004].
35. Christensen E, Curbera F, Meredith G, Weerawarana S. W3C Web Services Description Language (WSDL) 1.1, W3C Note March 15, 2001. http://www.w3.org/TR/wsdl [31 May 2004].
36. Cybok D. A Grid workflow infrastructure. *Proceedings of Workflow in Grid Systems Workshop in GGF10*, Berlin, Germany, March 2004. GGF: Berlin, 2004.
37. Neema S, Bapty T, Gray J, Gokhale A. Generators for synthesis of QoS adaptation in distributed real-time embedded systems. *Proceedings of 1st ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering (GPCE '02)*, Pittsburgh, PA, 6–8 October 2002 (*Lecture Notes in Computer Science*, vol. 2487). Springer: Berlin, 2002; 236–251.
38. Sztipanovits J, Karsai G. *Generative Programming for Embedded Systems* (*Lecture Notes in Computer Science*, vol. 2487). Springer: Berlin, 2002; 32–49.
39. Karsai G, Maroti M, Ledeczi A, Gray J, Sztipanovits J. Composition and cloning in modeling and meta-modeling. *IEEE Transactions on Control System Technology (Special Issue on Computer Automated Multi-Paradigm Modeling)* 2004; **12**(2):263–278.
40. Booch G, Rumbaugh J, Jacobson I. *The Unified Modeling Language User Guide* (*Addison-Wesley Object Technology Series*), Booch G, Jacobson I, Rumbaugh J (eds.). Addison-Wesley: Reading, MA, 1998; 482.
41. Warmer J, Kleppe A. *The Object Constraint Language Second Edition, Getting Your Models Ready for MDA* (2nd edn). Addison-Wesley: Boston, MA, 2003; 206.
42. Ledeczi A, Davis J, Neema S, Agrawal A. Modeling methodology for integrated simulation of embedded systems. *ACM Transactions on Modeling and Computer Simulation* 2003; **13**(1):82–103.
43. Guan Z, Jamil HM. Streamlining biological data analysis using BioFlow. *Proceedings of the 3rd IEEE Symposium on Bioinformatics and Bioengineering (BIBE03)*, Bethesda, MD, 10–12 March 2003. IEEE Computer Society Press: Los Alamitos, CA, 2003.
44. Chen L, Jamil HM. On using remote user defined functions as wrappers for biological database interoperability. *International Journal of Cooperative Information Systems (Special Issue on Data Management and Modeling Support in Bioinformatics)* 2003; **12**(2):161–195.
45. Cannan SJ, Otten GAM. *SQL—The Standard Handbook*. McGraw-Hill: New York, 1992; 584.
46. Darling AE, Carey L, Feng W. The design, implementation, and evaluation of mpiBLAST. *Proceedings of ClusterWorld Conference and Expo in Conjunction with the 4th International Conference on Linux Clusters: The HPC Revolution 2003*, San Jose, CA, June 2003.
47. Foster I, Kesselman C. The Globus toolkit. *The Grid: Blueprint for a New Computing Infrastructure*, Foster I, Kesselman C (eds.). Morgan Kaufmann: San Francisco, CA, 1999; ch. 11, 259–278.
48. Swofford DL. *PAUP*. Phylogenetic Analysis Using Parsimony (*and Other Methods) Version 4*. Sinauer Associates: Sunderland, MA, 2002.
49. ClustalW. http://www.ebi.ac.uk/clustalw [31 May 2004].
50. van der Alst W, van Hee K. *Workflow Management: Models, Methods, and Systems*. MIT Press: Cambridge, MA, 2002; 368.
51. Czarnecki K, Eisenecker U. *Generative Programming: Methods, Tools, and Applications* (1st edn). Addison-Wesley: Boston, MA, 2000; 864.
52. McCune D, Parashar M, Beck M, Klasky S, Bhat V, Atchley S. High performance threaded data streaming for large scale simulations. *Proceedings of 5th IEEE/ACM International Workshop on Grid Computing*, Pittsburgh, PA, November 2004. IEEE Computer Society Press: Los Alamitos, CA, 2004.
53. Wu JS-C, Sussman A. Flexible control of data transfers between parallel programs. *Proceedings of 5th IEEE/ACM International Workshop on Grid Computing*, Pittsburg, PA, 2004. IEEE Computer Society Press: Los Alamitos, CA, 2004; 226–234.
54. Afgan E, Velusamy V, Bangalore P. Grid resource broker with application profiling and benchmarking. *Proceedings of European Grid Conference*, Amsterdam, Netherlands, 14–16 February 2005. Springer: Berlin, 2005.
55. The GridFTP Protocol and Software. http://www.globus.org/datagrid/gridftp.html [31 May 2004].
56. Systinet. Systinet Products Overview. http://www.systinet.com/products/overview [31 May 2004].
57. The Globus Resource Specification Language RSL v1.0. http://www.globus.org/gram/rsl_spec1.html [31 May 2004].
58. Hernandez F. Domain specific models and the globus toolkit. *Technical Report UABCIS-TR-2004-0504-1*, Department of Computer and Information Sciences, University of Alabama at Birmingham, 2004.
59. Lawerence ML, Banes MM, Azadi P. The Edwardsiella Ictaluri O Polysaccharide Biosynthesis Gene Cluster: Correlation between Predicted Enzyme Functions and O Polysaccharide Composition, Unpublished Manuscript, Mississippi State University, 2003.
60. TMpred—Prediction of Transmembrane Regions and Orientation. http://www.ch.embnet.org/software/TMPRED_form.html [31 May 2004].