GRID-FLOW: A GRID-ENABLED SCIENTIFIC WORKFLOW SYSTEM WITH A PETRI NET-BASED INTERFACE

by

ZHIJIE GUAN

A DISSERTATION

Submitted to the graduate faculty of The University of Alabama at Birmingham,
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

BIRMINGHAM, ALABAMA

2006

ABSTRACT OF DISSERTATION
GRADUATE SCHOOL, UNIVERSITY OF ALABAMA AT BIRMINGHAM

Degree Ph.D. Program Computer and Information Sciences

Name of Candidate Zhijie Guan

Committee Chair Anthony Skjellum

Title Grid-Flow: A Grid-Enabled Scientific Workflow System With a Petri Net-Based Interface

Advances in computer power, network speed, and storage capacity have enabled scientists to explore research issues in their respective domains at scales both finer and greater 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 available massive computation, a suitable scientific workflow system is needed to help users to manage both data and programs, and to design reusable procedures of scientific experimental tasks. This dissertation describes the design and prototyping of such a scientific workflow infrastructure, the Grid-Flow system, which assists researchers in specifying scientific experiments using a Petri net-based interface. The Grid-Flow infrastructure is designed as a Service-Oriented Architecture (SOA) with multi-layer component models.

The major contributions of Grid-Flow are as follows: 1) a new, light-weight, programmable Grid workflow language, called the Grid-Flow Description Language (GFDL), is provided to describe the workflow process in a Grid environment; 2) a Petri net-based user interface, based on the Generic Modeling Environment (GME), is demonstrated to help users design the workflow process with a Petri net model; and 3) a data and program integration component of the Grid-Flow system is presented to integrate various data and non-interactive programs into the system.

This work furthermore contributes to design methodologies (data/program registration, workflow description and execution, and data/program integration), underlying models of modern scientific workflows (Petri net model and Data/Program Chart), the integration and orchestration of online data and programs within workflow cases, as well as the strategies of wrapping existing programs as web/Grid services in a Grid environment. Two workflow case studies, transmembrane region analysis and protein function and expression, are modeled and implemented by the Grid-Flow system in order to demonstrate the capability and usability of this workflow management system.

DEDICATION

I dedicate this dissertation to my mother and my wife, without whom I would not be where I am today.

ACKNOWLEDGMENTS

I would like to take this opportunity to thank the large number of people without whom I could not have reached this point.

First of all, I must thank my advisor, Dr. Anthony Skjellum, who not only motivates and guides me to take up this path, but also encouraged and supported me throughout my pursuit of the Ph.D. degree. Without his encouragement and guidance, my Ph.D. work would not have been done. Dr. Skjellum, as an expert in high performance computing and education, is always giving me illuminating advice and secure guidance at the right times. I am also grateful for the financial support provided by Dr. Skjellum during the second half of my Ph.D. study, as a Research Assistant in the High Performance Computing Laboratory (HPCL) in the Department of Computer and Information Sciences (CIS) at University of Alabama at Birmingham (UAB).

I also wish to thank my committee members: Dr. Purushotham Bangalore, Dr. Jeffrey Gray, Dr. Elliot Lefkowitz, and Dr. Tracy Hamilton for serving on my committee and providing me valuable suggestions and feedbacks. I thank Dr. Bangalore for leading me into the Grid computing area, Dr. Gray for enlightening me on model integrated computing and the Petri net modeling approach, as well as Dr. Lefkowitz and Dr. Hamilton for offering me suggestions on the area of Bioinformatics workflow applications.

I have much enjoyed the cooperative and supportive research environment of HPCL. I express my gratitude to the laboratory members Vijay Velusamy, Yin Liu,

Vetria Byrd, and William Johns for all the collaboration and warm-hearted encouragement. In particular, I thank Vijay and Yin for the cooperation on the WebRun project.

The graduate program at the Department of CIS at University of Alabama at Birmingham (UAB) is unique and multidisciplinary. I am grateful to UAB for providing me an opportunity to join this program. I thank this program for opening a door for me to cooperate with all the professors, instructors, and graduate students in this department. Specially, I thank Enis Afgan in the Collaborative Computing Laboratory for his cooperation on the G-BLAST project, Francisco Hernandez for his help on the workflow modeling in the Generic Modeling Environment, and Jing Zhang and Yuehua Lin in the SOFTCOM Laboratory for their comments on model-integrated computing. Thanks to the system administration staff, especially Fran Fabrizio, and the office administration staff, Kathy Baier and Janet Sims, for all their support.

This work was started while I was working as a Research Assistant in the SmartDB group led by Dr. Hasan Jamil at the Department of Computer Science and Engineering at Mississippi State University. I am grateful to Dr. Jamil for introducing me to the research field of scientific workflow and for enlightening me the fundamental knowledge of workflow management. I appreciate the SmartDB group members Dr. Liangyou Chen, Jianming Shi, and Nan Wang for their contributions and assistance throughout building the prototype BioFlow system. Without their encouragement and assistance, I would not have undertaken my dissertation in the area of scientific workflow system.

Furthermore, this work would not be possible without the active collaboration and interaction with Bioinformatics researchers at Mississippi State University. My special thanks go to Dr. Mark Lawrence in the College of Veterinary Medicine, Dr. Mark

Fishbein in the Department of Biological Sciences, and Dr. John Boyle in the Department of Biochemistry and Molecular Biology for providing workflow examples and for being the users of the workflow management system. I am grateful for their input, suggestions, and discussions during this work. Particularly I express my thankfulness to Dr. Fishbein for bringing me into the fascinating areas of Bioinformatics and System Biology. I also thank Dr. Xiufeng Wan, who would always like to share his knowledge and experience in the Bioinformatics research and encourage me to work hard toward my goal.

I express my gratitude to the Kepler scientific workflow research group at the San Diego Supercomputer Center (SDSC), especially the group leader Ms. Ilkay Altintas, for the enlightening discussions on workflow modeling approaches. I also thank Dr. Mark Miller for helping and supporting me during my internship in the Kepler group, as well as offering me the superb opportunity to continue working on the research of scientific workflow systems as a postdoctoral fellow at SDSC.

Thankfulness also goes to the editors of this dissertation. Their comments spurred me on to improve it and make it more readable and more consistent. So I want to acknowledge my debt of gratitude to Dr. Jeffery Gray and Ms. June Vayo. Dr. Gray donated large amounts of time to help me revise and polish the dissertation. He always provided profound comments relating to my dissertation. Ms. June Vayo helped me on fixing semantic bugs and by suggesting appropriate words to clearly convey my ideas.

Finally, I would not have been made this achievement in my life without the loving support of my mother Yufen Zhang and my wife Yin Liu. Their support and understanding during my Ph.D. study is deeply appreciated.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

Page

TABLE OF CONTENTS (Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF ABBREVIATIONS

CGI	Common Gateway Interface
DAG	Directed Acyclic Graph
DDF	Data Description File
DIR	Data Item Record
DR	Data Record
GFDL	Grid-Flow Description Language
GGF	Global Grid Forum
GME	Generic Modeling Environment
GWSDL	Grid Web Service Description Language
HTQL	Hyper Text Query Language
MIC	Model-Integrated Computing
ODBC	Open DataBase Connectivity
OGSA	Open Grid Service Architecture
OS	Operating System
PDF	Program Description File
PR	Program Record
SDE	Service Data Element
SOA	Service Oriented Architecture
SQL	Structured Query Language

LIST OF ABBREVIATIONS (Continued)

URL	Uniform Resource Locator
WFMS	WorkFlow Management System
XML	eXtensible Markup Language

1 INTRODUCTION

Computer information systems in earlier times were designed to support the execution of individual tasks, such as simulating chemical reactions (Allen & Tildesley, 1987) and solving nonlinear equations (Zeleznik, 1968). Currently advances in networking infrastructure, distributed processing, and collaborative computing enable the orchestration of these individual tasks as a whole computing process over heterogeneous data and computing resources. It is no longer sufficient for current information systems to focus only on individual tasks. More applications require computer systems to control, monitor, and support the logistical aspects of business and/or scientific processes. That is, the information system needs not only to support computational operations, but it also has to manage the flow of work through applications. Many researchers and organizations have identified the need for theories, techniques, and tools to support the management of flows of work, *a.k.a.* workflows. Therefore, the concept of *workflow management* was introduced to satisfy these needs (Hayes & Lavery, 1991; Koulopoulos, 1995).

1.1 Workflow Management System: History

According to the definition from the Workflow Management Coalition (WfMC) (*Workflow Management Coalition*), 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 business process is the key factor in workflow management, whose major function is to support the definition, execution, registration and control of these processes. The ultimate goal of workflow management is to make sure the appropriate actors perform the proper activities at the right time, for the correct duration of time, assigned to the right resource or resources, and with the required human interactions. Although it is possible to implement workflow management without using any computers, most researchers in this area associate workflow management with the workflow management system based on computers and computing devices. The WfMC (*Workflow Management Coalition*) defines a workflow management system as *"A system that completely defines, manages, and executes workflows through the execution of software whose order of execution is driven by a computer representation of the workflow logic."* A description of the most commonly used concepts, techniques and methods for workflow management can be found in section 2.1.

Workflow management systems emerged in the 1990's (Aalst & Hee, 2002). Before that time, most business processes were hard-coded within applications. People working on the UNIX platform at that time wrote shell scripts to control the execution procedure of multiple programs or tasks. For example, a user would define in a script that Program B should get Program A's output file as its input. In this case, Program B knows information about and is closely tied to Program A. This approach could cause two problems. First, whenever a process changed, the application (script file) would need to be modified manually. Second, the modules for reading and writing files needed to be implemented in each application, and it was hard to accommodate future changes, such as changes for the input/output file format. Therefore, a Workflow Management System (WFMS) is useful and needed to define the tasks and interfaces between software parts and applications, register such programs and tools, control the desired workflow, and execute the whole process. Some prototype WFMSs appeared in research and industry during the 1990's, demonstrating the broad recognition of the potential utility of work-flow management (Aalst, 1994; Hayes & Lavery, 1991; Stein, Rozen, & Goodman, 1995).

The history of workflow management systems shows the evolution of the architecture and design of the contemporaneous information systems (Aalst, 1998). Figure 1 illustrates the historical perspective of WFMS development, based on the architecture and constructing components of the WFMS. In the 1960's, applications ('APPL' for short in the graph) were directly based on the operating system (OS) (Aalst & Hee, 2002). Each of these applications handled the user interface, data storage, and flow control in an ad hoc manner. Information systems with multiple applications would rarely be constructed at that time because of the lack of unique interface between such standalone applications. Database Management Systems (DBMSs) (Ramakrishnan & Gehrke, 2003) were introduced in the 1970's to store the application data and hence reduce the burden of data management for applications. DBMSs also provided a uniform interface for data storage and retrieval, which facilitated the communication between applications. User Interface (UI) was subsequently brought into the architecture of information systems in the 1980's. The emergence of the UI (Jacko & Sears, 2003) enabled the application developer to separate the user interaction from the process logic of the application. Therefore, users were shielded from the implementation details of the application by interacting only with the interface.

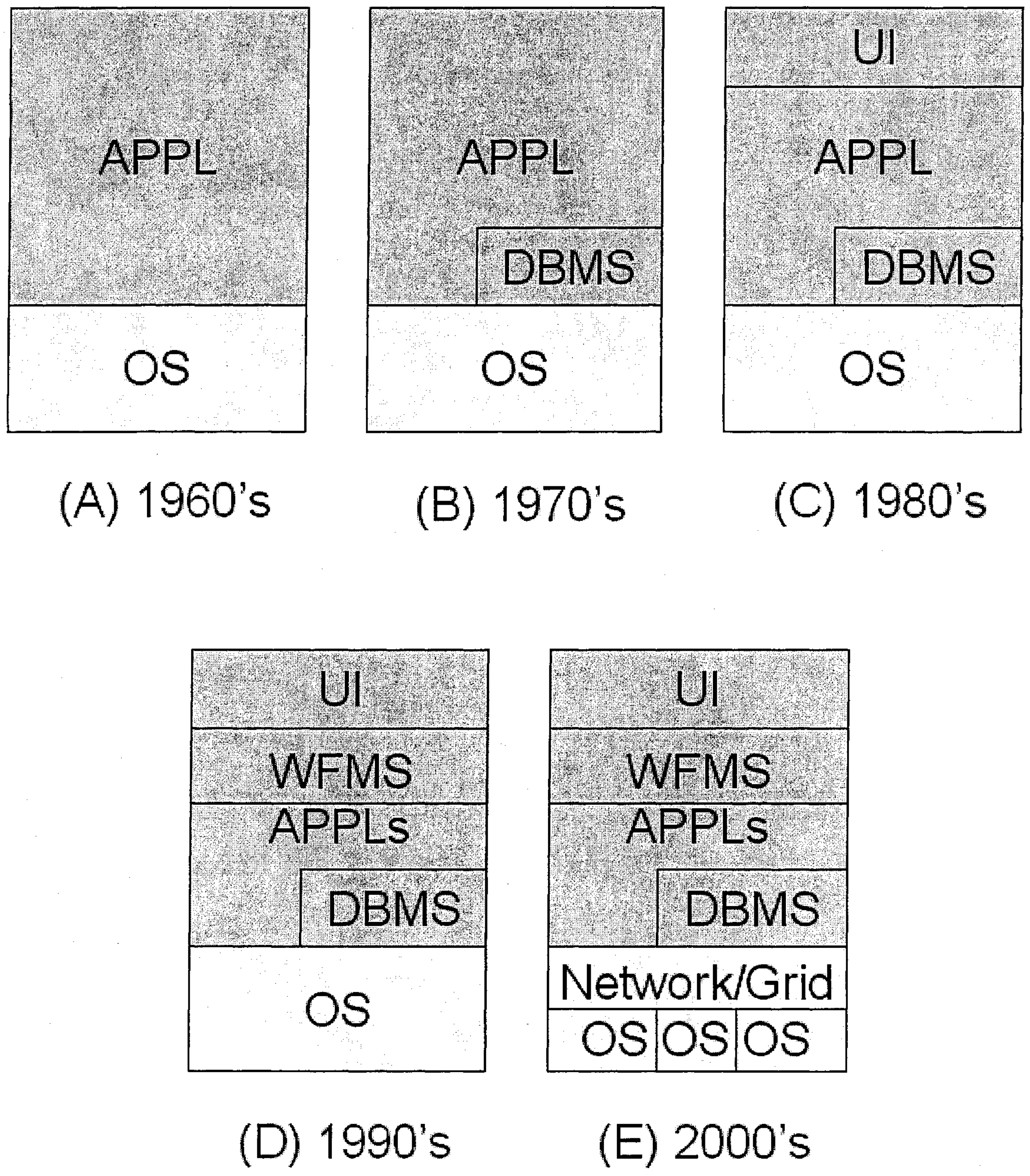


Figure 1. History of Workflow Management Systems.

(adapted from (Aalst, 1996) and redrawn)

With more applications involved into the information system in the 1990's, a layer of WFMS was added in order to manage the invocation of applications, as well as the data and control flow among them. This WFMS layer provided the user, through the UI, with a conceptual overview of the cooperation of applications in the system. It also freed the applications from the burden of control flow management and interaction.

As the distributed and Grid computing technologies matured gradually in the last decade, developers could make applications independent from the underling operating systems by using web or Grid services. WFMSs can take advantage of this opportunity to enable the execution of workflow process in a larger scope (that is, crossing the boundary of a single OS). WFMSs will eventually be able to control the process logic of an information system over Grid environments, which can span security, ownership, and geographic domains.

1.2 Grid Workflow System: State of the Art

Significant advances of computing technologies in recent years have enabled scientists to explore research issues in their areas at scales greater and also finer than ever before. Researchers in many scientific fields, such as system biology (Kitano, 2002), cheminformatics (Leach & Gillet, 2003), climatology (Thompson & Perry, 1997), and astronomy (Brumfiel, 2002), have seen the implications of the emerging powerful and effective data analysis tools enabled by computer technologies. They have urgently demanded the development of a problem-solving environment capable of utilizing distributed data and computing resources, orchestrate the data analysis tools crossing various platforms, and integrate efforts from collaborating participants of data/computation intensive applications. To satisfy this demand, two computer technologies, workflow management (*Workflow Management Coalition*) and Grid computing (Berman, Fox, & Hey), are used together to provide a promising solution of a distributed, collaborative workflow system, which is a Grid workflow system.

Compared with business processes, scientific workflows have the following special characteristics:

- Scientific workflows are used as part of accomplishing scientific research. Most of the workflows in research are based on academic activities, such as data analysis, methods testing, and hypothesis validation. In most cases, the scientist who executes the workflow is both the designer and the end-user of the workflow. This means that such workflows are often designed in an informal and spontaneous way. As compared to scientific workflows, business processes usually involve multiple computing and human resources from various organizations. Efficiently organizing and utilizing those resources commonly require business workflows to be carefully specified via a strict protocol, and also to be suitable for persistent execution. Informally, the nature of scientific workflows requires the supporting WFMS to provide relatively more flexible and comprehensible modeling approaches as compared to the methods used in current industrial WFMSs, in order to accommodate the ever-changing academic workflow designs.
- While scientific workflows are mostly concerned with throughput of data between and among various programs, tools, and services, business workflows focus on scheduling task execution, and such scheduling involves dependencies that may not be driven by data and that may well include human interaction. Most programs and services used in scientific workflows are provided by computer systems. These com-

puter systems may be distributed at physical locations and heterogeneous in system architecture and software. The applications used in business workflows are usually based on systems in a commercial organization. Since the systems in an organization are generally regulated by management rules, business workflows are more likely for now to be performed in a centralized and homogeneous environment (businesses vary in their conservativeness in this regard). Thus, scientific WFMSs generally need to have more adaptability and adjustability to various systems.

- Certain scientific workflows are used to analyze huge amounts of scientific data, such as that which could be generated by the human genome project (*Genomics and Its Impact on Science and Society: A 2003 Primer*, 2003), by black hole exploration experiment (Jacob et al., 2004), or by simulation of chemical reactions (Camarda, He, & Bishop, 2001), among others. Scientific WFMSs need to provide the ability to transfer that data over the network and analyze it with sufficient computing power. Moreover, the analysis of large amounts of data may take a long time, which of course prolongs the execution time of workflows to periods of time ranging from several days to several months. Some features for long-running workflows, particularly state information (Lomet & Weikum, 1998), bookkeeping (Aalst & Hee, 2002), check point (Aalst & Hee, 2002), and fault tolerance (Lomet & Weikum, 1998) should be supported by a scientific WFMS.
- The definition of scientific workflows needs to be interchangeable among various workflow systems. Unlike business workflows, scientific workflows usually act as communication tools conveying new academic ideas and design decisions. Since currently some scientific workflow management platforms exist with different modeling

methods and execution mechanisms, the workflow research community evinces great enthusiasm for the workflow language that can standardize the description of workflow process and therefore translate or transform workflow definitions among different WFMSs.

Although the definition of workflow in the scientific area may differ from business workflow through dual emphases on scientific data and analytical process requirements, they share common, essential characteristics of a procedure that applies specific activities on the data according to rules. Since the data and computing may be partitioned in a physically distributed environment, the workflow management system needs a mechanism to handle the data transfer issue and invoke the computational tools over a distributed and heterogeneous platform. This mechanism is precisely the capability that Grid computing technology strives to achieve in scientific environments (Foster, 2002). Grid computing technology satisfies the requirement by providing a new computing infrastructure for large-scale sharing of resources and distributed system integration. A Grid, according to the definition provided by Ian Foster in (Foster), 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 could support workflow definition (including process, resource, program defining), workflow execution, workflow monitoring and administration over a set of dispersed data/computing resources.

1.3 Research Issues for Grid Workflow System

Although a set of candidate features may be able to distinguish a Grid workflow system from others, there are only three major issues of a Grid workflow system needed to identify the system and even decide the system performance. These three issues are the model used to describe a workflow process, the language conveying the process definition, and the mechanism to integrate data/computing resources in a distributed environment.

1.3.1 Workflow Modeling Approach

Most current Grid workflow systems use directed acyclic graphs (DAG) as a modeling language to describe the workflow process in a graphical model. A DAG is a directed graph with no cycles. In a DAG, workflow tasks are represented as nodes. The edges between nodes are the channels that flow data from one task to another. Since there is no directed path starting and ending on the same node in a DAG, the data processed by one task can never return to the same task in the workflow process. A more detailed description for DAGs, including the formal definition, their properties, and features applicable to workflow modeling, is presented in section 2.6.1. Although using a DAG is intuitive for constructing process descriptions and is relatively easy to comprehend by users, a DAG has certain drawbacks. First, a DAG has limits in its modeling power. Since there is no cycle/circuit in a DAG, it is hard to explicitly express loop patterns in a DAG model. Second, a DAG model itself cannot associate with any status information about a process. Grid workflow users cannot monitor the execution of a process with a DAG model. Third and most important, the data of the workflow is implicitly expressed in a DAG. A DAG only models the path of the data transfer but not data itself. No features of the data can be represented and used to control the workflow process. This limitation reduces the usefulness of a DAG to model advanced, data-controlled workflow processes.

In order to overcome these drawbacks, the Petri net model (Peterson, 1977) has gradually been applied in several recent Grid workflow systems. The Petri net is a well-established process modeling technique. Section 2.6.2 defines the Petri net in detail, and offers examples of high-level Petri nets for workflow modeling. Compared with a DAG, a Petri net model has more powerful modeling capabilities. For instance, 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 the equivalent computational power of a Turing Machine (Peterson). According to (Aalst, 1998), Petri nets are more suitably applied than DAGs for workflow modeling for the following reasons.

1. Formal semantics

Classical Petri net and high-level Petri nets (like color (Jensen, 1997), time (Aalst, 1994) and hierarchy (Aalst, 1994) extensions) have been formally semantically defined. Workflow processes defined with a Petri net must have a clear and precise specification, which should facilitate the execution and verification step performed by the WFMS.

2. Graphical nature

Petri nets can act as a graphical modeling language. Workflow members can be easily mapped to Petri net components (Aalst, 1998). For example, tasks are modeled as transitions; data are modeled as tokens; and input/output data sets are modeled as places. Users can use the Petri net not only as a design model, but also as a communication tool of the design ideas to and/or with other users.

3. Expressiveness

By virtue of being formally semantically defined, Petri nets can easily model all routing constructs existing in current workflow management systems (Aalst, 1998). Each primitive used to model a workflow process can find its corresponding mapping in Petri net components. Moreover, the state information explicitly represented in Petri nets enables a WFMS to model checkpoints and implicit choice structure, which are crucial features needed by long-running parallel workflows.

4. Properties

Petri nets have been investigated in both theory and practical application during the last four decades. Many books, articles, and reports have been published that explore each aspect of Petri nets. The firm theoretical foundation in this literature equips people for reasoning about all the basic properties of Petri nets. These properties, consequently, strongly suggest the Petri net as an ideal modeling tool for advanced, complex workflow processes.

5. Analysis

The formal theoretical foundation also enables Petri nets with a number of analysis techniques (Aalst, 1998). These techniques can be used to verify model properties. For example, the verification procedure for a Petri net model (Peterson, 1977) should be able to answer the following questions:

- Is the model safe?
- Is the model sound?
- And, are there any deadlocks existing in the model?

These analysis techniques could also be used to measure the performance of the workflow, such as the response time and the usage rate of the resources (Aalst, 1998). Workflow processes can accrue benefits from those techniques since the properties of workflow processes are modeled as Petri net properties.

6. Vendor independence

The Petri net is a tool-independent modeling technique (Peterson, 1977). It models the workflow process without specifying any specific implementation interfaces (Aalst, 1998). Petri nets are not based on any software package or system architecture. Thus implementers can provide their own design and realization of a Petri net-based WFMS. The Petri net model is also a competitive candidate for the standard method of exchanging workflow processes between and among various workflow systems.

With more systems adopting Petri nets as the modeling approach of workflow process, many research issues have received attention from workflow researchers. These research issues include, but are not limited to:

1. Translating the workflow procedure into Petri net models (Aalst, 1998);
2. Building a platform to design workflow process with Petri net models (Hoheisel, 2004; Zhang, Gu, & Li, 2004);
3. Mapping the data and computing resources onto the Petri net models in order to execute the workflow tasks (Aalst & Hee, 2002).

1.3.2 Workflow Language

Although a graphical model greatly eases the description of a complex workflow process for users, it remains insufficient as a complete process definition by the Grid workflow engine, which is the executor of the workflow process. The reasons for this insufficiency can be summarized in two points. First, compared with its ability to describe control flows, a graphical model lacks the capability to elaborate the "meta-information" about data and computing resources. Such meta-information is crucial for the workflow engine to construct the references to the resources. Second, typical workflow engines are designed to process text-based languages instead of graphical models, since the designers of workflow engines may have limited exposure to date to the knowledge of emerging Model-Integrated Computing (MIC) technology (Akos Ledeczi et al., 2001). Thus, most workflow engines, including Triana (Shields & Taylor, 2004), Taverna (Oinn et al., 2004), and Kepler (Altintas et al., 2004), have architectures similar to a traditional compiler, which are not suitable to directly processing graphical models.

Consequently, a workflow language is designed to bridge the gap between the graphical model and the Grid workflow engine. The major functions of a 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 experts.

Two workflow languages, GSFL (Sriram Krishnan, Wagstrom, & Laszewski, 2002) and BPEL4WS (Tony Andrews et al., 2003), are popular choices in current Grid workflow systems. Both of these languages are capable of describing complex data entities and workflow processes. Each language also adopts the Extensible Markup Language (XML) (*Extensible Markup Language (XML)*) as a persistent storage notation. Their sophisticated description capabilities, however, also imply disadvantages for both languages. They are apparently too complicated to be mastered by casual users of a workflow system. Furthermore, it is difficult, if not really impossible, for an experienced user to describe even a 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 space (Oinn et al., 2004; *The Ptolemy II Project*, 1996; Shields & Taylor, 2004), they are seldom fully implemented for independent third-party reuse.

Compared with these two languages, a light-weight Grid workflow language that suitably addresses major workflow functions would be preferable for expert users. Such a language would have to be powerful enough to support general workflow patterns, as well as sufficiently flexible in order to let the expert users directly write workflow definitions (Guan & Jamil, 2003). This language should also be compatible or translatable to popular workflow languages. Furthermore, this language should be able to fully exploit support from both the graphical user interface and the workflow engine.

1.3.3 Resource Integration

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 computations on the data according to the schedule embedded in the description. Thus, the workflow engine needs to have the power to invoke computational programs and tools, and feed them with corresponding data. Since data and programs are often geographically distributed and heterogeneous, a mechanism to handle the distribution and heterogeneity features should be integrated into the workflow engine.

Most current workflow systems use ad hoc approaches to handle the distributed data and programs in heterogeneous system architectures and operating systems (Altintas et al., 2004; Oinn et al., 2004). For example, in order to feed the data to a program on a remote server, the workflow engine would authenticate to the remote site (log into the remote site with a user name and password), specify the target directory for storing the data, and finally transfer the data with FTP. In even a more complex scenario, in order to invoke a program in the remote server, a workflow engine would need to know more details about the remote site, such as the network connection protocol, the server's operating system, the file system, and the job submission mechanism on the server. There is evidently too much coupling between the remote server and the client-side software.

These ad hoc methods have at least two key deficiencies. First, this kind of method reveals too many remote site details to the workflow engine and end-user. This not only makes the workflow engine more complicated in architecture and functionality to implement, but also weakens the security strength of the whole architecture. Second, most of these ad hoc methods are hard-coded within the workflow engine, making it difficult for the workflow system to accommodate future changes.

Whenever a new resource is added into a WFMS for workflow execution, the workflow engine needs to configure itself for connecting the new resource and submitting jobs. With new techniques emerging and more resources becoming available for executing workflow tasks, a workflow engine needs to make changes frequently. An interface between the workflow engine and the data/computing resources is urgently needed by a WFMS to provide a uniform approach for accessing data and invoking programs.

Grid computing techniques (as described in section 2.3) can satisfy the requirements for building that kind of interface by providing the following: 1) a security infrastructure for user authentication and authorization with remote systems; 2) a data transfer protocol to transfer data between systems with the support of the security infrastructure; and 3) a service-oriented architecture (SOA) for wrapping programs and applications as web or Grid services. There has been a tendency to combine WFMS and Grid computing techniques in the recent development of WFMS.

Several current Grid workflow systems, such as McRunjob (Graham, Evans, & Bertram, 2003) and ScyFlow (McCann, Yarrow, DeVivo, & Mehrotra, 2004), 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 a workable mechanism to integrate their data analysis tools. A Grid workflow system that can integrate various tools in an impromptu manner is consequently preferable, no matter where the tools are physically located (for example, local computer, network environment, Intranet, or even the Internet).

1.4 Hypothesis and Approach

The hypothesis of this dissertation is that it is possible, via a Petri net modeling approach, workflow management technology, Internet computing, and Grid computing, to create a Grid service-based workflow management system that can model workflow processes with Petri nets, and execute workflows over distributed and heterogeneous environments. Such a WFMS would not sacrifice any functionality, feasibility, or generalizability of the workflow system. Furthermore, this system would be an open platform that could integrate new emerging disparate and distributed data and computing resources to provide new capabilities for workflow applications.

To evaluate this hypothesis, a prototype of a Grid workflow system, namely Grid-Flow, has been created to manage the modeling and execution of workflow processes over Grid environments. The Grid-Flow system based on a service-oriented architecture (SOA) shown in Figure 2 illustrates the architecture of Grid-Flow and the interaction between services provided by each component of the system. Grid-Flow is an assembly of functional components that can be roughly classified 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, is designed to facilitate the modeling of workflow processes via a graphical editor, convert the graphical workflow specification into the Grid-Flow Description Language (GFDL), and orchestrate the execution of the workflow cases. The GFDL, whose ultimate goal is to describe and store all kinds of workflow processes, acts as an intermediary connecting the user interface with the Grid-Flow engine to convey workflow specifications. The Grid-Flow engine layer is responsible for interpreting and executing workflow cases by orchestrating the data and program services supported by the data and program integration framework, as well as answering users' monitoring requests. The data and program integration layer of the system infrastructure plays a critical role in interconnecting distributed storage and computing resources together and standardizing accesses to data and programs in the whole system. Each component in this architecture communicates with others through its well-defined interface, which covers implementation details and reveals only necessary functionalities as services. Each component in this architecture is an individual functional unit that consumes services from other components, performs clearly defined functions, and provides services to other components.

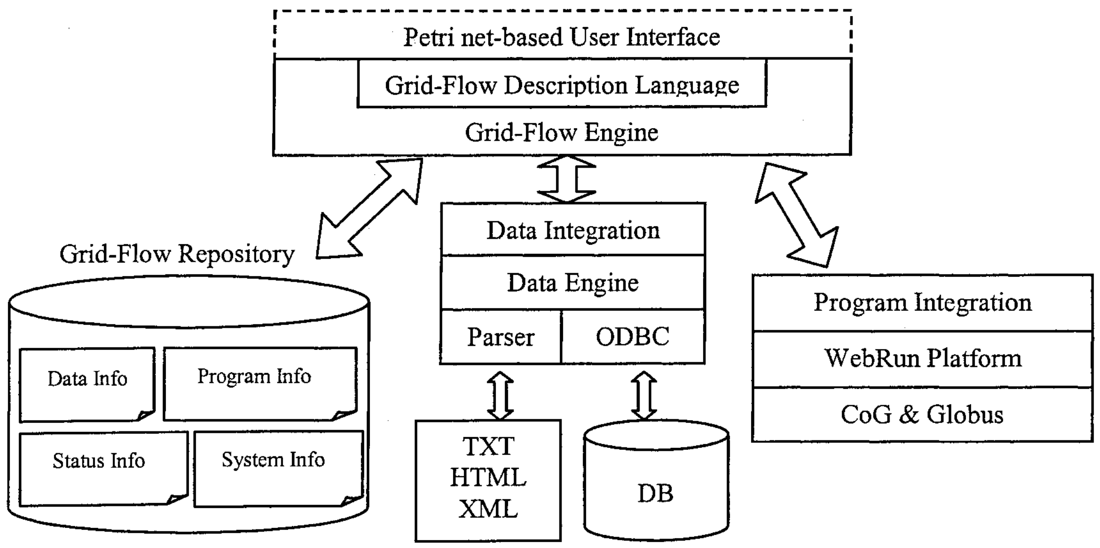


Figure 2. The Grid-Flow System Architecture.

In this dissertation, the Grid-Flow system has been prototyped in order to show the feasibility of realizing a Grid scientific workflow system based on an SOA. The focus here is not to design and run several workflows on Grid-Flow, but to provide a feasible, common framework that facilitates users to model workflow processes and incorporate various data and programs required to execute their workflows. Such a framework should enable the development of a workflow management system suitable for scientific workflows by reducing the complexities involved in accessing and using distributed and heterogeneous resources, by supporting modeling of advanced workflow patterns, and by improving productivity of users by providing the ability of directly defining workflow processes with the workflow language as well as reusing predefined routines.

Grid-Flow exploits established technologies such as Petri net modeling to provide services for modeling workflow controlling structures accurately and formally, and representing data, features, and states of workflow processes explicitly. These services enable Grid-Flow to model complex workflow processes involving advanced patterns, and to monitor the execution of workflow processes by displaying the workflow status to the user. The Petri net modeling approach is realized in the Grid-Flow user interface, which is based on the Generic Modeling Environment (GME) (Akos Ledeczi et al., 2001). A meta-model for the Petri net is designed and stored in GME. Thereby, users can define domain models (that is, workflow process models) with the same environment. The meta-model and domain models for Petri net are demonstrated in detail in section 2.6.3. In addition, Grid-Flow introduces an intuitive approach, called the Data/Program Chart, to help users communicate with workflow designers by drawing workflow processes under minimal restrictions.

GFDL is used in Grid-Flow to record any information related to defining and executing workflows. This information can be classified into two categories. Information about data and programs, such as the format of the data or the location of the program, is recorded by data and program registration scripts. This type of information is stored in the Grid-Flow repository and would be used to integrate data and programs while workflow is running. The other type of information is used to describe the flow control of workflow processes. This information is recorded by process description sentences and stored in GFDL scripts. When a user defines a workflow using the user interface, Grid-Flow asks the user to register all the data and programs used in the workflow into the system with system-provided registration wizards. Grid-Flow then translates the flow control information into GFDL scripts with the Petri net model interpreter. After receiving a user's command to run the workflow, the Grid-Flow engine retrieves the flow control description from GFDL scripts, integrates data and programs by reading their associated information, and then schedules and performs the execution of programs on appropriate data.

The data/program integration framework can actually be divided into two components: the data integration component and the program integration component. Data used in scientific workflows come from different resources in different formats. The major function of the data integration component is to retrieve data from different data sources, and transform data among various formats if necessary. Parsers corresponding to relevant data formats are developed to retrieve information for the data integration component. For example, the XML parser is responsible for extracting information from an XML file. A Structured Query Language (SQL) parser is used to query a relational database to retrieve records. Data transformers that transform data between two different formats are designed and implemented in a more ad hoc way, compared to parsers. For instance, the transformer that converts integers into character strings is implemented as an internal program directly supported by the Grid-Flow engine.

As with data, programs can also be located in various environments and have special execution requirements and mechanisms. The program integration component addresses all of these heterogeneities by providing a uniform program invocation service to the Grid-Flow engine. The program integration component handles different type of programs in different ways. For programs on a local machine, the program integration component invokes them by calling system functions. For Common Gateway Interface (CGI) programs, the program integration component provides a mechanism to register and invoke them. For programs that are not accessible on a remote server, the program integration component employs WebRun (Guan, Liu, Velusamy, & Bangalore, 2004) to wrap them as web/Grid services. A data matching mechanism is also discussed in section 6.3 to pipeline programs that works by mapping outputs of one program to inputs of another.

To constructively prove the hypothesis, Grid-Flow is first prototyped based on a multi-tier architecture that integrates several functional subsystems; then two bioinformatics workflow cases, protein transmembrane region analysis and protein functionality and expression, are modeled using this enabling workflow management system. The two workflow processes depicted using Petri net-based user interface illustrate that Petri nets can be used to model advanced workflow structures. In addition, the fact that one workflow model can be reused in another workflow illustrates the ability to build hierarchical workflow models in Grid-Flow. The GFDL scripts used to convey workflow definitions demonstrate the usability and efficiency of the light-weighted workflow language to describe control flow structures and the data flow. Several programs located in distributed and heterogeneous environments are integrated into Grid-Flow as workflow tasks. The effectiveness and utility of the data and program integration component are demonstrated by integrating those data and programs used in these two workflow cases.

1.5 Contributions

In addition to prototyping Grid-Flow for developing advanced scientific workflow applications over Grid environments, this dissertation also makes the following contributions to the research of scientific workflow management systems:

1. Grid-Flow provides a systematic methodology, as well as a flexible test bed, for designing distributed scientific workflows over heterogeneous data and computing resources, based on model-driven computation.
2. A Petri net-based user interface is demonstrated to help users design workflow processes with Petri net models. An interpreter is implemented in GME (Akos Ledeczi et al.) to translate Petri net workflow models into GFDL.
3. A process modeling approach, the Data/Program Chart, is introduced to help users to express workflow process specifications and communicate with other users or workflow designers.
4. A light-weight, programmable Grid workflow language is provided to convey workflow processes in Grid-Flow. This language is powerful enough to describe all types of workflow patterns, and is easily mastered by advanced users.
5. A data integration component is implemented to retrieve data from various data resources, and transform data among various formats if necessary.
6. A program integration component is presented to integrate applications into workflow processes as tasks. This component can 'wrap' programs on remote servers as atomic executable services.
7. A methodology for registering online data and CGI programs is defined for porting them to serve the execution of workflows.
8. An intuitive method for matching data between output and input sets of connected programs is provided to streamline the execution of workflow tasks.
9. Two real-world workflows have been built to evaluate the feasibility, usability, capability, and generalizability of the workflow platform.

1.6 Organization

This dissertation describes the design and prototyping of a Grid workflow system using Petri net modeling and Grid enabled service-oriented architecture. The remaining chapters are organized as follows. Chapter 2 briefly reviews concepts of workflow management, the history, and current status of scientific workflow systems, as well as technologies used in prototyping Grid-Flow, including Grid computing technologies, modeling approaches, and tools. Chapter 3 illustrates the overall architecture of Grid-Flow and the decisions made during its design and implementation. Research issues of the Grid workflow management system, such as the workflow description language, the modeling approaches, and the integration of online data and programs, are discussed in Chapter 4, 5, 6, respectively, with the practical implementation in Grid-Flow. Chapter 7 presents the Grid service used in Grid-Flow for integrating analysis tools, and one of its applications -- G-BLAST services. Two illustrative examples of running workflows with Grid-Flow are reported in Chapter 8 in terms of their user requirements, workflow designs, execution details, and the results. Chapter 9 evaluates the whole Grid-Flow system and techniques used in each of its subsystems, as well as provides the summary and conclusion of Grid-Flow. Chapter 10 discusses the future research and development work of Grid-Flow. The syntax of the Grid-Flow Description Language (GFDL) is presented in Appendix A, while the strategies of wrapping existing applications as web/Grid services in WebRun system are fully explored in Appendix B.

2 LITERATURE SURVEY

Chapter 1 described the motivations and requirements for building a Grid workflow system to serve scientific computing applications. In this chapter, workflow management systems, Grid computing, and Petri net modeling are discussed in further detail. Other efforts in the literature that build Grid or scientific workflow systems are surveyed. The origin of Grid-Flow is presented at the end of this chapter for the purpose of recording its design and development history.

2.1 Nomenclature of Workflow Management

A workflow is a combination of cases, resources, tasks, and states that relate to a specific application. According to the definition provided by Wil van der Aalst in (Aalst & Hee, 2002), workflow management refers to "the ideas, methods, techniques, and software used to support structured business process." The workflow management system, therefore, is a software package that supports the design, configuration, and implementation of generic workflow applications. The workflow management system includes the following principal concepts.

- **The case.** The "case" is a workflow instance that is controlled by the workflow management system. Examples of a case could include an analysis of a protein structure, an auto insurance claim, or a transaction completed on the Internet.
- **The task.** A "task" is an atomic logical unit of work. A task is either carried out in full or not at all in a workflow instance. Tasks are important components for a work-

flow process. When a special case is performed in the workflow system, a task is referred to as an activity once it is carried out.

- **The process.** The “process” indicates the way in which a particular kind of case should be carried out. It involves not only which tasks need to be carried out, but also the order in which those tasks should be done. A process consists of tasks, conditions, and subprocesses that are scheduled with a predefined execution plan. Subprocesses enable the potential for hierarchical structuring of complex processes.
- **Routing.** “Routing” is the definition of a process that determines which tasks need to be performed and in which order along particular branches of a process. Four kinds of routings are often used in workflow process: sequencing, selection, parallelization, and iteration.
- **Enactment.** Workflow enactment services is a component that “creates new cases, generates work items based upon the process description, matches resources and work items, supports the performance of activities, and enables the recording of particular aspects of the workflow” (Aalst & Hee, 2002).

Accompanying the development of WFMS, workflow system architecture has evolved from supporting simple workgroup type environments to providing enterprise-wide, and even multi-owner levels of functionality (Mohan, 1998). A single workflow process is now capable of being authorized to integrate servers and clients across wide area networks with the enhancement in WFMS. Such workflows with additional scalability, availability, and manageability are called *distributed workflows*, which are the predecessors of current Grid workflows.

2.2 Scientific Workflow System

Scientific workflow systems have been developed for at least twenty years. The technology is maturing with more development of workflow products providing support to various kinds of applications (Aalst & Hee, 2002; Dogac, Leonid, Ozsu, & Sheth, 1998). However, many additional limitations still remain in the field of modeling processes. Many researchers work still on developing more efficient workflow languages. They try to simplify the workflow language while keeping its power for describing diverse workflow models. Vortex workflows are provided to solve this problem by using enabling conditions, attribute rules, and declarative semantics (Davulcu, Kifer, Ramakrishnan, & Ramakrishnan, 1998; Hull et al., 1999). On the other hand, object-oriented workflow languages are also provided to support flexible and distributed workflows in heterogeneous environments (Vossen & Weske, 1999; Weske, 1999). Another trend in workflow research is the cooperation and interaction between workflow processes from different sources. Studies have been reported on workflow models and systems that enable cooperation between workflows executed in the same or in different organizations (for example, (Ames, Burleigh, & Mitchell, 1997; Casati & Discenza, 2000; Kang, Park, & Froscher, 2001)). These systems extend the scope of workflow models from local to broader applications.

Refined definition language and cross-organization cooperation comprise the essential requirements of workflow applications in the bioinformatics field, one of the most important application areas of current scientific workflow systems. Because of the large scale of workload, many genomic projects require the cooperation among crossing groups, organizations, or even nations. The application of a workflow system for bioinformatics is flourishing with the dramatic development of genomic technologies. A batch of small commercial workflow systems, such as the Functional Bioinformatics System (*Functional Bioinformatics System*), TurboBench Workflow (*TurboBench Workflow System*), and the überTOOL (*uberTOOL*), are coming forth to integrate certain biological tools that are used over multiple network platforms. The TurboBench workflow system can even let the user design parallel, distributed, and computational grid utilities on particular tasks. However, none of these commercial workflow systems constitute general-purpose workflow systems. They work well only on particular, predefined biological fields of study. For example, the Functional Bioinformatics System only focuses on GSE sequencing and access to sequencing-related information.

Many academic workflow systems are general-purpose and can be applied to a wide range of areas in bioinformatics. For example, the workflow system used at the Whitehead Institute/ MIT Center for Genome Research (Stein et al., 1995) focuses on managing workflow in large semi-automated laboratory projects. The METEOR project (Kochut et al., 2002), which has been developed at the LSDIS lab of the computer science department at the University of Georgia, is also a general-purpose system that focuses on R&D of innovative Multiparadigm Transactional Workflow Management technology. These systems have each a powerful workflow definition language that can describe almost all kinds of complex models. But for any projects in those systems, the design of workflow must be handled by professionals with expertise in that workflow area. It is difficult for biologists to master such a complex workflow language and design their own workflow processes.

2.3 Grid Computing

Researchers in distributed computing have proposed many schemes for connecting distributed resources shared by multiple organizations. Among these schemes, the proposal of a "computational grid" (Foster & Kesselman, 1999) gradually proved itself to be an infrastructure that can enable ubiquitous computing over heterogeneous high performance computing resources distributed geographically. In the last decade, more scientific research projects have started to support the computational grid and thus developed several tools and services, which are known as Grid technologies(Berman et al., 2003). With the development of these Grid technologies, a new computing model, known as Grid computing (Berman et al., 2003), emerged. The goal of Grid computing is to gain more computing power over shared heterogeneous resources that locate in diverse and possibly distant places, belong to multiple administrative domains over a network, through the interoperability abled by open standards (Foster, Kesselman, Nick, & Tuecke, 2002). Grid computing aims to achieve the following goals:

- Virtualizing computing resources by sharing distributed and heterogeneous computing resources belonging to different organizations accessible from anywhere (Berman et al., 2003);
- Providing secure access to these resources while protecting securities for both users and remote sites (Foster, 2002);
- Providing single login service to all users over all distributed resources while keeping the individual site autonomy system impregnable (Foster, 2002);
- Covering the heterogeneity and the complexity of interoperation among distributed sites from end-users (Berman et al., 2003);

- Providing resource management, information services, transparent job submission, monitoring, and secure data transportation (Foster & Kesselman, 1999); and
- Designing to solve problems too big for any single supercomputer (Berman et al., 2003).

The Global Grid Forum (GGF) (*Global Grid Forum*), a collaboration between industry and academia, has the purpose of defining specifications for Grid computing. The specifications cover a variety of issues from security and authorization to resource description, discovery, and reservation, to monitoring and event notification, and many more. Many vendors (for example, IBM, Oracle Corp., Sun Microsystems, and Cluster Resources, Inc.,) have started to support Grid computing with their hardware and software products. Some academic computing Grid prototypes have already been deployed. The Grid Physics Network (GriPhyN) (*GriPhyN - Grid Physics Network*), Earth System Grid (ESG) (*Earth System Grid*), and TeraGrid (*TeraGrid*) are several examples.

2.3.1 Globus Toolkit

The Globus toolkit (Foster & Kesselman, 1999) is the *de facto* standard for Grid middleware. It is and was implemented by the Globus Alliance (*The Globus Alliance*) following the standards developed at the GGF. As a middleware ensemble, it provides a standard platform upon which to build services. This platform also provides software tools and services that enable the development of computational grids in many areas defined by the GGF as protocols (Foster & Kesselman, 1999). These software tools realize the following functional modules:

- Grid Security Infrastructure (Foster, Kesselman, Tsudik, & Tuecke, 1998), used by the Globus Toolkit for authentication, authorization, and secure communication;

- Monitoring and Discovery Services (Foster & Kesselman, 1999), used to store and access various system specific information, as well as to discover, publish and access static and dynamic information about the status of resources in the computational grid;
- Globus Resource Allocation Manager (Berman et al., 2003), used to allocate, manage, and request resources on the computational grid using a resource specification language;
- Data Movement and Management Services (Foster & Kesselman, 1999), used to access, manage, transfer (using GridFTP (Allcock et al.)), and replicate data over the computational grid;
- Commodity Grid Kits (Laszewski, Foster, Gawor, & Lane, 2001), which provide client-side APIs for Grid services through language bindings and implementations in multiple popular programming languages; and
- MyProxy (Novotny, Tuecke, & Welch, 2001), which provides an online credential repository that can store and retrieve proxy credentials for users without accessing their permanent certificates.

2.3.2 Open Grid Service Architecture

The Open Grid Service Architecture (OGSA) (Foster et al., 2002) is a specification of the SOA for the Grid computing environment for business and scientific use, as developed by GGF. It provides an interface based on web services for managing Grid service instances. The interfaces and behaviors of a Grid service are defined as OGSA specification, and are implemented by Open Grid Service Infrastructure (OGSI) (Tuecke et al.), which is as of now superseded by the Web Service Resource Framework (WSRF) and WS-Management.

2.4 Grid Workflow System

Although Grid workflow systems lack a long history when compared to other technologies in the Grid computing community (Zhang et al., 2004), there is 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 (Bhatia et al., 1997) and the Common Component Architecture (CCA) (*Common Component Architecture*). 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 Web/Java 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 (*Common Component Architecture*). This strategy of component composition has been inherited by many current Grid workflow systems, such as GridAnt (Laszewski et al., 2004), Triana (Shields & Taylor, 2004), Symphony (Lorch & Kafura, 2002), XCAT (S. Krishnan, Bramley, Gannon, Govindaraju, Alameda et al., 2001), GridFlow (Cao, Jarvis, Saini, & Nudd, 2003), and Ptolemy II (*The Ptolemy II Project*, 1996). The strategy has been extended to applications of composition of software modules and web/Grid services (Cao et al., 2003; S. Krishnan, Bramley, Gannon, Govindaraju, Alameda et al., 2001; Laszewski et al., 2004; Lorch & Kafura, 2002; *The Ptolemy II Project*, 1996; Shields & Taylor, 2004).

Almost all Grid workflow systems mentioned above have the common characteristic that they are derived from some existing flow control tools or systems, and are applied to orchestrate Grid-enabled programs or services. For example, GridAnt (Laszewski et al., 2004) was developed from Ant (*Ant - A Java-based Build Tool*), while XCAT was based on the Common Component Architecture (Laszewski et al., 2004). 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 (Dozsa, Kacsuk, Lovas, Podhorszki, & Drotos, 2004), ScyFlow (McCann et al., 2004), McRunjob (Graham et al., 2003), Taverna (Oinn et al., 2004), GriPhyN (Deelman, Blythe, Gil, & Kesselman, 2003), and Kepler (Altintas et al., 2004). From these descriptions (Altintas et al., 2004; Deelman et al., 2003; Dozsa et al., 2004; Graham et al., 2003; McCann et al., 2004; Oinn et al., 2004), it is evident that these Grid workflow systems intend to provide interoperability among various applications across heterogeneous computational platforms. Since applications have more diverse interfaces than components/services, these program-based workflow systems are designed to be 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 systems and process planning/scheduling systems could also provide workflow control functionality in a Grid environment. UNICORE (Erwin & Snelling, 2001) and the Directed Acyclic Graph Manager (DAGMan) (*DAGMan (Directed Acyclic Graph Manager)*, 2002) are two good examples.

2.5 Grid Workflow Language

Besides 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) (Leymann, 2001), XLANG (Thatte, 2001), the Web Services Conversation Language (WSCL) (*Web Services Conversation Language (WSCL 1.0)*, 2002), and the Web Services Choreography Interface (WSCI) (Arkin et al., 2002). With the development of web services, WSFL and XLANG converged and this led to a new generation of specification language for business interaction, the Business Process Execution Language for Web Services (BPEL4WS) (Tony Andrews et al., 2003). BPEL4WS (Tony Andrews et al., 2003) expands the Web Service interaction model and makes it applicable to depict business transactions. With the emergence of Grid services, the Grid Services Flow Language (GSFL) (Sriram Krishnan et al., 2002) was proposed to address the issue of integrating Grid services across distributed and heterogeneous platforms within the OGSA (Foster et al., 2002). Both BPEL4WS and GSFL have adopted XML (*Extensible Markup Language (XML)*) as their basic language format (Tony Andrews et al., 2003; Sriram Krishnan et al., 2002). Most Grid workflow systems also choose an XML-based language as their workflow process language. For instance, Triana (Shields & Taylor, 2004) uses an XML-based language similar to the Web Services Description Language (WSDL) (Christensen, Curbera, Meredith, & Weerawarana, 2001). At the same time, Triana can use any language that is compatible with BPEL4WS (Shields & Taylor, 2004). Since no standards have been agreed upon for Grid services, the research area of Grid workflow languages is still undergoing rapid change and further development. For example, a relatively new

Grid Workflow Execution Language (GWEL) (Cybok, 2004) has been proposed to reuse ideas from BPEL4WS on describing interactions between services defined with WSDL.

2.6 Process Modeling

The majority of Grid workflow systems use the Directed Acyclic Graphs (DAG) and its variants as their approach for workflow process modeling (Altintas et al., 2004; Erwin & Snelling, 2001; Lorch & Kafura, 2002; Shields & Taylor, 2004). For example, UNICORE (Erwin & Snelling, 2001) and Symphony (Lorch & Kafura, 2002) use DAGs to describe the workflow process, while Kepler (Altintas et al., 2004) and Triana (Shields & Taylor, 2004) use Directed Cyclic Graphs (DCG) as their modeling methods. DAGs, as well as DCGs, have limited modeling abilities. Since a DAG has no cycles in its model, it cannot be applied to express loops explicitly. Because of this drawback, the Triana system (Shields & Taylor, 2004) does not explicitly support iterative control constructs. All of the loop controls are handled by a specific loop component in Triana. A more powerful modeling approach, Petri nets (Peterson, 1977), is gradually being introduced into Grid workflow systems to model workflow process (Hoheisel, 2004; Zhang et al., 2004). A Petri net modeling method is applied in (Hoheisel, 2004) to describe user tools and workflow schemes developed in the Fraunhofer Resource Grid (FhRG) (*Fraunhofer Resource Grid*). An extension of the Petri net (Peterson, 1977), D-Petri net, is also employed in (Zhang et al., 2004) to capture the characteristics of dynamism of Grid job scheduling. As mentioned in Chapter 1, Petri nets are central to the work of this dissertation as well.

2.6.1 Directed Acyclic Graph

A DAG is a directed graph with no directed cycles (Harary, 1995). That means for any node *v* in the graph, there is no directed path starting and ending on *v*. A source node in a DAG is a node with no incoming edges, while a sink node in a DAG is a node with no outgoing edges (Wikipedia). Figure 3 shows an example of a DAG. The rectangles with names represent the nodes. Arrows in the graph represent the directed edges of DAG. In this example, node A does not have any incoming edges, it is a source. Similarly, node E is a sink since there is no outgoing edges associate with it.

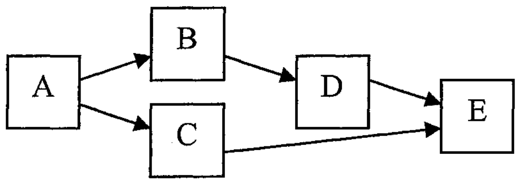


Figure 3. An Example of Directed Acyclic Graph.

A DAG has two features that make it popular for modeling simple workflows (Hoheisel, 2004). First, a finite DAG has at least one source and at least one sink (Wikipedia). Corresponding to workflow models, each workflow model usually has at least one starting task, which does not have any precedent tasks, and at least one ending task, which has no following tasks. So a source node can be used to model a starting task, and a sink as an ending task. Second, each DAG has a topological sort, which is an ordering of the nodes such that each node comes before all nodes it has edges to (Wikipedia). In general, this ordering is not unique. For example, one possible topological sort for Figure 3 is A-C-B-D-E. However, if a workflow process is modeled with DAG, this ordering can help workflow systems determine the execution order of each task before they are executed in the real world. So the whole workflow process would be executed following a static order once the topological sort is selected. This simplifies the design and implementation of a workflow management system, especially for a workflow engine.

2.6.2 Petri Nets

This section introduces the definition of classical Petri nets (Peterson, 1977) and the special features of high-level Petri nets.

A classical Petri net is a directed bipartite graph with two node types: places and transitions (Peterson, 1977). Arcs are used to connect nodes, but arcs between two same type nodes are prohibited. The formal definition of Petri net is as follows (Wikipedia):

A Petri net is a triple (P, T, F), where

1. P is a finite set of places,
2. T is a finite set of transitions $(P \cap T = \varnothing)$
3. $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relations)

Usually places are represented by circles, and transitions as bars (Peterson, 1977). At anytime a place contains zero or more tokens, drawn as black dots. States, often referred to as markings, are the distribution of tokens over places. The number of tokens may change during the execution of the net. Transitions change the state of the net according to firing rules (assuming the weight of each input place is 1) (Peterson, 1977):

A) A transition is said to be enabled if and only if each input place of this transition contains at least one token;

B) An enabled transition could fire. As the result of firing a transition, one token from each input place of this transition will be consumed and each output place of this transition will be filled with one token.

Petri nets can be used to model workflow processes (Aalst, 1998). Each concept in a Petri net has a corresponding mapping in a workflow process. The transition in a Petri net is mapped to the task in a workflow process. The token represents the data, while the place corresponds to the container of data. Each arc shows the data transfer path in the workflow process. The state or marking can be naturally explained as the status of the workflow process. Thus, the corresponding firing rules in workflow process should be (Aalst & Hee, 2002):

A) A task is said to be enabled if and only if each data it requires is loaded into the data container for each input port;

B) An enabled task can be executed. As the result of execution, the data provided by the input port will be consumed and the task will generate the data for each output port.

Although the classical Petri net is powerful in modeling tasks, data, status, events, and all of the control flow structures, it has its own drawbacks (Aalst & Hee, 2002). First, the Petri nets used to model complex workflow processes tend to be extremely large. Second, the classical Petri net cannot model data attributes and time, which may be crucial to some workflow processes. To solve these problems, many extensions have been proposed to enhance the classical Petri net model (Peterson, 1977). A well-known extension called "extension with color" (Jensen, 1997) focuses on recording the attributes of data as colors associated with each token. The transition generates tokens with various colors, just like the task generates data with different attribute values. The transition can select tokens with appropriate colors by specifying a precondition to filter unsatisfied tokens. This mimics the strategy that tasks only take data with appropriate attributes as inputs. To model time duration and delays of workflow processes, classical Petri net needs to bring in a timing concept, which could associate with tokens, places, and/or transitions. To avoid large Petri net models when describing complex processes, an extension with hierarchy could be added into the classical Petri net to allow it to construct subnets (Peterson, 1977). Thus, the Petri net model can be organized within a subnet hierarchy.

2.6.3 Generic Modeling Environment

From a modeling perspective, the expressive power in software specification is often gained from using notations and abstractions 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 (Neema, Bapty, Gray, & Gokhale, 2002). 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 (Akos Ledeczi et al., 2001). 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 (for example, embedded systems) (Sztipanovits, 2002). In such systems, MIC has been shown to be a powerful tool for providing adaptability in evolving environments (Karsai, Maroti, Ledeczi, Gray, & Sztipanovits, 2004). The GME (Akos Ledeczi et al., 2001) is a domain-specific modeling tool that realizes the principles of MIC. 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) (Neema et al., 2002) 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 (Neema et al., 2002). 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) (Neema et al., 2002).

As an example of domain-specific modeling, Figure 4 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 (Booch, Rumbaugh, & Jacobson, 1998) and OCL (Warmer & Kleppe, 2003) (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 Transmembrane Region Analysis workflow (described in section 3.5) as specified in the Petri net modeling language.

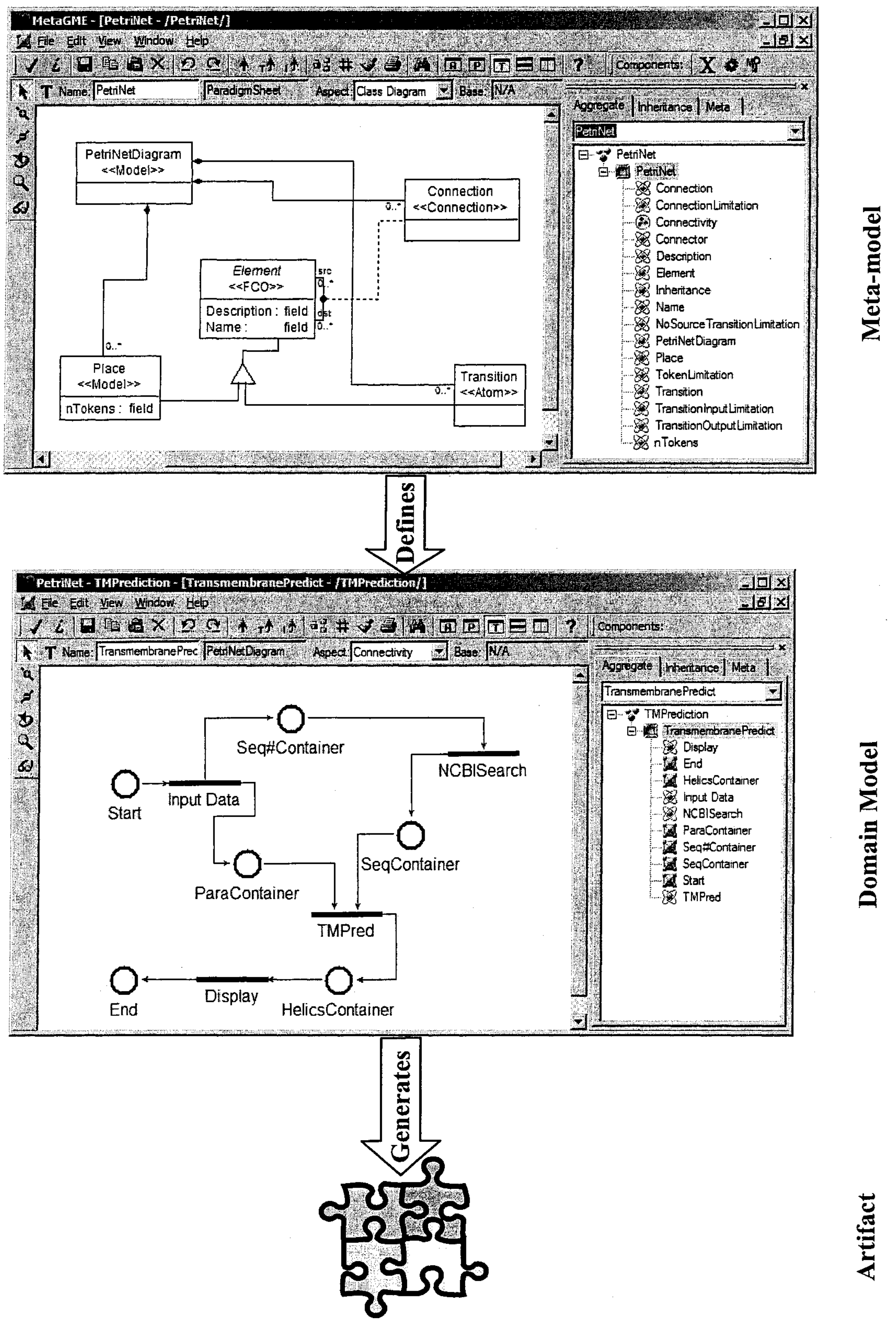


Figure 4. A Petri Net Domain in GME.

GME permits model interpreters to be associated with specific domains as tool plug-ins (Akos Ledeczi et al., 2001). 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 (*High-Level Java Interface to GME -- Users Manual Version 1.0*, 2004). The bottom of Figure 4 symbolically represents a generated artifact from the interpretation of the workflow model; this artifact could be source code, an XML representation of the model, or some other translation. A model interpreter is like compiler in that it generates machine code from a domain-specific model. Although the example in Figure 4 was chosen for simplicity, GME has been used to create rich modeling environments containing thousands of modeling components (Ledeczi, Davis, Neema, & Agrawal, 2003).

2.7 The Origin of Grid-Flow

The BioFlow system (Guan & Jamil, 2003) is the previous work of the author in cooperation with others. Its initiation dates back to the early stages 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. Using the Hyper Text Query Language (HTQL) (Chen & Jamil, 2003), BioFlow can invoke Internet programs, query information from web pages, as well as shield users from the distributed and heterogeneous details of the online resources. In addition, the BioFlow (Guan & Jamil, 2003) 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 complex workflow patterns that arise from real-world applications (Guan & Jamil, 2003).
2. BioFlow directly accesses web pages for submitting jobs to the Internet programs, rather than employing web or Grid services.
3. BioFlow is not capable of tailoring itself in response to the rapid change of the Internet programs (Guan & Jamil, 2003). Thus, the quality of the target Internet programs cannot be guaranteed.

On the other hand, the architecture and the BioFlow language (Guan & Jamil, 2003) 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 provides an interface designed to be user-friendly and based on the Petri net modeling approach; and
2. The Grid-Flow system adapts the program integration component to support web and Grid services. The program integration component is based on WebRun (Guan et al., 2004), which is a unified platform supporting Grid computing (Berman et al., 2003) environments.

2.8 Summary

This chapter focused on introducing two parts of key knowledge to build the Grid-Flow system. One part is the basic concepts for the Grid-enabled workflow system, such as concepts of the workflow management system, Grid computing, and Petri net modeling. The other part is the surveys of scientific workflow systems, Grid workflow systems, Grid workflow languages, and mechanisms to model processes. Starting with the next chapter, the design and implementation details of the Grid-Flow system are presented based on the knowledge discussed in this chapter.

3 ARCHITECTURE AND RATIONALE FOR GRID-FLOW

This section introduces the design and implementation approaches of 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, the grid-enabled data and program integration subsystem, and an illustrative workflow example that will be used in the following chapters to demonstrate the salient features of each aspect of the Grid-Flow system.

3.1 System Architecture

The Grid-Flow architecture (as shown in Figure 2) is adapted and extended based on the BioFlow system architecture presented in (Guan & Jamil, 2003). This architecture can be categorized into three layers: the Petri net-based user interface (please refer to Chapter 5 for details), 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. It can translate the workflow specification into the Grid-Flow Description Language (GFDL) (see Chapter 5 for details), 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 process 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 Internet program integration and WebRun system (Guan et al., 2004), a unified platform that invokes remote programs via Grid computing technology.

3.2 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 each of the programs. This checking 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 (that is, a 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, once all preconditions are met. The approach of reading and writing files acts as the inter-program communication mechanism since it is easy to implement and a practical method for checking the data dependency. Advanced models of pipelining data between programs in Grid environments, including data streaming (McCune et al., 2004; Wu & Sussman, 2004) will be applied to the Grid-Flow system with improvement over time enabled by ubiquitous Grid file systems.

The Grid-Flow engine controls the scheduling and monitoring of the workflow process execution. The implementation of the Grid-Flow engine is derived from the previous "BioFlow" system (Guan & Jamil, 2003).

The detailed inner structure of the Grid-Flow engine is depicted in Figure 5. The engine has the following five components, mirroring the BioFlow engine structure (Guan & Jamil, 2003):

1. The *Syntax Analyzer* parses and compiles the workflow definition conveyed by GFDL to generate executable tasks;
2. The *Repository Interface* maintains data and program information by interacting with the Grid-Flow repository;
3. The *Data Manager* maintains application data through the Data Integration component;
4. The *Program Manager* invokes the program tasks through the Program Integration module, while coordinating the acquisition and organization of local and remote site responses;
5. The *Administrative Module* is responsible for the marshalling the user applications, execution, management of the program flow, and task allocation and the coordination for the abovementioned components. A Grid resource broker and scheduler (Afgan, Velusamy, & Bangalore) could be employed by this module to map the programs reg-

istered in the Grid-Flow repository to the real-world Grid services, and schedule the invocation of those Grid services via the Program Manager component.

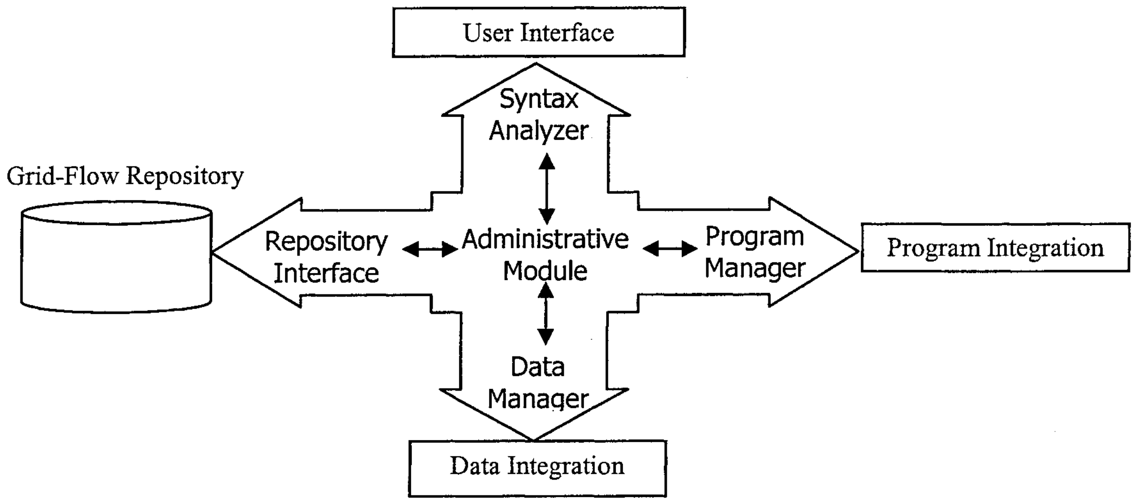


Figure 5. Architecture of Grid-Flow Engine (*cf.* (Guan & Jamil, 2003)).

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.

3.3 Grid-Flow Repository

This component assists in storing the information of registered data and program, process variables, and numerous system metadata necessary for application execution. It also maps all data and program registration information into system tables for use by the

Grid-Flow engine at run-time. The Grid-Flow data repository is implemented on a relational database management system that supports the Open Database Connectivity (ODBC) (Cannan & Otten, 1992) and Structured Query Language (SQL) (Cannan & Otten, 1992). Four primary system tables are maintained (as in the BioFlow system (Guan & Jamil, 2003)):

- The *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.
- The *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 identification (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.
- The *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 may also be stored in the registered data table. The Grid-Flow engine is charged with job cleanup of this table after the execution of each workflow process.
- The *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 the 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.

3.4 Integrating Data and Computing Resources

This section introduces the data and program integration components of the Grid-Flow system.

3.4.1 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 concern for 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 acquiring the required registration information, the Grid-Flow engine provides the Data Integration component with that registration information and its desired data format. Data Integration then retrieves the data with its own data engine, and subsequently transforms that data into the desired format with the HTQL (Chen & Jamil, 2003). 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 parser. For data in plain text, HTML, or XML formats, 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) (*The GridFTP Protocol and Software*) to transfer data files when needed. After retrieving data, Data Integration needs to transform data into the destination format. The transformation approaches are discussed in more detail in section 6.4. Finally, the Data Integration component returns the data to the Data engine and accomplishes its data access service.

3.4.2 Program Integration

The Grid-Flow engine treats every Grid-Flow task as an executable program that 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 an Internet program accessible only with CGI in web pages, or even a web/Grid service that hides the software and hardware details of the implementation. Therefore, a unified platform able to cover over diverse environmental details from the users is what is to be desired in order to manage various programs on distributed and heterogeneous computing resources. The program integration component in the Grid-Flow system is just such a unified platform. It can shield the variety of details and handle different types of programs in different ways. For programs on the local machine or direct-accessible environments (such as clusters that users can remotely log in), the program integration component invokes them with system calls or the Secure Shell (SSH) command executor. For CGI programs on the Internet, the program integration component provides a systematic mechanism to register and invoke them, and transfer and transform data for them as input and output. A detailed description of methodology for CGI program integration is presented in Chapter 6. For programs in remote systems that are not directly accessible, WebRun (Guan et al., 2004) is employed as a reference model for wrapping those programs as web/Grid services. WebRun also provides different deployment strategies for different types of programs. Appendix B describes the WebRun system in more detail. In summary, the program integration component provides the Grid-Flow system with an integral service that can handle most of the non-interactive programs.

3.5 An Illustrative Example

The workflow described in this section will be used as an illustrative example in the following chapters to demonstrate the methodologies and technologies that can facilitate users to design and execute scientific workflows with Grid-Flow. Section 8.1 presents the implementation and the execution of this workflow as a whole.

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 (Lawerence, Banes, & Azadi, 2003). The method of prediction (Lawerence et al., 2003) essentially relies on several aspects of related information, such as the functionality of similar protein sequences, the transmembrane regions of the given sequence, and the promoter regions on the corresponding DNA sequence. The transmembrane 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 transmembrane helices is likely an integral membrane protein that functions in transport or polymerization of molecules, while proteins with no transmembrane helices are likely cytosolic and function in synthesis of molecular building blocks (Lawerence et al., 2003). An overview of the process design on prediction of transmembrane regions is illustrated in Figure 6. 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 that process the data.

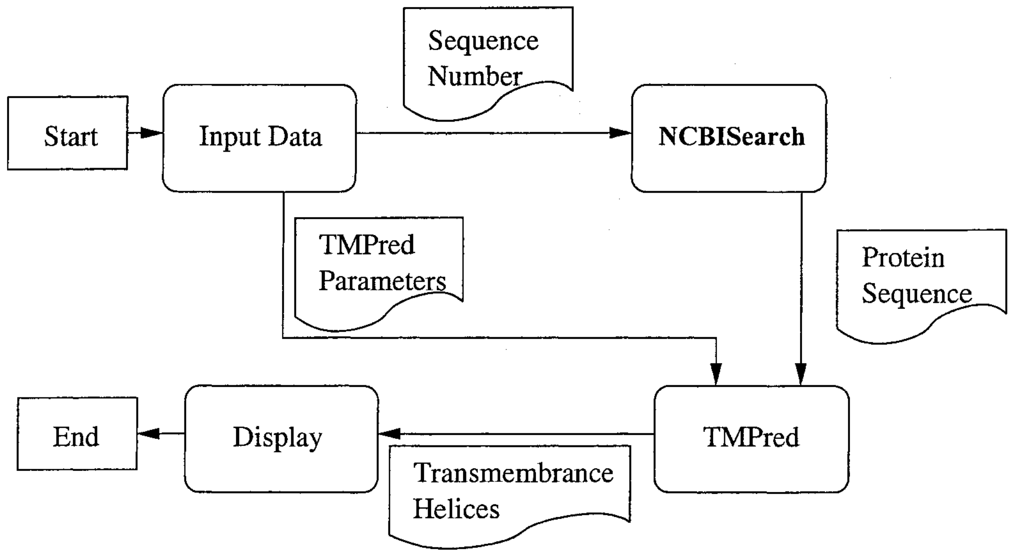


Figure 6. Process of Transmembrane Regions Analysis.

The process operates in the following way. When a biologist plans to analyze the transmembrane regions of the given protein sequence, he or she initializes an instance of the workflow model for predicting transmembrane regions. This process first asks the biologist to provide the protein sequence accession number and decide the analysis parameters of the *TMPred* task (*TMpred - Prediction of Transmembrane Regions and Orientation*), which is a program that can predict the transmembrane 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 *NCBISearch* and search the protein sequence in the protein database located on the same resource of the National Center for Biotechnology Information (NCBI) website (*NCBI Website*). The program *NCBISearch* is an Internet CGI program that compares a protein sequence against a large protein database distributed on a cluster of computers. After using NCBISearch, 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 *NCBISearch*; the other is the data *TMPred Parameters* that served as input in the first task. The *TMPred* task is responsible for predicting the transmembrane regions according to the input protein sequence and parameters. Once the *TMPred* task is executed, it predicts the possible Transmembrane 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.”

3.6 Summary

The architecture of the Grid-Flow system has been presented as a whole in this chapter based on its three-layer architecture. The following chapters discuss each sub-components of the Grid-Flow system from differing aspects. Among the three layers, the Grid-Flow engine layer is the only layer that has fully been investigated in this chapter including its inner structure. The other two layers are studied in further detail in Chapter 5 (graphical user interface), Chapter 6 (data integration and matching), and Chapter 7 (program integration), respectively. The "glue" between the graphical user interface layer and the Grid-Flow engine layer, that is, the Grid-Flow Description Language (GFDL), is examined in the next chapter as the first study of a subcomponent. The subcomponents and their relationships comprise the horizontal integration theme of this dissertation. On the other hand, the real-world workflow introduced in the last section of this chapter represents the vertical integration theme. This workflow is revisited by each of the subsequent chapters and is partially implemented piece-by-piece based on the technology introduced and discussed in those chapters. Section 8.1 summarizes the implementation of this workflow by organizing all the pieces and presents the complete vision of the workflow design and its realization in Grid-Flow.

4 GRID-FLOW DESCRIPTION LANGUAGE

The Grid-Flow Description Language (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) (Cannan & Otten, 1992). Resource registration can be thought of as similar to the SQL Data Definition Language (DDL), while the process descriptions closely resemble the SQL 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 has been designed to provide three main functions: data registration, program registration, and process description. The grammar of GFDL is provided in Appendix A.

4.1 Data Registration

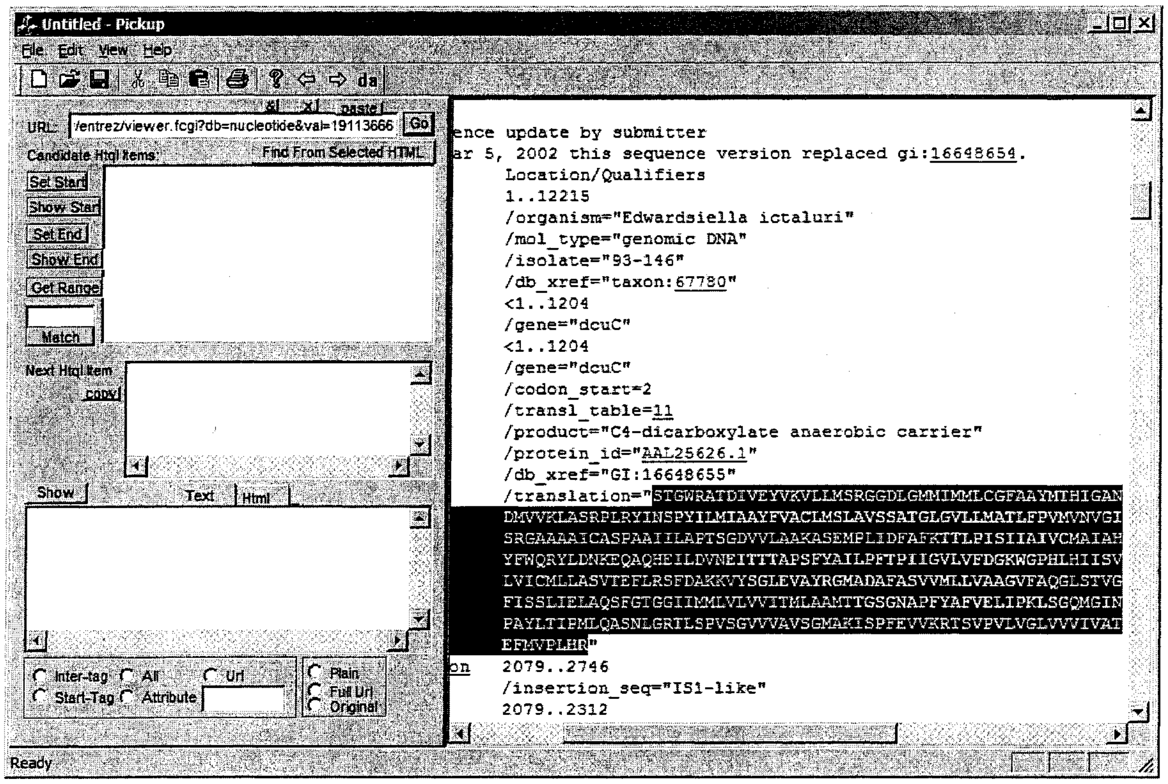
Most workflow processes involve a set of operations on various data from distributed resources. The data upon which the Grid-Flow system operates must first be registered. Registration implies recording meta-information about the data into the system repository. Generally, the registration program needs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

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

Figure 7. Syntax of Data Registration in GFDL.

From the above declarative 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. For example, if the data is a database record in a table, the SQL sentence querying the data from the table can be added as a *sql* feature. The Grid-Flow engine interprets these features and stores the corresponding meta-information into the Grid-Flow repository.

The Grid-Flow system simplifies the data registration procedure by providing users with a wizard (the interface is shown in Figure 8) 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 for processing data-registration sentences and storing the meta-information about the data into Grid-Flow. For example, when a user registers the data *TMPredInput* used in the workflow presented in section 3.5, the wizard generates the sentences shown in Figure 9 and sends them to the Grid-Flow engine.



```
ence update by submitter
ar 5, 2002 this sequence version replaced gi:16648654.
     Location/Qualifiers
     1..12215
     /organism="Edwardsiella ictaluri"
     /mol_type="genomic DNA"
     /isolate="93-146"
     /db_xref="taxon:67780"
     <1..1204
     /gene="dcuC"
     <1..1204
     /gene="dcuC"
     /codon_start=2
     /transl_table=11
     /product="C4-dicarboxylate anaerobic carrier"
     /protein_id="AAL25626.1"
     /db_xref="GI:16648655"
     /translation="STGWRATDIVEYVKVLLMSRGGDLGMMIMMLCGFAAYMIHIGAN
     DMVVKLASRPLRYINSPYILMIAAYFVACLMSLAVSSATGLGVLLMATLFPVMVNVGI
     SRGAAAAICASPAAIILAPTSGDVVLAAKASEMPLIDFAFKTTLPISIIAIVCMAIAH
     YFWQRYLDNKEQAQHEILDVNEITTTAPSFYAILPFTPIIGVLVFDGKWGPHLHIISV
     LVICMLLASVTEFLRSFDAKKVYSGLEVAYRGMADAFASVVMLLVAAGVFAQGLSTVG
     FISSLIELAQSFGTGGIIMMLVLVVITMLAAMTTGSGNAPFYAFVELIPKLSGQMGIN
     PAYLTIPMLQASNLGRTLSPVSGVVVAVSGMAKISPFEVVKRTSVPVLVGLVVVIVAT
     EFMVPLHR"
on   2079..2746
     /insertion_seq="IS1-like"
     2079..2312
```

Figure 8. Interface of the Data Registration Wizard.

```
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 9. Registration Sentences for Data TMPredInput.

4.2 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 "mpiblastp" (Darling, Carey, & Feng, 2003) 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 mpiblastp via the Globus Toolkit (Foster & Kesselman, 1999), and fetch the results from the remote side after the execution.

In Grid-Flow, 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, "<", ">", "=") and fundamental logical operations (for example, "^", "v", "!"). 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 (Swofford, 2002) and ClustalW (*ClustalW*). 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 reused as a simple process definition. Such reusable programs are called Grid-

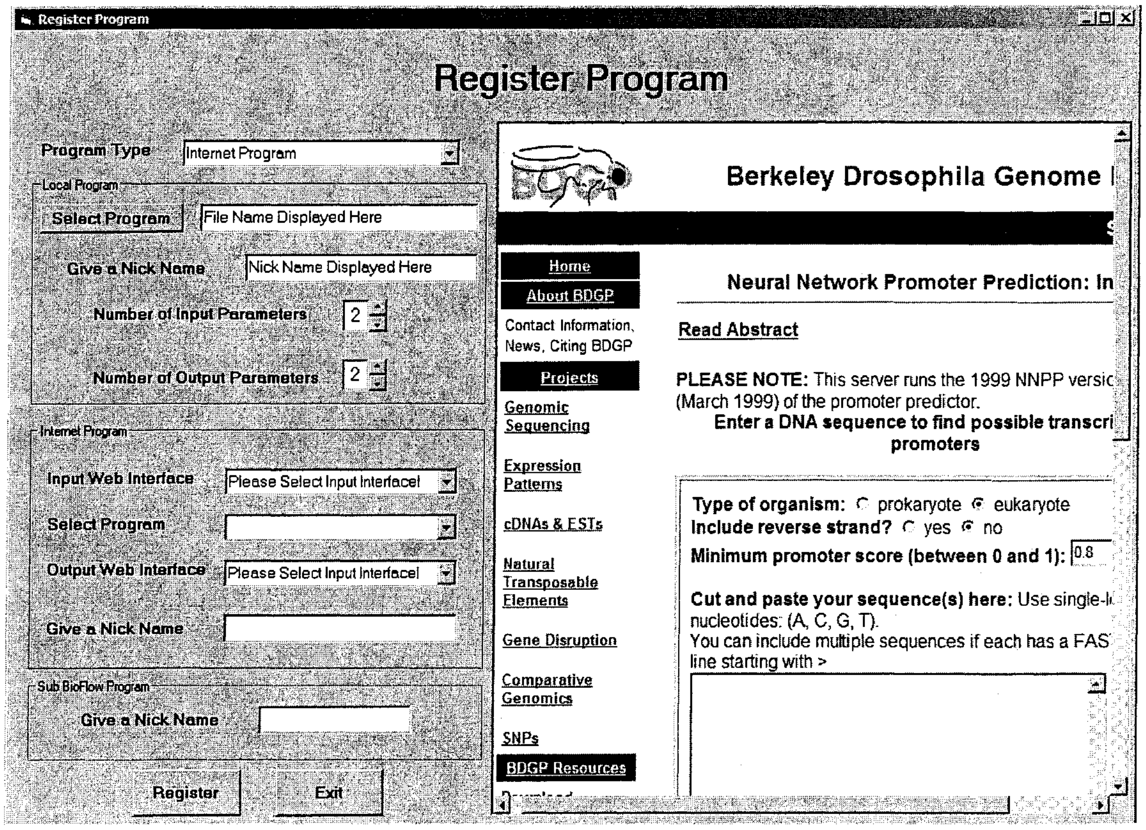
Flow Programs. Actually, Grid-Flow programs are a kind of reusable sub-process. The syntax for registering programs is listed in Figure 10.

```
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 10. Syntax of Program Registration.

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 after the program ends. The remaining syntax follows with different contents according to different types. For an Internal Program, it is unnecessary to provide any additional information. For an OS Program, additional information is provided about the program, such as the program's location, operating system(s) 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.

Similarly to the data registration wizard, the Grid-Flow system provides users with a program registration wizard (shown in Figure 11) for recording program-related meta-information. The registration wizard automatically generates program-registration sentences and submits them to the Grid-Flow engine. As an example, the sentences in Figure 12 will generate a program record for the program named *TMPred*, which is used in the example described in section 3.5.



Program Type: Internet Program
Select Program: File Name Displayed Here
Give a Nick Name: Nick Name Displayed Here
Number of Input Parameters: 2
Number of Output Parameters: 2
Input Web Interface: Please Select Input Interface!
Select Program:
Output Web Interface: Please Select Input Interface!
Give a Nick Name:
Give a Nick Name:

Berkeley Drosophila Genome

Contact Information, News, Citing BDGP

Neural Network Promoter Prediction: In

Read Abstract

PLEASE NOTE: This server runs the 1999 NNPP versic (March 1999) of the promoter predictor.

Enter a DNA sequence to find possible transcri promoters

Type of organism: ○ prokaryote ● eukaryote
Include reverse strand? ○ yes ● no
Minimum promoter score (between 0 and 1): 0.8

Cut and paste your sequence(s) here: Use single-l nucleotides: (A, C, G, T).
You can include multiple sequences if each has a FAS line starting with >

Figure 11. Interface of the Program Registration Wizard.

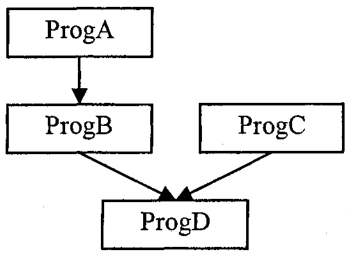
```
Register Program TMPred As OS_Program;
Input TMPredInput;
Output TMPredOutput;
Set program_source="http://ardra.hpcl.cis.uab.edu:8080/webservices/...";
End;
```

Figure 12. Registration Sentences for Program TMPred.

4.3 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 further divided. 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 GFDL can be found in (Guan & Jamil, 2003). Compared with the BioFlow language that it is derived from, GFDL is equipped with more features to support program integration in Grid environment and more abilities to describe advanced flow control structures such as *AndSplit* and *OrJoin*.

According to the workflow models provided in (Aalst & Hee, 2002), 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, which 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 13 can be described as the expression on the right with GFDL.



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

Figure 13. Example of Sequence Structure.

In Figure 13, ProgA, ProgB, ProgC, and ProgD are all registered programs in the Grid-Flow repository (described in section 3.3). This program-driven structure means the output of ProgA will be transferred to ProgB as an input parameter; then 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 natural sequence, in which the corresponding tools are used for real workflow process. This makes it simple and intuitive for users to model their workflow procedure with GFDL.

Sequence structure has an explicit dependency between its activities. This relationship relies not only on the input/output connection, but also reveals the execution order between activities, which is, in turn, dependent on the input/output connection. If no input/output connection exists between a set of activities, those activities can, however, be executed in any order. A parallel structure is used to describe this unordered relationship between activities. Figure 13 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.

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 14 captures the same meaning as the single sentence in Figure 13 expresses via assignment statements.

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

Figure 14. Extended Expression of Sequence Structure.

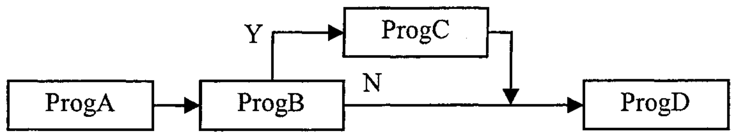
In the choice structure, one particular activity is selected out of two or more possible activities according to the running conditions set in the process. The following syntax describes the choice structure.

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. Similarly to conventions often used in 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 15, if

ProgB (ProgA()) returns true, ProgC will be executed before ProgD, otherwise only ProgD will be executed after the execution of ProgB and ProgA. The corresponding GFDL sentences are provided in the lower part of Figure 15.



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

Figure 15. Example of the Choice Structure.

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 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 denoted 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 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. This syntax is 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.

4.4 Summary

With the data/program registration templates and the flow controlling structures provided by GFDL, a user can register all of the needed information of a workflow and describe it via GFDL scripts in Grid-Flow. Though it is powerful and flexible on recording meta-data and describing workflow processes, GFDL is not so easy to learn and master by many classes of potential users, especially for those users who are lack of programming experiences. Thus, a more intuitive graphical user interface, described in the next chapter, is provided by Grid-Flow to assist users in designing scientific workflows.

5 GRAPHICAL USER INTERFACE

The User Interface component is responsible for the interaction between users and Grid-Flow system. The four major functionalities of the User Interface component are described as the following.

- **Help the user register programs and data.** Grid-Flow guides the registration procedure of data and programs with different tools. To register a data source, a user first executes PickUp (Chen & Jamil, 2003) (see Figure 8 for the interface) in order to create a Data Description File (DDF) for the data. The DDF stores data-item-related information, such as the data item name, format, keywords, and wrapper. After saving the DDF to the local disk, the user interface automatically generates the data registration procedures in GFDL and sends those sentences to the Grid-Flow engine. The Grid-Flow engine compiles the procedure, executes it, and registers the data into the Grid-Flow repository. The program registration procedure is similar to data registration except the user interface provides a step-by-step wizard (see Figure 11) instead of executing the PickUp component. In the registration wizard, users need to select the type for the registering program first. Then user interface will promote the user with necessary information needed for registration. After users input all of the necessary information, the user interface will automatically generate the corresponding Grid-Flow registration scripts and send them to the Grid-Flow engine. The Grid-Flow engine executes the registration scripts and registers program into the Grid-Flow repository.

- **Help the user design or reload the Grid-Flow process description.** Grid-Flow provides users with a graphical user interface to design workflow processes. To describe the workflow process, a user can use provided graphic components, (for example, transitions, places, and connections) to draw a Petri net domain model (as described in section 2.6.3). The system can automatically translate this Petri net model into GFDL and execute the process thereafter. A user can also save the model of the process to local disk for future modification and execution. In addition, the user interface can help the user to reload existing Grid-Flow processes and integrate them into other processes as subsystems.
- **Help the user execute a Grid-Flow process and monitor the execution.** Users can send commands to the Grid-Flow engine through the User Interface. These commands can accomplish the operations such as starting, suspending, resuming, stopping a Grid-Flow process, and querying process status.
- **Display the output result to the user.** Grid-Flow has already integrated several most commonly used display functions for output data, like the integer number display, string display, text file display, and HTML/XML document display. However, because of the variety of output data formats, Grid-Flow allows users to register their own display functions. The User Interface can utilize both the system functions and user-defined functions for display. Correspondingly, a user's response input can also be transmitted into the Grid-Flow engine through the User Interface.

5.1 Petri Net Modeling

There are at least three categories of method that have been developed in the Grid workflow research community to describe the workflow process in a Grid environment. A Grid workflow process can be described in a scripting language (such as GridAnt (Laszewski et al., 2004) and XCAT (S. Krishnan, Bramley, Gannon, Govindaraju, Indurkar et al., 2001)), in a graphical model (like Condor DAGman (*Condor: The Directed Acyclic Graph Manager*, 2003) and FhRG (*Fraunhofer Resource Grid*)), or in a mixture model of both (like WSFL (Leymann, 2001), BPEL4WS (T. Andrews et al., 2003), and GSFL (Sriram Krishnan et al., 2002)). Scripting languages may be effective and efficient for skilled users, but are not so intuitive for users who are unfamiliar with the control logic and dataflow. On the contrary, graphical models allow even users without expertise to describe the complex workflow process with only a few basic graph elements. That is, users only need to know what the graph elements of the data and program components are and how to connect them to control the executing flows. A graphical model, namely the Petri net (Peterson, 1977), is used here to describe the Grid workflow process. Compared with commonly used DAGs, the Petri net model has more powerful ability to describe various kinds of workflows, as are described in earlier chapters of this dissertation.

Although practically speaking, GFDL is powerful enough to describe most current existing workflow processes accurately, 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 graphic tools that automatically translate the graphical model into the workflow description language using techniques of generative programming (Czarnecki & Eisenecker, 2000). Thus, a graphical 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.

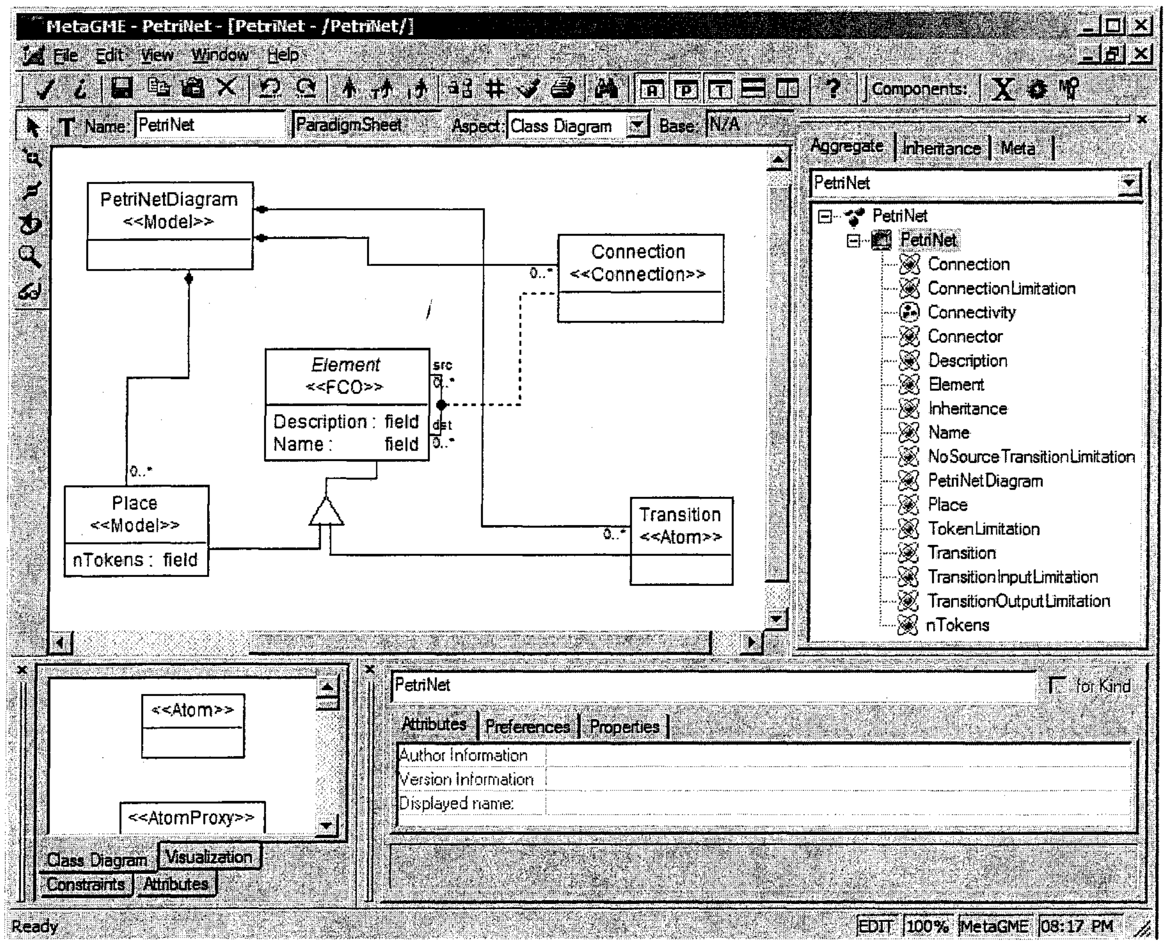
A workflow process is usually described as a Petri net by intuitively mapping programs as transitions, status as places, and data as tokens. Although the concept of a Petri net has been acknowledged to be one of the most powerful tools to describe multi-task procedures, especially for asynchronous and concurrent tasks, the Petri net is seldom used in practical scientific workflow systems because users are required to be equipped with the knowledge of Petri net modeling for using it. The work described in this dissertation 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 toolset (Akos Ledeczi et al., 2001) is used as the foundation of the modeling language.

5.2 Workflow Meta-model

GME (A. Ledeczi et al., 2001) is a promising toolset that supports the easy creation of Petri net models. In the Grid workflow domain, the data and computing resources are mapped onto grid resources, as well as the relationships are treated as the control flows which dominate the execution of workflow process.

Three steps should be followed to model a Grid workflow process with a Petri net model in the GME. First, a meta-model is created to characterize the basic elements of the Petri net, such as places, transitions, connections, and tokens. Constraints and attributes of these elements are also integrated in this meta-model. The second step is to let the user define an application (domain-specific) model with the basic elements of a Petri net. In an application model, a transition represents a program or computing resource. A place represents a particular state in the process. Thus, a connection between two places and one transition means the process transfer from one state to another state through the execution of a program. Users can compose an application model by defining a set of states of the workflow process and choosing proper applications/programs to connect those states to fulfill the control flow logic. The third step is to translate the application model to the abstract (virtual) workflow language. This could be done automatically by setting the interpreter in the GME. After the modeling process, an executing script of the Grid workflow language is generated and transferred to workflow engine to execute.

To model the workflow process with a Petri net in GME, a meta-model (shown in Figure 16) need to be created to describe the basic modeling components, their attributes, relationships, constraints, and visualization preferences. In the class diagram of the meta-model, a Petri net diagram is composed of classes representing *places*, *transitions*, and *connections* (Peterson, 1977). From the viewpoint of class hierarchy, the *place* and *transition* are sub-classes of the super-class *element*, which has two common attributes, *name* and *description*, to identify and describe the place and transition objects. The class *place* has an attribute *nTokens* for recording the number of tokens stored in the *place*. There exists a many-to-many association between *element* objects, which means any *element* object, as a source (*src*), can be connected with any number (including zero) of element objects, as destinations (*dst*). This association is implemented as an association class *connection*.



Name: PetriNet
ParadigmSheet
Aspect: Class Diagram
Base: N/A

PetriNet

Author Information	
Version Information	
Displayed name:	

Figure 16. Meta Model of Petri Net in GME.

Since the *connection* relationship in a Petri net workflow model can only exist between different kinds of fundamental objects, that is, from a *place* to a *transition* or vice versa, some constraints must be added on the connection to restrict the type of objects that it can connect. GME provides an interface for the user to define the constraints based on the Object Constraint Language (OCL) (Booch et al., 1998). Figure 17 shows one of the constraints, namely *ConnectionLimitation*, defined in the Petri net meta-model to restrict the types of the connected objects of a connection.

```
parts("Place")->forAll(x| x.connectedFCOs("dst")->forAll(y | y.kindName
= "Transition"))
and
parts("Transition")->forAll(x| x.connectedFCOs("dst")->forAll(y |
y.kindName = "Place"))
```

Figure 17. ConnectionLimitation Constraint.

This constraint defines two conditions that a *connection* must satisfy:

1. For any *place* object participated in a *connection* as a src, the dst object must be in the type of *transition*; and
2. For any *transition* object participated in a *connection* as a src, the dst object must be in the type of *place*.

This constraint is called an association constraint since it is based on an association class *connection*. In addition to association constraints, GME can help users to define constraints based on classes, operations, and attributes. For example, to restrict that a *transition* object must have at least one input connection, a designer can define a class constraint on class *transition* as shown in Figure 18.

```
parts("Transition")->forAll(x| x.connectedFCOs("src")->size() >=1)
```

Figure 18. NoSourceTransitionLimitation Constraint.

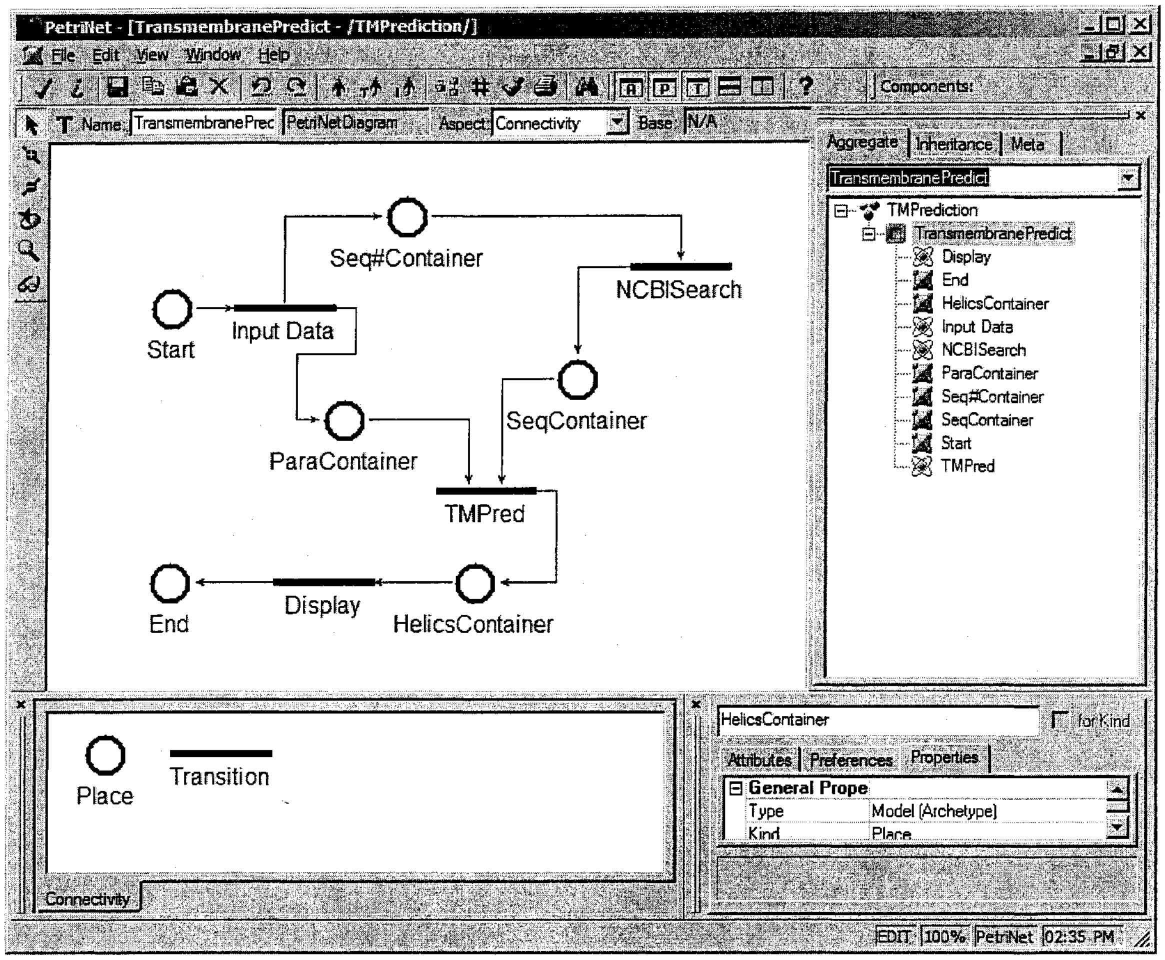
This constraint defines that for any *transition* object in a Petri net, the number of objects that are connected with it as sources must be larger or equal to 1. This guarantees that any *transition* object connects at least one *place* object as an input source. Thus, the whole Petri net would not start with a transition, but with a place. Constraints can also facilitate the user to manage the Petri net domain model by setting restrictions on the attributes. For instance, to make sure a domain model (a workflow) does not accumulate too many tokens (data or tasks) in one *place*, a user can set an attribute constraint associated with the *place* class as *self.nTokens<=5*. This constraint limits the number of *tokens* stored in a *place* to be less than 6. If domain model violates this constraint, GME will report an error to the user at the design time.

The visualization preferences for each component in the meta-model can be easily set with the GME interface so that each building block has a unique and meaningful expression in the domain model. Following the idiom of the Petri net used in (Peterson, 1977), a circular icon and a bar is used to represent the *place* and *transition*, respectively. Other visualization features, like the color of the icon, the location of the displaying name of the class, the auto router preference of connections, can also be set in the preferences tab of the attribute panel in the visualization view.

The meta-model is itself a correlated result of the Grid-Flow system. It provides users a sound base for modeling workflow processes as Petri net domain models. No other workflow management systems, to the author's knowledge, clearly define such a meta-model of Petri net in theory. After creating all the modeling components and the constraints in the meta-model, users construct their own domain models by mapping workflow processes into combinations of Petri net building blocks.

5.3 Workflow Domain Model

After meta-model designers define and register the meta-model with the local operating system, the Petri net model can be used as a modeling tool in GME to describe domain models. The GME interface used to create domain models is the same one for building meta-models, except that the fundamental building components are different. The basic components used for domain models are not classes, attributes, or constraints, but rather are building blocks defined in the meta-model; that is, places and transitions. The interface to define domain models in GME is shown in Figure 19. This interface includes three panels and one canvas. The building components of places and transitions are displayed in the *part browser* panel. Users can drag the place and transition icons from the *connectivity* tab and drop them on the canvas to build domain models. On the right side of the canvas is the *tree browser* panel showing the tree structure of all the components in the domain model, their inheritance relationships, and their meta-data. The properties of each component in the canvas can be set in the *attribute* panel. The whole domain model, associated with all of the property and attribute settings, can be saved in a MultiGraph Architecture (mga) file. This file is interpretable by the interpreter of the domain model for generating executable scripts, which, in this case, are the GFDL sentences.


PetriNet - [TransmembranePredict - /TMPrediction/]
File Edit View Window Help
Components:
Aggregate Inheritance Meta
Seq#Container
NCBISearch
Start
Input Data
SeqContainer
ParaContainer
TMPred
End
Display
HelicsContainer
Place
Transition
Connectivity
Attributes Preferences Properties
for Kind
EDIT 100% PetriNet 02:35 PM

Name: TransmembranePrec PetriNetDiagram Aspect: Connectivity Base: N/A

TransmembranePredict

- TMPrediction
  - TransmembranePredict
    - Display
    - End
    - HelicsContainer
    - Input Data
    - NCBISearch
    - ParaContainer
    - Seq#Container
    - SeqContainer
    - Start
    - TMPred

HelicsContainer

General Prope	
Type	Model (Archetype)
Kind	Place

Figure 19. GME Interface for Defining Domain Models.

Transition icons on the canvas represent different programs that are executed in a workflow process. Places in a domain model play two roles: one role is to store tokens generated by the connected source transitions; the other is to indicate the current status of a workflow process. Once a transition finishes its execution, it usually generates tokens and sends those tokens to places that are connected with it via connections. If a token arrives in a place, the arrival action to the place indicates that the workflow has finished the execution of the previous transition and is ready for invoking the next transition.

Based on the roles played by places and transitions, users have two methodological options for building the domain model. One option is to build the transitions first, then consider the status and connections between and among transitions. Within this methodology, the building procedure can be divided into two steps. First, a user needs to analyze the programs used in the workflow process, and create representations for them on the canvas. Then the second step is to consider the data generated from those programs, build the data containers to store them, and connect containers with corresponding programs. For example, in the use case described in section 3.5, a workflow process uses four programs: two Grid-Flow internal programs *InputData* and *Display*, and two OS programs *NCBISearch* and *TMPred*. A user can draw four transition icons on the canvas and change their names to represent these four programs respectively. The data generated by these programs are analyzed as follows: *InputData* asks users to input two types of data and sends them to two different programs. Thus, two data containers should be created. Both *NCBISearch* and *TMPred* generate one data. One data container for each program is sufficient. *Display* does not generate any data but present the data to the user. So no data container is needed for *Display*. As shown in Figure 19, four data containers (places) are created and denoted *Seq#Container*, *ParaContainer*, *SeqContainer*, and *HelicsContainer*. These data containers are connected with their transitions correspondingly. Furthermore, two special places, which are used to represent the start (the *start* place) and end (the *end* place) statuses of the workflow process, are also created. The *start* place is connected with the program *InputData* since *InputData* is the first program to be executed. It provides the initial token to *InputData*, and therefore triggers the whole workflow. The *end* place stores the token generated by the process and indicates the workflow is finished.

The other method to build the domain model is to consider the status of the workflow first, then to insert transitions among statues. This method can also be applied to the example demonstrated in section 3.5. A workflow process of the example workflow should experience the following status according to the order of execution: pre-input-data, post-input-data, pre-NCBI-search, post-NCBI-search, pre-TMPred, post-TMPred, pre-display, and post-display. After reconsidering the order of the execution of each program, a user can easily find that some statuses can be combined together because they are representing a continuous status between which the workflow process remains unchanged. For example, post-TMPred can be combined with pre-display since no program is executed between them. Therefore, five statuses are left after the combination step. They are as follows: pre-input-data, pre-NCBI-search, pre-TMPred, pre-display, and post-display. Users can then design places corresponding to these statuses, and insert transitions among places to represent programs. Note the mapping between statuses and places may not be a one-to-one mapping. For example, since program *TMPred* needs two input data from two different programs *InputData* and *NCBISearch*, the status pre-

TMPred should be translated into two places to denote the data arriving from different transitions.

Both of these two methods have their own pros and cons. The first method focuses on modeling the programs. The mapping between the workflow process and the domain model is intuitively straightforward. But the mapping is program-centric instead of data-centric, which may not be easily accepted by some users. In addition, the start and end statuses are treated as special cases in this method. The second method clearly defines the status of each step in the workflow. The mapping in this method is not so straightforward since in most cases users need to combine or split status to model places in a workflow. However, the status-oriented definition matches the most popular understanding of workflows. Users can choose different methods to model workflow processes according to their preferences and experiences.

5.4 Interpreter

One of the advantages of GME is that it can automatically generate GFDL based on the Petri net models, which exempt users from the burden of writing code for workflow processes. The generation of GFDL from Petri net models is implemented by a GME model 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.

GME defines a high-level Java interface (*High-Level Java Interface to GME -- Users Manual Version 1.0*, 2004) that facilitates model designers to build interpreters for domain models. This interface provides support for easy traversal through domain models of the containment hierarchy, the atoms, the connections, and the references. An interpreter for the domain model needs to implement the function *invokeEx* in the Java interface as shown in Figure 20. A *JBuilder* object is passed as a parameter to this function for accessing the whole hierarchy of the domain model. The hierarchy is denoted the builder object network (BON) in which each model component, such as atoms, references, and connections, has a corresponding builder object. When the GME user starts the interpreter, the BON is first created based on the components in the domain model, and then the function *invokeEx* is called. Thus, the interpreter can traverse the BON, apply predefined operations on builder objects and their relationships, and generate useful information, like configuration files, database schema, and source code.

```
import org.isis.gme.mga.MgaFCO;
import org.isis.gme.mga.MgaFCOs;
import org.isis.gme.mga.MgaProject;
… …
public class GridFlowInterpreter implements BONComponent {
  … …
  public void invokeEx(JBuilder builder,         // root folder of the model
                       JBuilderObject focus, // model that has the focus
                       Collection selected,    // objects that were selected
                       int param)                     // not used
  {
        // Traverse the BON and generate GFDL
        … …
  }
}
```

Figure 20. Java Interface for Domain Model Interpreter.

Grid-Flow has a Grid-Flow interpreter implemented based on the high-level Java interface to translate Petri net domain models into GFDL scripts. Figure 20 shows the Java class *GridFlowInterpreter* that realizes the *BONComponent* interface for the GFDL translation. The pseudocode of the method *invokeEx* is displayed in Figure 21. By calling this method as an interpreter function, GME can compose the executable GFDL script for a workflow process with tokens, places and transitions in the Petri net domain model.

```
METHOD invokeEx PARAMETERS builder, focus, selected and param
    1. Search builder BON to find the start place in the domain model
    2. Put the Start place in the empty place set Π
    3. FOR each transition in the domain model, DO
        3.1 IF all the input places of a transition τ is in the set Π, THEN
            3.1.1 Generate the GFDL for the transition τ
            3.1.2 Remove the input places of transition τ from the set Π
            3.1.3 Insert all the output places of transition τ into the set Π
    4. REPEAT step 3 UNTIL the End place enters into the set Π
```

Figure 21. Pseudocode of the Method *invokeEx*.

The pseudocode in Figure 21 demonstrates that the interpretation procedure starts from the *Start* place, goes through each transition in the BON following the order defined in connections, generates GFDL scripts, and stops at the *End* place. When applying the interpreter to the domain model described in Figure 19, GME can generate the GFDL script as shown in Figure 22.

```
Set _Seq#ContainerContent =InputData();
Set _ParaContainerContent=InputData();
Set _SeqContainerContent =NCBISearch(_Seq#ContainerContent);
Set _HelicsContainerContent=TMPred(_ParaContainerContent,
_SeqContainerContent);
Set Display(_HelicsContainerContent);
```

Figure 22. GFDL Script Generated by the Interpreter.

The GFDL script for the workflow of Transmembrane Regions Analysis is generated as follows:

1. The interpreter first puts *Start* place in the set Π;
2. After checking all the transitions in the model, the interpreter finds that only transition *InputData* can be processed. So it generates the first and second GFDL sentences, removes the *Start* place from the set Π, and brings two places *Seq#Container* and *ParaContainer* into the set Π;
3. The interpreter checks the transitions again. This time, transition *NCBISearch* is ready to be processed. Therefore the interpreter generates the third line of GFDL sentence. The place *Seq#Container* is removed from the set Π and the *SeqContainer* is added;
4. The transition *TMPred* is now the only candidate for the interpretation since all of its input places are in the set Π. The interpreter processes this transition and generates the fourth line of GFDL sentence. Places *ParaContainer* and *SeqContainer* are removed and the place *HelicsContainer* is added into the set Π;
5. The transition *Display* is finally selected to be interpreted into the fifth line of GFDL sentence. The *End* place is brought into the set Π. Thus, the interpreter procedure stops successfully.

The implementation of the interpreter for workflow models has several drawbacks. First, the code generated by the interpreter is not so efficient compared with the scripts written directly by an experienced GFDL user. As shown in Figure 22, each transition in the domain model is translated into a GFDL sentence. Temporary variables are automatically generated to store the intermediate results for each program in the workflow. Since the Grid-Flow engine cannot distinguish the intermediate data from those permanent data that users want to investigate after the workflow execution, it needs to explicitly store the intermediate results and register their meta-data in the Grid-Flow repository. In this case, the data between programs cannot be pipelined or streamlined. That is, the data cannot be fed to another program for processing while it is being generated. Workflows without data streamlining not only hinder the parallel execution of data-dependent programs, but also waste large amounts of time and space on storing temporary data. The code generated by this interpreter implementation may diminish the efficiency of the execution for some workflows that need to process large amounts of data.

Second, the interpreter cannot parse moderately complex flow control structures. Most flow control structures, such as the sequence structure, the parallel structure, and the choice structure, can be translated into appropriate GFDL scripts by the interpreter. The loop structure, however, cannot be parsed by the interpreter since it contains pointing-back connections. For example, the End-controlled Loop Structure in Figure 23 has a connection pointing from the *OrSplit* place to the *ProgA* transition. According to the logic embedded in the interpreter, the *ProgA* can never be parsed since its interpretation requires the place *OrSplit* in the set Π, while the *OrSplit* cannot enter into the set before the *ProgA* is parsed. To bring the place *OrSplit* into the set Π, the transition *ProgB* must be parsed. The *ProgB* is dependent on the *ProgA*, of which the interpretation requires the place *OrSplit*. This dead loop cannot be solved by the interpreter. Thus, it needs the GFDL user's intervention.

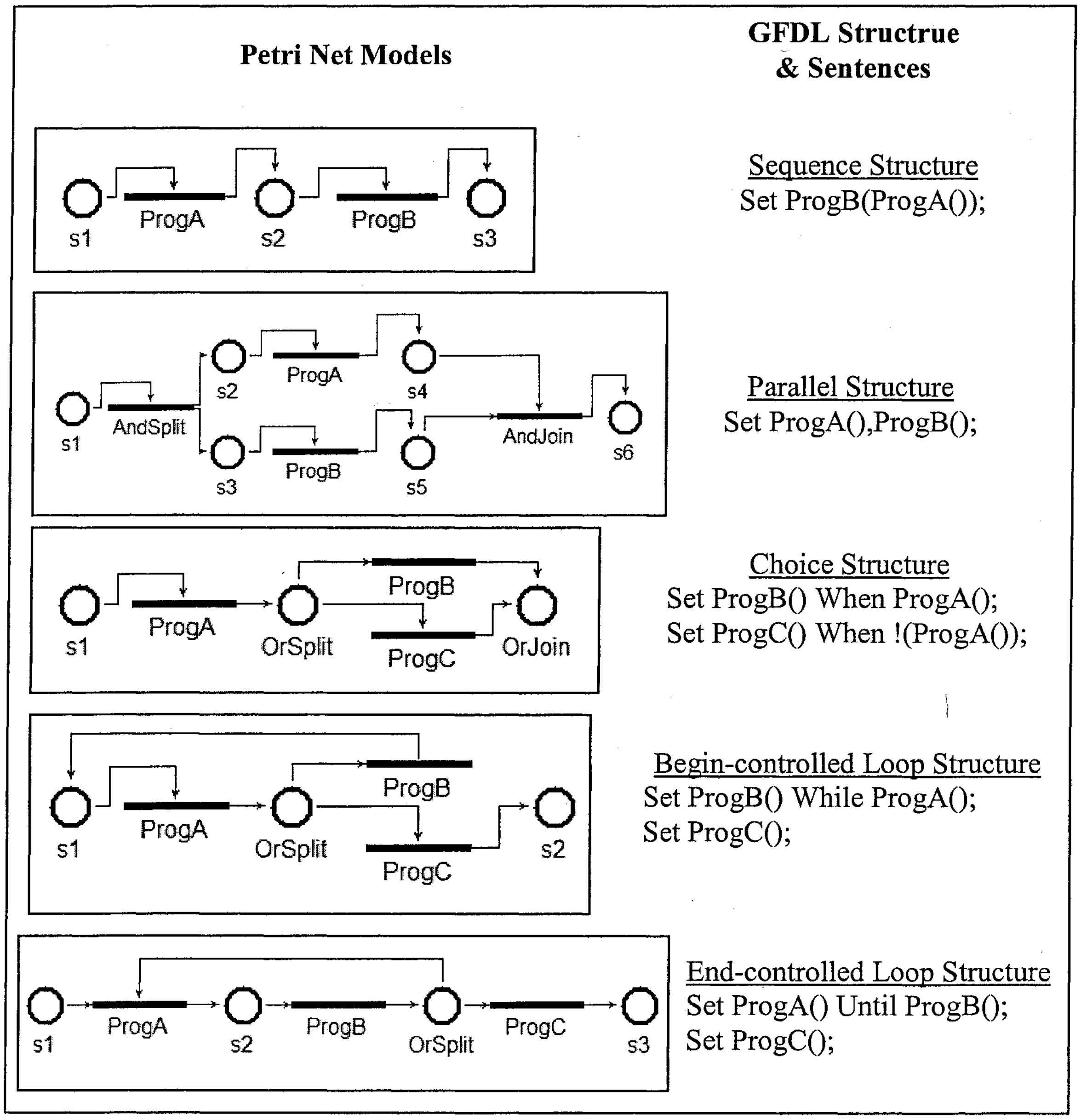
Third, the interpreter cannot support some advanced features of the Petri net model, which limits the positive impact of the Petri net on workflow process modeling.

For instance, this interpreter does not distinguish the color of tokens in Petri net models. Different colors of a token represent different attribute values of a data. Petri net models usually set filters on the color of tokens as pre-conditions to prevent some tokens from initializing a transition. This feature is especially useful for implementing the *OrSplit* place. As shown in the Begin-controlled Loop Structure of Figure 23, the value of a token generated by *ProgA* can determine if the workflow keeps executing the loop body (*ProgB*), or jumps out the loop for the following transitions (*ProgC*). Since the value of the data (the color of the token, correspondingly) cannot be recognized by the interpreter, the *OrSplit* cannot be translated into a GFDL script.

However, all of the drawbacks noted above can be overcome if the experience of a GFDL user can be integrated into the interpreter design as routine patterns. Figure 23 shows some patterns and their corresponding flow control structures embedded in the GFDL interpreter.

5.5 Graphical Design for Workflow Process

There are many existing approaches for modeling and analysis of processes. A Petri net is one of those formal approaches that are based on established formalism (Aalst & Hee, 2002). The use of Petri net has a number of major advantages. First, it guarantees the precision of process definition. Compared with informal diagramming techniques, a Petri net avoids any ambiguous definitions because of its precise structure limitations. Second, the formalism of Petri nets can be used to verify logical properties and analyze system performance.



GFDL Structrue & Sentences

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 23. Mapping between Petri Net Models and GFDL.

However, a Petri net has its own disadvantages. A Petri net is not so easy or intuitive to be grasped by users who do not have the experience on model-driven computation. To use Petri net modeling workflow processes, a user needs to know at least the concepts of transition, token, and place, as well as the flow controlling structures, such as *OrJoin* and *AndSplit.* Users also need to know how to convert the workflow procedures in their mind to the models in Petri nets. A user without any special training on Petri nets may not be able to use Petri nets directly for workflow process modeling.

Since Petri nets are not such an intuitive tool for untrained users to model workflow process, a more straight-forward graphical mapping approach needs to be either developed or adopted to help users describe the workflow process in WFMS. This mapping approach should possess the following competencies.

1) This approach should be able to represent the programs and data used in the workflow process;
2) This approach should be able to describe the relationship between data and programs;
3) This approach should be able to model the flow controlling structures.

There is no modeling approach of which the author is aware in the current workflow research community that can satisfy the three requirements provided above, and more important, be easily learned by new WFMS users. Therefore, a new graphical process modeling approach, namely the Data/Program Chart, is designed in Grid-Flow to help users model the workflow processes in their mind. Data/Program Chart approach is designed to be a concise, intuitive, and easy to learn approach. Although the model described with this approach cannot be directly interpreted by the Grid-Flow user interface, the Data/Program Chart approach is still useful for communicating the workflow specifications between workflow end-users and workflow system experts. End-users can use the Data/Program Chart approach conveying workflow processes to the workflow designer. The workflow designer can sequentially translate those processes into Petri net models or directly to GFDL. The Data/Program Chart approach acts as a bridge and an interface between end-users and workflow designers, which also lowers the learning curve for end-users to use WFMS. The following sections present more detailed information about Data/Program Chart, its usage, and its translation into the Grid-Flow language. The translation between Data/Program Charts and Petri net models needs more expertise and is mainly based on workflow designers' improvisation. Although these two models are used to describe workflow processes with the same flow controlling structures, there is no formal procedure currently existing for users to follow in order to map them to each other. This dissertation does not describe potential translation and mapping techniques; it remains for future work.

5.5.1 Data/Program Chart

A Data/Program Chart consists of three kinds of component: *data, program,* and *start/end position.* An oval is used to represent data; a program is shown as a rectangle; and the start/end position is presented as a circle. Figure 24 shows the simple Data/Program Chart corresponding to the process illustrated in section 3.5. This chart consists of four programs (InputSN, NCBISearch, TMPred, and Display), two data (PSeq and TMPredpar), and two start/end positions (Start and End). This network models the process for prediction of transmembrane regions.

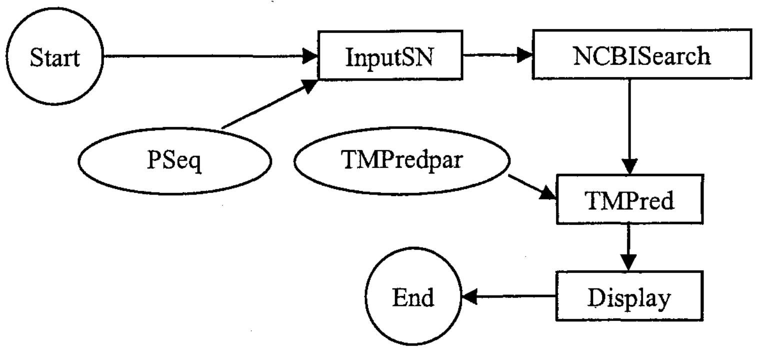


Figure 24. Example Data/Program Chart: Prediction of Transmembrane Regions.

Data, program, and start/end position in a data/program chart can be linked by directed arcs. There are two types of arcs, described as follows: those that indicate the starting and ending programs of the process, and those that present the flow of data between data/program and program/program. The former one conveys the signals of starting and ending the workflow process, while the latter indicates the path that data flows.

In any Data/Program Chart, there are only one start position and end position. The start position indicates via directed arcs which programs the process should start from. A process may start from multiple programs simultaneously, thus, the Data/Program Chart may have multiple arcs connected from the unique start position to multiple starting programs. Similarly, a process may end upon multiple programs, thus, the Data/Program Chart may have multiple arcs connected from ending programs to the unique ending position. Note that the start/end position can only be connected with programs, never with data. The start/end position can also play the role as the "accessing points" when a small process is integrated into a large, hierarchical process. That is, the upper-level process invokes the subprocess from the subprocess' start position, and gets response from the end position.

Based upon the arcs between data/program and program/program, one can determine the inputs of any programs. A data *d* is the input of a program *p* if and only if there is a directed arc running from *d* to *p*. Similarly, an output of a program *p1* is the input of a program *p2* if and only if there is a directed arc running from *p1* to *p2*. For example, in Figure 24, *PSeq* is the input data of *InputSN* because there is an arc from *PSeq* to *InputSN*. Program *TMPred* has two input data sources, one is data *TMPredpar*, the other is the output of program *NCBISearch*. Note that data can only connect with a program, but a program can be connected with either data or other programs.

A program may only fire if all of its inputs are available. When a program gets all of its inputs, it turns into the status of "enabled." Then the program can load its inputs, run and generate the output. The output generated may, therefore, enable other programs and push the progress of the process forward. The first "firing" signal is sent by the start position. The end position responses for collecting all of the outputs generated from the programs it connects with, and then notify the end-user of the completion of the process.

5.5.2 Mapping Process onto Data/Program Chart

A process is used to indicate in which way a particular category of applications should be handled in a workflow management system. The process contains all of the information about the application, such as which programs should be carried out, in which order the programs should be performed, and what data need to be used in the application. It therefore is obvious to define a process using a Data/Program Chart. Each process has one entrance and one exit, which can be exactly mapped to the start/end position of a Data/Program Chart, respectively. The data and conditions in the process can be mapped to data in the Data/Program Chart. Similarly, the tools and programs used in a process can be mapped to programs in the Data/Program Chart. The order in which the programs should be performed can be expressed as arcs in Data/Program Chart.

The procedure for mapping process onto the Data/Program Chart is intuitive for the end-user. A procedure with three steps can be followed. In the first step, users need to plan what data and programs should be involved in the process. After collecting all of these components of a process, users can draw corresponding ovals and rectangles of a Data/Program Chart on the paper. The second step is to connect data and programs using arcs. Data should be connected with corresponding programs, as well as programs having input/output streamlining relationships should be connected. Users can also describe the executing order with some patterns provided in Figure 25. The patterns are introduced in detail in the following paragraphs. The third step is to indicate the entrance and exit of the process, that is, a user should draw the start/end position and connect them with proper starting and ending programs.

Since the relation between data and programs is relatively simple in contrast with the relation among programs, all of the related data in the pattern are omitted and the following focuses on the arcs between programs. Pattern (a) in Figure 25 represents the sequential execution of three programs: A, B, and C. This pattern is the most commonly used pattern in streamlining data analysis. It means the output of A is fed to B, and the output of B is treated as one of the inputs of C. Consequently, A is executed prior to B, and B is executed prior to C. Pattern (b) is generally called parallel and-split. It means that the output of A should be feed to B and C, thus, B and C can be executed in parallel after the execution of A. Pattern (c) and (d) are both selective routings. Program A decides which branch should be selected according to particular conditions, and then B or C is executed following the branch. Program A is regarded as or-split since the two branches from it are selected alternatively. Note that the difference between "or-split" and "and-split" is the two arcs of "or-split" start from different sides of the rectangle A, while arcs of "and-split" start from the same side of the rectangle. Patterns (e) and (f) are both iterative routings. The difference between them is the location of the program that makes the choices. In pattern (e), program B evaluates the conditions and decides to either repeatedly execute A, or transit to C. Program A in pattern (f) plays the same role except it first checks the condition, and then goes into the iteration.

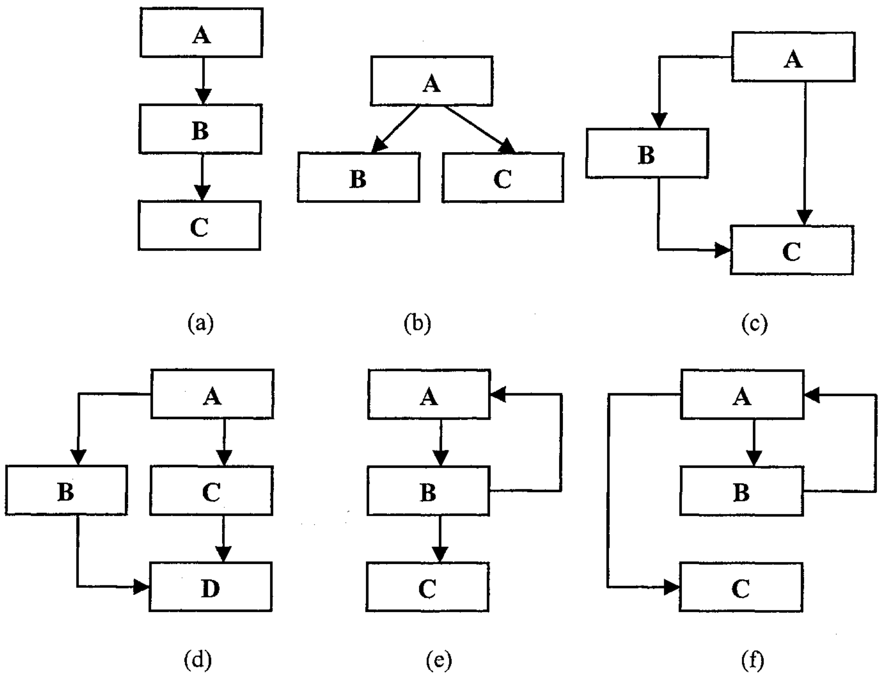


Figure 25. Data/Program Chart Patterns.

5.5.3 Translating Data/Program Chart into GFDL

The Data/Program Chart is meaningless to a computer, though it is helpful for users to describe the process. Thus, one needs to translate a Data/Program Chart into a language that computers can understand, compile, and execute. In Grid-Flow, the simple Data/Program Chart can be directly translated into GFDL without going though the Petri net modeling step. Complex Data/Program Charts may need the help of Petri net modeling to translate the embedded workflow definitions into GFDL indirectly.

The translation between Data/Program Chart and GFDL could be performed by workflow designers. The translation process starts from the start position in the chart, follows the direction of the arcs through the network, and finishes at the end position. During the translation, data are translated as parameters of the programs which are connected with corresponding arcs. Programs in the Data/Program Chart are translated into executable programs in GFDL. Streamline programs connected with arcs in Data/Program Chart are translated into a sentence with cascading programs in GFDL. Patterns in Figure 25 will be translated into the sentences and structures in GFDL (as shown in Figure 26), respectively. Note that x, y are both temporary variables. NULL means an initial empty value for the variables.

5.6 Summary

Two workflow modeling approaches have been introduced in this chapter; they are the Petri net modeling approach and the Data/Program Chart. Petri net modeling is a formal defined mechanism that has a sound theoretical foundation and strict modeling procedures (from a Petri net meta-model, to workflow domain models, then to GFDL scripts translated by the Grid-Flow interpreter.) The Data/Program Chart method, however, is not so profound from a theoretical perspective but proves agile because of its component structure and because it is easy to learn for relatively unsophisticated users. After modeling the workflow and translating it into GFDL scripts, a WFMS need to check the availability of data and programs, and a schedule for execution of programs to process data is also required. The data and program integration components are crucial for a WFMS on the aspects of accessing heterogeneous data and orchestrating distributed programs. The next two chapters discuss the data and program integration components in Grid-Flow.

```
(a)     Set C(B(A(...)));
(b)     Set x=A(...);
        Set B(x);
        Set C(x);
(c)     Set C(B(x)) When x=A(...);
        Set C(x) When .NOT. x;
(d)     Set D(B(x)) When x=A(...);
        Set D(C(x)) When .NOT. x;
(e)     Set x=NULL;
        Set y=A(x, ...) Until x=B(y);
        Set C(x);
(f)     Set x=NULL;
        Set y=NULL;
        Set y=B(x) While x=A(y, ... );
        Set C(x);
```

Figure 26. Translated Data/Program Chart Patterns in GFDL.

6 ONLINE RESOURCES REGISTRATION

Large-scale scientific researchers request, intuitively, the integration of vast amounts of online diverse data, the cooperation among ever changing and sophisticated analysis tools, as well as the collaboration between physically distributed investigators and organizations. With the development of enabling workflow management techniques, workflow researchers now urgently need a system that can be used to model the distributed and heterogeneous online resources as one unique-form source, and to design workflow process using online tools without taking too much concern about the underlying details of data transformation. To address the integrated data analysis issue presented in current workflow research, a powerful mechanism is needed to unite the online data and programs together. The development of Internet and distributed computing in recent years catches up with the rapid-evolving requirements of current workflow research. More workflow systems have been developed to employ web data and web services with limited success to address the issue of integration and automation. Since bioinformatics is a major application area of contemporary scientific workflow systems, bioinformatics workflow systems and processes will be used as the major research objects in the rest of this chapter.

Some data integration and mediation systems have already been developed to facilitate the knowledge sharing among biological data sources and experience exploitation among biologists. One of these systems in this direction is Transparent Access to Multiple Bioinformatics Information Sources project (TAMBIS) (Goble et al., 2001). This system applies domain ontology for molecular biology and bioinformatics on the retrieval-based information integration system for biologists. Most of these projects focus on accessing and integrating data from remote sites based on some kind of ontology of the data. Currently, these systems face two major deficiencies.

First, integrating any remote data needs to register complex information and manually tune the system, both of which need domain-specific knowledge related to data integration. According to the OIL system described in (Stevens, Goble, Horrocks, & Bechhofer, 2002), three aspects of information (that is, metadata, terminologies, and ontology) are required to make an understanding of the data between machines and biologists. To the author's knowledge, not too much, if any, biological resources on the Internet provide all of these three kinds of information until now. Due to the large amount of data used in biological analysis, one cannot expect the biologists to provide ontology information about the data they utilized, not to mention the arbitrary information sources on the Internet. It is the author's contention that ad hoc querying involving arbitrary sets of resources is almost impossible using current technologies. Yet, a compact, declarative, but powerful resource registration schema is needed for resource integration.

Second, most of these systems focus only on data integration. The integration of remote programs and online analysis tools is not emphasized. To use the tools on a remote site, these systems force the code migration, instead of data migration, from remote site to local. In (Chen & Jamil, 2003), Chen *et al.* argued that in most cases the code migration is expensive, if not impossible. A few projects try to apply ontology and semantic web technologies, which is successful for data integration, onto the issue of program integration with limited success. Obviously the ontology for data integration is dramatically different with the ontology for data/program interaction. For data integration, ontology information is used to reconcile the difference among heterogeneous data sources and build up a universal view of the data. But, for data and program interaction, ontology extracted from the data is used to cater to the ontology of the program and submit the data to the program. This difference makes it necessary to build up a sophisticated matching mechanism for online resource integration.

This chapter focuses on aspects of online resource registration and online data/program integration in Grid-Flow. A compact and powerful mechanism to handle the heterogeneity of online resources is presented with its application on an illustrative example. This mechanism not only facilitates the design and execution of processes on data analysis, but also strengthens the extensibility and usability of the system.

6.1 Related Work

Modern scientific workflow systems need to support ad hoc integration of arbitrary distributed and heterogeneous data sources and online programs. An object-based information exchange model is defined in (Papakonstantinou, Garcia-Molina, & Widom, 1995) to handle the integration of diverse information sources. This model uses a laconic expression to represent an object in the resources, which motivates compact design offered here of the data item description in the *Data Description File*. Several data integration projects using ontology information have developed resource description languages to specify the resources used in data integration and analysis, such as OIL (Stevens et al., 2002) and Context Specification Language (CSL) (Gupta, Ludascher, & Martone, 2002).

Recently, workflow technology has been exploited into biological processes to support the heterogeneous and distributed environment. IntelliGEN (Kochut et al., 2002) is developed to map protein-protein interactions of fungi for a large genomic project. While the Remote User Defined Function (RUDF) in (Chen & Jamil, 2003) exploits the advantages of Internet Functions and HTQL in order to provide the local database a unique calling mechanism for both local programs and programs on the remote sites. However, a substantial amount of coding is still needed in such systems to cope with change, especially when new resources need to be incorporated into the system.

A research prototype studied in (Cardoso & Sheth, 2003) addresses two problems about program integration. One is how to efficiently discover a web service. The other is how to facilitate the interoperability of heterogeneous web services. This workflow prototype employs semantic web and ontology technology to handle the heterogeneity of web services. These technologies facilitate the automation of web service integration. However, the ontology of web services cannot be generated automatically, which damages the autonomy of the whole workflow system. The offered matching mechanism is enlightened by the matching evaluation function for service template and service object described in (Cardoso & Sheth, 2003).

6.2 Registration of Data/Program

This section begins to present a brief discussion of the online data and program integration in Grid-Flow. Data and program registration is introduced first.

The major purpose for registration of data/program is to record necessary information of the data/program used in workflow processes. Data and program registration usually involves the recording of three related structures: Data Record (DR), Program

Record (PR), and Data Description File (DDF). In this section, each of these structures is formally described.

6.2.1 Data Record

DR is used to store fundamental meta-information of the data used in the workflow process. A DR is formally defined as a five tuple, which is shown in Figure 27.

DR(d) = <ID, Label, Type, Source, DDF>

Figure 27. DR Definition.

Where *ID*, *Label*, *Type*, *Source*, and *DDF* are the identification number, name, data format, location, and data description file of the data record *d*, respectively. The *ID* is a token for the data which is unique in the universal Grid-Flow system. In denoting a DR in this dissertation, *ID* will often be dropped (that is, one writes <Label, Type, Source, DDF> instead) since *ID* field is automatically generated by Grid-Flow. The *Label* field is the name of data; note it need not to be unique in the system. *Type* field indicates the format of the data, such as a relational database table (*TABLE*), a HTML file (*HTML*), or a plain text file (*TXT*). The *Source* field is mainly concerned about where to get the data. If the *Type* of the data is *TABLE*, the *Source* stores the connection method of the corresponding database and the table name. If the *Type* of the data is a structured or semi/unstructured file, the *Source* field is a Uniform Resource Locator (URL) (*Naming and Addressing: URIs, URLs, ...*) address of the data. The *DDF* field stores a link to the corresponding Data Description File of the data, which is described in section 6.2.3.

For instance, in the illustrative workflow (described in section 3.5) “prediction of transmembrane regions,” the task *TMPred* is an Internet Program, which has a webpage interface (shown in Figure 30) where one can set the parameters and submit the sequence. This webpage is data in the process definition and has the DR as shown in Figure 28.

```
DR(TMPredInput) = <"TMPredInput", "HTML",
"http://www.ch.embnet.org/software/TMPRED_form.html",
"File:///c:\BioFlow\DDF\TMPredInput.ddf">
```

Figure 28. DR for Data TMPredInput.

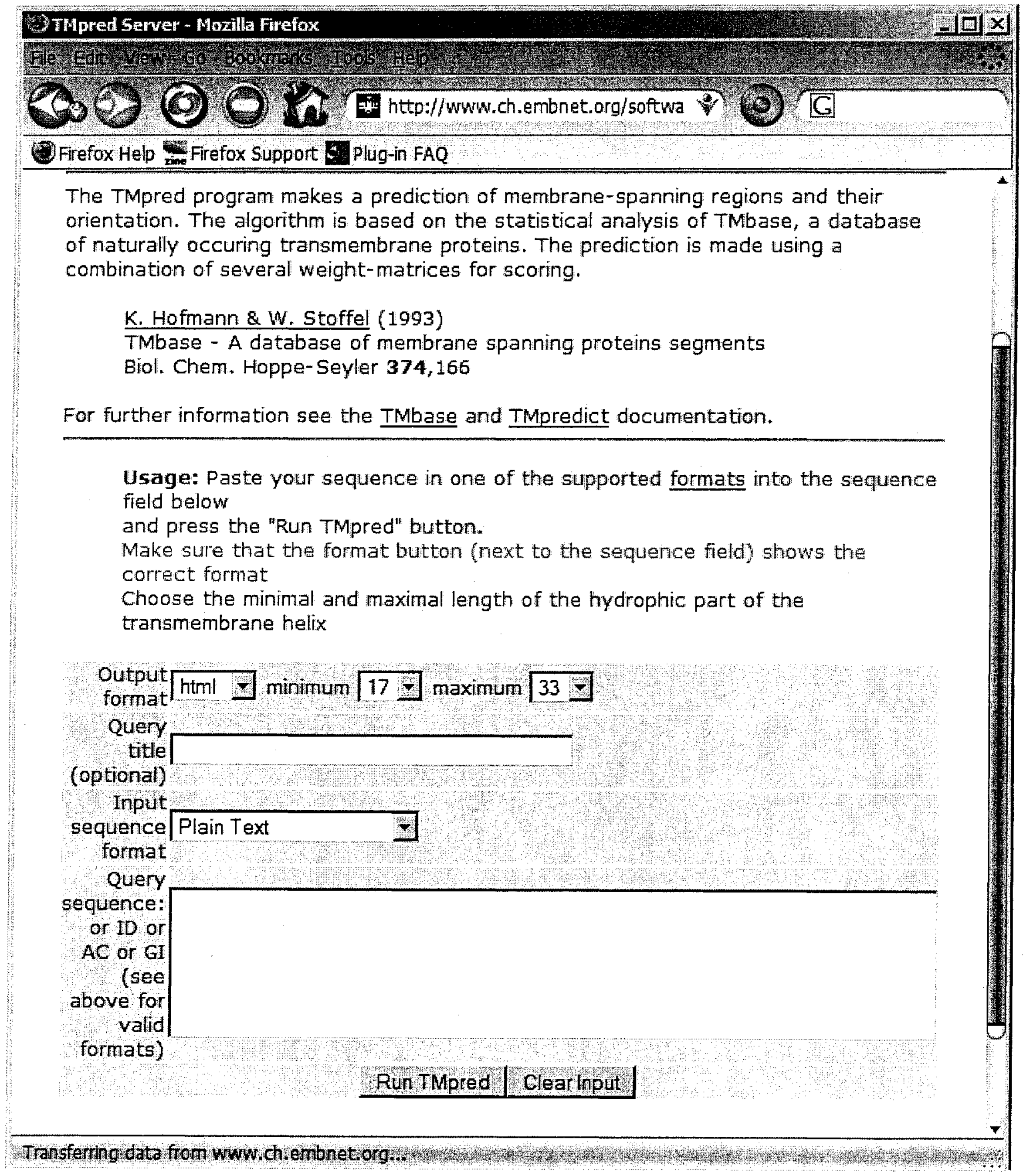
6.2.2 *Program Record*

PR is a structure that records the related ad hoc information of a program. A PR is specified as shown in Figure 29.

```
PR(p) = <ID, Label, Type, Source, InputSet, OutputSet>
```

Figure 29. PR Definition.

The fields *ID*, *Label*, *Type*, and *Source* have similar meanings for program *p* as described above for DR. The fields *InputSet* and *OutputSet* are two sets of DRs, which respectively describe the input data record set and output data record set for the program *p*. For instance, let one assume to have already defined the output data record for the Internet program *TMPred*, which is *TransHelix*. Then, one can define the PR of program *TMPred* as shown in Figure 31.


TMpred Server - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.ch.embnet.org/softwa
Firefox Help Firefox Support Plug-in FAQ

The TMpred program makes a prediction of membrane-spanning regions and their orientation. The algorithm is based on the statistical analysis of TMbase, a database of naturally occuring transmembrane proteins. The prediction is made using a combination of several weight-matrices for scoring.

K. Hofmann & W. Stoffel (1993)
TMbase - A database of membrane spanning proteins segments
Biol. Chem. Hoppe-Seyler **374**,166

For further information see the TMbase and TMpredict documentation.

Usage: Paste your sequence in one of the supported formats into the sequence field below
and press the "Run TMpred" button.
Make sure that the format button (next to the sequence field) shows the correct format
Choose the minimal and maximal length of the hydrophic part of the transmembrane helix

Output format: html minimum 17 maximum 33
Query title (optional)
Input sequence format: Plain Text
Query sequence: or ID or AC or GI (see above for valid formats)

Figure 30. Input Web Interface for TMPred (*TMpred - Prediction of Transmembrane Regions and Orientation*).

PR(TMPred) = <"TMPred", "OS_Program", "http://www.ch.embnet.org/cgi-bin/TMPRED_form_parser", "TMPredInput", "TransHelix">

Figure 31. PR for Program TMPred.

6.2.3 Data Description File

In Grid-Flow, data item is defined as the meaningful segment of data which is operable by the Grid-Flow language. For example, in a webpage about protein sequence, the locus, definition, features and sequence of the protein are all data items. It is actually the data items, not the data, which can be extracted, transferred, transformed, and operated in a workflow process. One data could contain more than one data item. Thus, the registration information of data is not enough for one to understand the details of the data items contained in the data. So the DDF was designed to describe the inner details of the registered data. A Data Description File is composed with a set of data item records (DIRs). DIR is a four tuple with the format shown in Figure 32.

DIR(i) = <Name, Format, Keywords, Access>

Figure 32. DIR Definition.

In the DDF of one certain data, each data item has a unique *Name* to identify itself from the other data items. The *Format* depicts the format of the value of the data item. In

Grid-Flow, the formats of data for bioinformatics research are grouped into three primary categories, which are *Basic* type, *Sequence* type, and *Tree* type, as shown in Figure 33. Each primary category can be subdivided into several sub categories, for instance, *Basic* type can be refined as *String* type, *Number* type, *Date* type, and *Boolean* type. In this hierarchical format system, subdividable types are denoted conceptual types. Contrarily, types without any sub-types are called primitive types. The *Format* value of a data item must be one primitive type in the hierarchy because any data item has only one predefined format in a real-world data source definition. The advantages of this hierarchical structure are demonstrated in section 6.3.

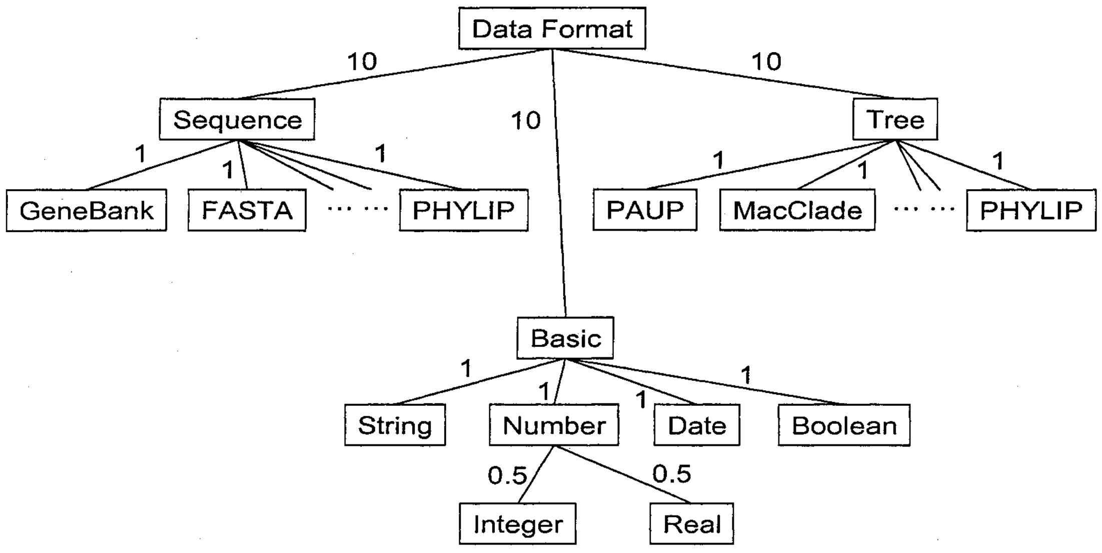


Figure 33. Hierarchy of Format Categories for Bioinformatics Workflows.

The field *Keywords* is a set of words that are used to give a description of the data item. The *Keywords* field plays a great role during the matching of the output and input data items, which will be explained in section 6.3. *Access* field depicts the access method of the data item. The content of *Access* field varies according to the different data sources. For a data item in a relational database, *Access* field stores a SQL sentence which could generate the data item as the query result, while the *Access* field of a data item in a semi/un-structured file would be a wrapper that can extract or access that corresponding data item. In Grid-Flow, the semi-structured data, which is specialized as HTML files, can be categorized into two types according to various services they can provide. One kind of semi-structured data can only provide information, but not accept information, such as most common web pages with textual content. The other kind of semi-structured data, such as the interfaces of Internet analysis tools, can not only provide information, but also accept information from the end-users or other programs. These two kinds of data are called information-based and query-based semi-structured data, respectively. Wrappers stored in the *Access* field have different formats for these two kinds of semi-structured data. The wrapper for a data item in an information-based webpage is a HTQL (Chen & Jamil, 2003) sentence, which is used to extract the data from the webpage. The wrapper for a data item in a query-based webpage is the name of that data item in the query form, which is used to construct the query when the Internet program is called.

Figure 34 shows part of the *DDF* (whose name is *TMPredInput.ddf)* of the data *TMPredInput* (the interface webpage is shown in Figure 30). Note this part of the *DDF* describes those input boxes of the query form in the interface webpage.

```
… …
<OutputFormat, String, "output, format", outmode>
<Minimum, Integer, "minimum, length, helix", min>
<Maximum, Integer, "maximum, length, helix", max>
… …
<QuerySeq, Sequence.plain, "query, protein, sequence", Seq>
```

Figure 34. Part of the DDF for TMPredInput.

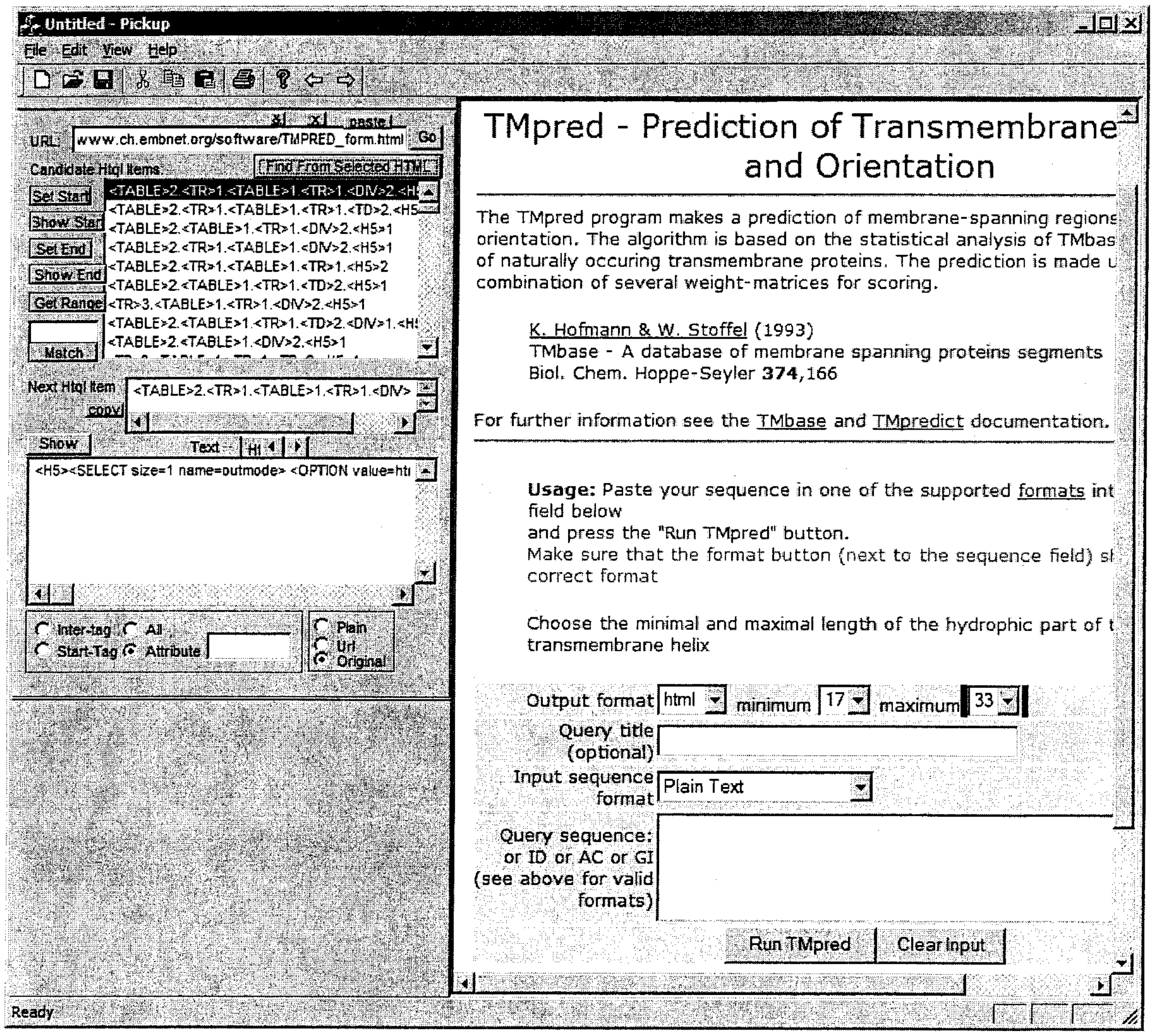
Since the data item is the meaningful segment of the data, it can have its own hierarchical structure. For example, a user can define a publication of a certain protein sequence as a data item *publication*. Meanwhile, the user can also define the author, title, and journal name of that publication as three different data items: *author*, *title*, and *journal*, respectively. Thus, data item *publication* is composed of three small data items. This hierarchical structure cannot be presented in the *DDF*, since the *DDF* is only an unordered set of *DIRs* without any expression ability for the relationships between *DIRs*. However, it will be proved in the next section that this deficiency does not harm the roles that the DDF plays in the streamlining analysis.

The next issue about DDF is how to generate it. To generate a DDF, the Grid-Flow users need to provide accurate information for each data item in the data. This information includes the name, format, keywords of the data item and the wrapper to access the data item. Grid-Flow provides users an integrated environment to generate the DDF of the data with limited interaction between user and the data. To generate a DIR with this integrated environment, a user first generates the wrapper of that data item with a semi-automatic tool, PickUp (Chen & Jamil, 2003), then the user need to input name, keywords of that data item, and finally select the data format for that data item. The interface of software PickUp (Chen & Jamil, 2003) is shown in Figure 35.

6.3 Matching Output and Input

The streamlining analysis of biological data is carried out by a key operation: the matching function. Suppose Grid-Flow needs to call the ongoing program *P* and feed *P* with a set of data $\{OD_1, OD_2, ..., OD_n\}$ which have the corresponding DDFs: $O_1, O_2, ..., O_n$. The program *P* has a set of predefined input data set $\{ID_1, ID_2, ..., ID_m\}$ which have the corresponding DDFs: $I_1, I_2, ..., I_m$. Grid-Flow first takes the union of all of the DDFs $O_1, O_2, ..., O_n$ together and names it as DDF *O*. All of the DIRs in DDF *O* are called output DIRs. Similarly Grid-Flow takes the union of DDFs $I_1, I_2, ..., I_m$ together and names it as DDF *I*. Thus, all of the DIRs in DDF *I* are called input DIRs. The matching function is dedicated to find correspondences between an output DIR *O* and an input DIR *I*.

During the matching phase, the matching function is employed to match any output DIR against any input DIR and rank the output-input DIRs pair according to the similarity and compatibility. All of the output-input DIRs pairs are finally sorted according to their ranks. The user can select the best matched output-input DIRs pairs as the match result, or manually solve the output-input differences that are not solved by the Grid-Flow system. A matching mechanism is implemented in Grid-Flow to realize the above function. Given a set of output DIRs and a set of input DIRs, this mechanism examines the DIRs and tries to find similarities and compatibilities between any output-input DIRs pair. This is done by using syntactic information matching and data transformation evaluation. The syntactic matching is based on DIR's *Keywords* field, and the transformation evaluation is based on DIR's *Format* field.


Untitled - Pickup

URL: www.ch.embnet.org/software/TMPRED_form.html

Candidate Htql items:

```
<TABLE>2.<TR>1.<TABLE>1.<TR>1.<DIV>2.<H
<TABLE>2.<TR>1.<TABLE>1.<TR>1.<TD>2.<H5
<TABLE>2.<TABLE>1.<TR>1.<DIV>2.<H5>1
<TABLE>2.<TR>1.<TABLE>1.<DIV>2.<H5>1
<TABLE>2.<TR>1.<TABLE>1.<TR>1.<H5>2
<TABLE>2.<TABLE>1.<TR>1.<TD>2.<H5>1
<TR>3.<TABLE>1.<TR>1.<DIV>2.<H5>1
<TABLE>2.<TABLE>1.<TR>1.<TD>2.<DIV>1.<H
<TABLE>2.<TABLE>1.<DIV>2.<H5>1
```

Next Htql item

```
<TABLE>2.<TR>1.<TABLE>1.<TR>1.<DIV>
```

```
<H5><SELECT size=1 name=outmode> <OPTION value=htı
```

Inter-tag All Start-Tag [✓ Attribute] Plain Url [✓ Original]

Ready

TMpred - Prediction of Transmembrane and Orientation

The TMpred program makes a prediction of membrane-spanning regions orientation. The algorithm is based on the statistical analysis of TMbas of naturally occuring transmembrane proteins. The prediction is made u combination of several weight-matrices for scoring.

K. Hofmann & W. Stoffel (1993)
TMbase - A database of membrane spanning proteins segments
Biol. Chem. Hoppe-Seyler **374**,166

For further information see the TMbase and TMpredict documentation.

Usage: Paste your sequence in one of the supported formats int field below
and press the "Run TMpred" button.
Make sure that the format button (next to the sequence field) sł correct format

Choose the minimal and maximal length of the hydrophic part of t transmembrane helix

Output format: html minimum: 17 maximum: 33

Query title (optional):

Input sequence format: Plain Text

Query sequence: or ID or AC or GI (see above for valid formats):

Figure 35. Interface of PickUp (Chen & Jamil, 2003).

The similarity and compatibility of an output DIR (o) and an input DIR (i) is calculated with the function *Match(o,i)* presented as Equation 1. The similarity and compatibility computation relies on functions *Transform(o.Format, i.Format)* and *KeyMatch(o.Keywords, i.Keywords)*, and the weights ω_1 and ω_2. Function *Transform(o.Format, i.Format)* evaluates the compatibility between the output data item and input data item. Function *KeyMatch(o.Keywords, i.Keywords)* takes into account the functional similarities between the output data item and input data item. Both of these two functions return a real value between 0 and 1, for the sake of comparison based on the same scale. The weights ω_1 and ω_2 are real values between 0 and 1. They indicate the ratio of importance that the designer considers for the data format transformation and the functionality match. The sum of the weights should be 1 to keep a reasonable ratio.

$$\begin{aligned} Match(o,i) = {} & \omega_1 \times Transform(o.Format, i.Format) \\ & + \omega_2 \times KeyMatch(o.Keywords, i.Keywords) \end{aligned}$$

Equation 1. Matching Function.

Function *Transform(o.Format, i.Format)* evaluates the compatibility between the formats of output data item and input data item according to the length of the path in the format category tree (Figure 33). The length of the branch in the format category tree indicates the cost of transformation from one data type on one end of the branch to the other data type on the other end of the branch. For example, in Figure 33, the length of path from *integer* to *real* is 1 (0.5+0.5), which means the transformation between them is easy. On the contrary, the length of path between *Sequence.PHYLIP* and *real* is 22.5

(1+10+10+1+0.5), which means the cost of transformation is so high that makes it hardly possible. Equation 2 is applied in order to calculate function *Transform(f_1, f_2)*. This equation returns a value between 0 and 1; the larger the return value, the more compatible these two formats.

$$Transform(f_1, f_2) = 1 - \frac{Length(f_1, f_2)}{MaxLen(Tree)}$$

Equation 2. Transform Function.

In Equation 2, *Length(f_1,f_2)* returns the length of the path between format type f_1 and f_2, and *MaxLen(Tree)* returns the length of the longest path in the format category tree.

Function *KeyMatch(o.Keywords, i.Keywords)* depicts the functional similarity between the *Keywords* fields of the two data items. Since the *Keywords* field is a set of keywords, one uses string-matching as a way to calculate how closely these two data items resemble each other. To achieve a better comparison between two *Keywords* fields, the keywords are preprocessed. The preprocessing includes removing prefixes and suffixes of the keywords, and equalizing the synonyms. For instance, after removing suffixes, "analysis" matches to "analyze." Similarly, "phylogeny" matches to "tree of life" since they have the same meaning. Suppose those two *Keywords* fields k_1, k_2 contain kn_1 and kn_2 keywords, respectively, Equation 3 can be used to calculate the function *KeyMatch(k_1, k_2)*.

$$KeyMatch(k_1,k_2) = \frac{StringMatch(k_1,k_2)}{Min(kn_1,kn_2)}$$

Equation 3. KeyMatch Function.

In Equation 3, *StringMatch(k_1,k_2)* calculates the number of string match and *Min(kn_1,kn_2)* returns the minimum number between kn_1 and kn_2.

The equation makes sure the return value is between 0 and 1. The higher value of this equation indicates the compared two data items are more similar.

From the analysis above, one can see if two data items match well, the return value of *Transform(o.Format, i.Format)* and *KeyMatch(o.Keywords, i.Keywords)* must be high. Thus, the value of *Match(o,i)* must be high. The perfect match generates high evaluation value. On the other hand, if *Match(o,i)* returns a high value, it means either *Transform(o.Format, i.Format)* or *KeyMatch(o.Keywords, i.Keywords)* or both of them return high values. This means the functionalities of the compared two data items are similar or the data formats are compatible. Both are prerequisites of the perfect matches. So, one can draw the conclusion that the function *Match(o,i)* is an effective evaluation function for data and program matching.

Figure 36 is a data/program matching example corresponding to the examples presented in section 3.5. In this example, the hexagons represent registered programs, while the rectangles indicate the DDF of each data source. Suppose that the execution of program *NCBISearch* has finished and the execution of next program *TMPred* is to begin. Before the program *TMPred* is called, Grid-Flow must satisfy the input requirements of *TMPred* with two data: "*Protein Sequence*" and "*TMPred Parameters*." "*Protein Se-*

quence" is generated by *NCBISearch* and contains the protein sequence that needs to be analyzed. "*TMPred Parameters*" is provided by the user and contains the necessary program parameters for *TMPred.* So the best matches should be matching "*Origin*" in "*Protein Sequence*" to "*QuerySequence*" in "*TMPredInput.*" Meanwhile, "*Minimum*" and "*Maximum*" in "*TMPred Parameters*" should be matched to "*Minimum*" and "*Maximum*" in "*TMPredInput*" respectively.

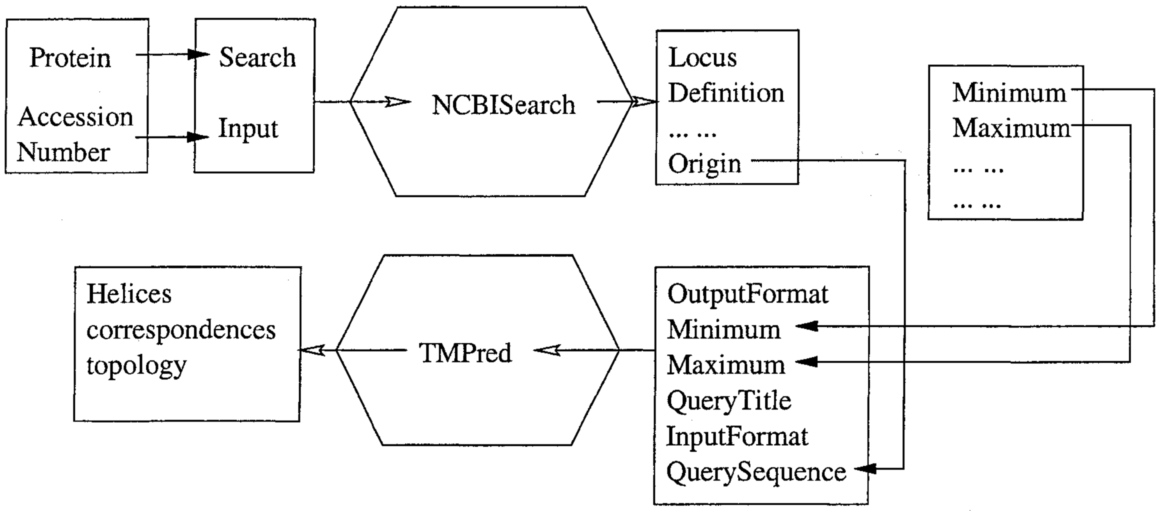


Figure 36. Matching Data for Predication of Transmembrane Regions.

The matching step is the next thing to be accomplished, just before the program *TMPred* can be invoked by the Grid-Flow engine. First, Grid-Flow need to combine those two DDFs together for the output data "*Protein Sequence*" and "*TMPred Parameters,*" since more than one output data is obtained and there is a desire to match these data to one input data "*TMPredInput.*" To match the output-input pairs, Grid-Flow can calculate the *Match(o,i)* value for each output-input data item pair *(o,i)* and sort them according to this value. Table 1 shows the matching results under the parameter ω_1=0.2 and ω_2=0.8.

Table 1. Matching results for TMPred.

Output DIR	Input DIR	*Match(o,i)*
Origin	QuerySequence	1.0
Minimum	Minimum	1.0
Maximum	Maximum	1.0
... ...	... ...	...
Locus	QuerySequence	0.54
... ...	... ...	...

From the results, one can safely select the first three output-input data item pairs as the perfect matches. Thus, the input requirements of *TMPred* can be satisfied and the whole workflow can progress.

6.4 Data Transformation

Grid-Flow makes a best effort to alleviate the burden of data format transformation by providing a group of automatic transformation programs. The data formats used in bioinformatics workflows are shown in Figure 33. Within any major format category, most of the sub-types can be automatically transformed into other sub-types by using some Grid-Flow internal program. For example, in the major category *Basic*, Grid-Flow provides programs *StrToInt()*, *IntToStr()*, and *StrToDate()* to carry out the automatic transformation from *String* to *Integer*, from *Integer* to *String*, and from *String* to *Date*, respectively. In the major category *Sequence*, Grid-Flow employs a tool named Readseq (*Readseq: Read and reformat biosequences*) to handle the data transformation issue. With the help of these programs and tools, most of the data transformation can be handled by

Grid-Flow without users' interference. This feature, accompanied with the output/input matching mechanism, greatly reduces the labor of designing and executing bioinformatics workflows with Grid-Flow.

6.5 Summary

In this chapter, the Grid-Flow system was investigated regarding the aspects of online resource registration and online data/program integration. A methodology for recording and representing data/program meta-data in Grid-Flow was presented, accompanied with the output/input data matching mechanism. The data matching mechanism is an important enabling capability for streamlining the data processes with incompatible program input/output interfaces. With the development of web and Grid computing technologies, more web servers provide their data and computing resources in the format of web/Grid services. Compared with online programs, web/Grid services provide users a unique, stable, secure, and program accessible way to access remote resources. Then next chapter will examine the benefits of using web/Grid services and Grid-Flow's ability to utilize web/Grid services for scientific workflows.

7 GRID-SERVICE-BASED PROGRAM DEPLOYMENT

The online resource registration and online data/program integration discussed in Chapter 6 enable Grid-Flow to utilize physically distributed tools for heterogeneous data analyses through a web-based accessing interface. The online data and programs, however, only account for part of the resources that are scheduled by Grid-Flow for running real world workflow applications. Grid-Flow also needs to integrate large amounts of data and programs which lack a web-based interface for public access. These data and programs could locate in the local computer as well as in local and/or wide area networks. Choosing ways to effectively integrate these distributed resources is an active research topic in the workflow community.

In this chapter, programs invoked by Grid-Flow are first categorized according to their location and user interface. Then methods to access these programs are provided for each category. The benefits of using Grid services for remote non-interactive programs in scientific workflow systems are discussed, followed by a demonstrative example of wrapping a popular bioinformatics application BLAST (Altschul, Gish, Miller, Myers, & Lipman, 1990) into a Grid service, and integrating this Grid service as a transition in Grid-Flow.

7.1 Categories of OS Programs in Grid-Flow

As described in section 3.4.2, all the external programs invoked by Grid-Flow are denoted OS programs, as opposed to the internal programs and Grid-Flow programs that are directly executed by the Grid-Flow engine. The program integration component in Grid-Flow is responsible for integrating these external programs and providing the Grid-Flow engine a unified platform to invoke them. In this section, the external programs are first categorized into four classes. Then different invoking strategies used to call programs in different classes are discussed in detail.

The external programs invoked by Grid-Flow could be categorized according to their location and the type of their user interface. For the criteria of program location, programs could be classified as 'local' programs and 'remote' programs. Here local programs are defined as programs that are located in the same operating system with Grid-Flow and can be invoked via system calls. Remote programs, as opposed to local programs, are programs that are located out of the scope of the operating system where Grid-Flow resides. Usually, the programs that are in local operating system but cannot be invoked by system calls are not considered since Grid-Flow cannot utilize them in any way. Similarly, programs cannot be invoked via system calls if they are not in the local operating system. Thus Grid-Flow is safe to distinguish the local and remote programs by estimating whether the program can be invoked with system calls. For user interface, a program could be interactive, which means user inputs are required during the program's execution, or non-interactive, which means users should set all the parameters when invoking the program and the program does not need any further inputs during its execution. As shown in Figure 37, with the two criteria of location and user interface, programs used in WFMS can be categorized into four classes: local interactive, local non-interactive, remote interactive, and remote non-interactive programs.

Location / User Interface	Local	Remote
Interactive	Local interactive	Remote Interactive
Non-interactive	Local non-interactive	Remote non-interactive

Figure 37. Categories of Programs in WFMS.

Local programs can be handled easily by most of the current WFMSs with system calls. The operating system where the WFMS is based on provides abundant system calls for local program invocation and process management. Correspondingly, most of the programming languages provide interfaces for applications to perform system calls in order to utilize local programs. Grid-Flow, similar to all the other scientific WFMSs, uses the interface provided by the programming language to invoke local programs. Particularly, since Microsoft Visual Basic is used to implement the Grid-Flow engine, Grid-Flow uses Visual Basic system calls (Shell Function (Microsoft)) to invoke programs in the local operating system. For instance, in order to invoke the program Notepad in Microsoft Windows system for displaying a text file, the code demonstrated in Figure 38 is embedded in the program integration component.

```
filePath = App.Path & fileName
Dim retValue As Variant
retValue = Shell("c:\windows\system32\notepad.exe " & filePath, vbMaximizedFocus)
```

Figure 38. System Call to Invoke Notepad in Grid-Flow Engine.

Although both local interactive programs and local non-interactive programs can be invoked by system calls, Grid-Flow treats them differently on the aspect of process management. When Grid-Flow engine calls a non-interactive program, the system call would not return until the non-interactive program finishes its execution. Thus the Grid-Flow engine remains halted and cannot process any other workflow tasks. However, for an interactive program, the system call returns immediately after the program starts up. The Grid-Flow engine can gain the control of the workflow without waiting till the program execution ends. The benefit brought out by this immediate return mechanism is that Grid-Flow engine can execute multiple tasks at the same time, which in turn improve the performance and throughput of the whole workflow system. The pitfall of this mechanism, however, is that the Grid-Flow engine cannot be informed when the program stops. Owing to lack of time information on program termination, the Grid-Flow engine cannot precisely schedule tasks that are time dependent on other tasks/programs. That is, if one task A in the workflow can only be started after another program B's fully termination, the Grid-Flow engine may not be able to know when to start task A since it cannot get notification about the accomplishment of program B. For instance, if a workflow task (A) is designed to process the data users inputted through an anterior local interactive program (B), it should not start until the user inputs all the data. The Grid-Flow engine should only schedule the start of the data processing task (A) after the finish of the data input task (B). If the Grid-Flow engine cannot trace the whole data input task, it may improperly invoke the data processing task so early that users have not inputted the data. This may cause the whole workflow crash.

To overcome this drawback, Grid-Flow employs the function of process management provided by the Microsoft Visual Basic (Microsoft) to monitor the whole lifecycle of the local program execution. Figure 39 shows the code of process management for local interactive programs in Grid-Flow. With the process management, each execution of an interactive program is monitored as a process. The Grid-Flow engine starts the process by feeding the program name and necessary inputs to the *Shell* function, and then calling *OpenProcess*. The process management afterwards takes over the control and checks the exit code of the process periodically till the process is inactive. Once the process terminates, the process management hands the control back to the Grid-Flow engine. Thus the Grid-Flow engine is informed of the time of the event of execution termination. It can then perform the post-execution processing and schedule other tasks which are time and/or data dependent on this terminated program.

```
Dim pid As Integer
Dim hProcess As Long
Dim RetVal As Long

hProcess = OpenProcess(PROCESS_QUERY_INFORMATION, False,
Shell(commandString, vbMinimizedNoFocus))
Call GetExitCodeProcess(hProcess, RetVal)
Do While RetVal = STILL_ACTIVE
        pid = DoEvents()
        Call GetExitCodeProcess(hProcess, RetVal)
Loop
```

Figure 39. Code for Process Management of Interactive Programs.

Although the system calls are broadly used to call local programs in current WFMSs, they cannot be applied to the invocation of remote programs because the system calls cannot act beyond the local system boundary to reach remote systems. The program integration component in Grid-Flow provides different strategies to invoke remote non-interactive programs according to the availability of their accessing methods: for programs that are wrapped as CGI programs, the Grid-Flow engine uses the techniques described in Chapter 6 to integrate the program's web interface; for programs that are available through web/Grid services, the program integration component has a programming interface to contact those services and utilize the functions provided by the programs; for programs that are only available on remote site without any web/programming interfaces, the program integration component also provides predefined functions to help users manually login into the remote system, run the program, and retrieve back the output results. The next section compares these three strategies and analyzes the disadvantages of calling remote programs with CGI programs. Grid-Flow prefers web/Grid services to other strategies for accessing remote non-interactive programs.

Invoking remote interactive programs is more complex compared to interactions with remote non-interactive programs. The system needs to not only invoke the program properly, but also to transfer and transform the interaction between the remote program and the end users. Current scientific WFMSs do not have any approach available to transfer users' input directly to the remote program during execution. Grid-Flow does not support invoking remote interactive programs either. However, a potential approach to enable the interaction between the user and the remote programs is to combine a desktop sharing system (like Virtual Network Computing (VNC) (Wikipedia)) with the traditional remote program calls. Thus a remote program could be invoked by remote program calls and displayed on the virtual desktop simulated by the desktop sharing system. The user could then interact with the program process running on the remote site. The desktop sharing system handles all of the communication between remote systems and the user during the execution. When the remote program terminates, it would return the control back to the Grid-Flow engine and automatically turn off the virtual desktop.

7.2 Grid Services in Grid-Flow

Most non-interactive programs could be invoked by calling an executable, followed by optional or mandatory parameters and input data that influence the execution of the program. WFMS can invoke programs in the local operating system using system calls. To run remote non-interactive programs in the network environment, WFMS needs to either login into the remote system for direct access, or wrap the programs with wrappers for indirect invocation. Conventional WFMSs usually use the mechanism of remote login to access programs. That is, WFMSs are delegated by the user to stage input data onto the target system, remotely log into the system, invoke the program, check the status of the execution periodically, wait until the execution finishes, and fetch the output data back to the local computer. This ad hoc remote login method not only introduces excessive coupling between WFMS and target systems, but also brings potential threats to the security mechanism of remote systems. Facing these drawbacks, most of the advanced scientific WFMSs adopt various kinds of wrappers or adapters to access remote non-interactive programs, without directly login into the remote system. Wrappers of remote programs, with well designed interfaces, could provide a unique, compact, and secure way to access data and programs on the remote site.

The most commonly used wrappers in WFMSs are CGI programs and web/Grid services. Grid-Flow supports both of these two wrappers for remote program accessing. Integrating CGI programs as Grid-Flow functional units has been discussed in Chapter 6.

Compared with web/Grid services, using online CGI programs as the method to access remote data and programs has the following disadvantages:

1. Not all authors prefer to publish their data/programs online. Some authors would rather like to only reveal their private data/programs to limited authorized access. Currently CGI programs are able to provide a login mechanism for user authentication and authorization. This login mechanism is dependent on the website that contains the corresponding CGI programs. For any workflows that need to access multiple online programs, the user must manually login into all the relevant websites and remember all the associated usernames/passwords. In addition, the login process is usually integrated in the website. It is hard, if not impossible to model each login process in workflow system without revealing all the inner details. On the other hand, from the perspective of WFMS, this login mechanism not only makes the workflow engine more complicated in architecture and functionality to implement, but also weakens the security strength of the whole architecture. These integrated login methods are hard-coded within the workflow engine, making it difficult for the workflow system to accommodate future changes. A single sign-on system would greatly facilitate the user to access multiple password protected CGI programs within a workflow. Grid Security Infrastructure (GSI) provides such a single sign-on system for the user to login once and automatically communicate with all the services for the issue of user authentications.
2. Transferring the input data is still a problem for CGI programs. CGI programs do not have a standard approach to transfer the input data from the client side to the server. Some CGI programs ask users to cut-and-paste the content of the input files into the

webpage interface, which in turn transfers the content as a string stream. Some other CGI programs ask users to upload input files to a specific directory on the remote site, and then read those input files locally on the server side. The channel used to transfer the data could be either non-secured or secured, depending on whether the CGI program supports the secured communication. To integrate CGI programs, Grid-Flow needs to model the data staging from the client to the server. Since there is no standard way defined for CGI programs, Grid-Flow can only handle the data transfer case by case. This ad hoc manner damages the generalizability and extensibility of the workflow system. Thus a mechanism to transfer large amount of input data to programs on remote sites is required for scientific workflow system. GridFTP (*The GridFTP Protocol and Software*) is over time becoming the standard and secure approach for transferring data among Grid nodes. Grid-Flow has integrated GridFTP into the Grid services used for workflow composition.

3. CGI programs are designed for human interaction, not for program-level interaction. Both the input interface and the output result of CGI programs are displayed in web pages with HTML format. HTML format is designed to convey information over Internet, as well as to control the display format of the information. The information, format controlling tags, and functional scripts are mixed and tangled together in HTML files. Though flexible, the semi-structured, entangled format impacts the development of efficient HTML parsers for information query and retrieval. The lack of efficient parsers further encumbers WFMSs to integrate CGI programs as functional units because it is hard to flow the data through CGI programs without the support of parsers. Workflow engines prefer a program accessible interface through which they

can feed the input data into the CGI program, and retrieve the information out of the output result. Web/Grid services defined exactly the programmable interfaces required by workflow engines.

4. CGI programs are ever-changing. Compared with parameter settings and the function, the input/output web interface of a CGI program are prone to changes since they need to satisfy users' ever-changing requirements. Some changes of the interfaces may not affect the human interaction, but they could dramatically mess workflows that use the CGI program. For example, changing the position of one text box in the input interface may not even noticeable to the user's eyes. But the workflow could fail to feed parameters to that text box if the parser of the input interface cannot adapt itself to this change. The lack of adaptability and robustness of CGI parsers essentially makes the workflow fragile and sensitive to changes. On the other hand, there exists no notification mechanism for the changes of the CGI programs. Thus users can only passively change the parsers of CGI programs when they find the workflow fails to execute the CGI programs. Chen *et al* (Chen & Jamil, 2003) provide an semi-automatic way to wrap the CGI programs with interface parsers. But even with regeneration of the parsers, it is still hard to keep up with the changes of the CGI programs. Web/Grid services provide a program-accessible interface, which could be easily integrated into the WFMS engine. This interface is an agreement between the service provider and the end user. So it strictly follows its definition and seldom changes once the web/Grid service is published. WFMS always prefer a stable, robust, and uniform interface to access the data and programs.

5. Adding a CGI program into WFMS needs more efforts than integrating a web/Grid service. To access the CGI program, the parsers for input/output interfaces are required to be registered with WFMS. The workflow engine then needs to configure itself for connecting the CGI program for submitting jobs. With new CGI programs and more computing resources becoming available for executing workflow tasks, the workflow engine needs to make changes frequently. On the contrary, the component to access web/Grid services has already been integrated in most of the advanced WFMS engines. WFMS engines can automatically read the definition of web/Grid services and integrate them for workflow execution. No more work needs to be done for adding a new web/Grid service.

Similarly to many advanced academic WFMSs, Grid-Flow employs Grid computing technique as its method to transfer data and invoke programs through web/Grid services. In Grid-Flow, Grid authentication mechanism (Foster et al., 2002) is used to simplify the management of users account, and provide a single login for all the Grid applications. GridFTP (*The GridFTP Protocol and Software*) is employed to transfer data between systems securely. Grid-Flow also has the ability to access service-based programs on the remote site if the programs are deployed as Grid services (Foster et al., 2002). Grid computing techniques (described in section 2.3) can satisfy the requirements for building that kind of interface by providing the following: 1) a security infrastructure for user authentication and authorization with remote systems; 2) a data transfer protocol to transfer data between systems with the support of the security infrastructure; and 3) a service-oriented architecture (SOA) for wrapping programs and applications as web or Grid services. There has been a tendency to use more web/Grid services instead of CGI programs for program invocation in the recent development of WFMS.

With more WFMSs using web/Grid services as their program integration mechanisms, the methodology and framework of wrapping programs as web/Grid services have brought forth broad research interests in the workflow community. A novel system, namely WebRun (Guan et al., 2004), is designed for Grid-Flow and described in Appendix B to provide an infrastructure and strategies for wrapping existing programs distributed on various hosting environments as Grid services for users' directly and/or programmatically access. WebRun provides a reference model for building Grid services based on existing applications, as well as the methodologies for developing and deploying Grid services in a Grid environment. The conceptual design of WebRun has been finished and the implementation of the system is on the schedule. To prove the idea of WebRun, a Grid service for Blast (Altschul et al., 1990), namely G-Blast, is implemented following the WebRun model and strategies, and integrated into Grid-Flow. The following section described G-Blast in detail.

7.3 G-BLAST

G-BLAST is an illustrative example of Grid services that are designed and developed based on the architecture of WebRun. The overall architecture of G-BLAST is illustrated in Figure 40. G-BLAST has the following four key components:

1. G-BLAST Core Service: Provides a uniform interface through which a specific version of BLAST (Altschul et al., 1990) was able to be instantiated. This service enables application developers to extend the core interface and incorporate newer versions of BLAST applications.

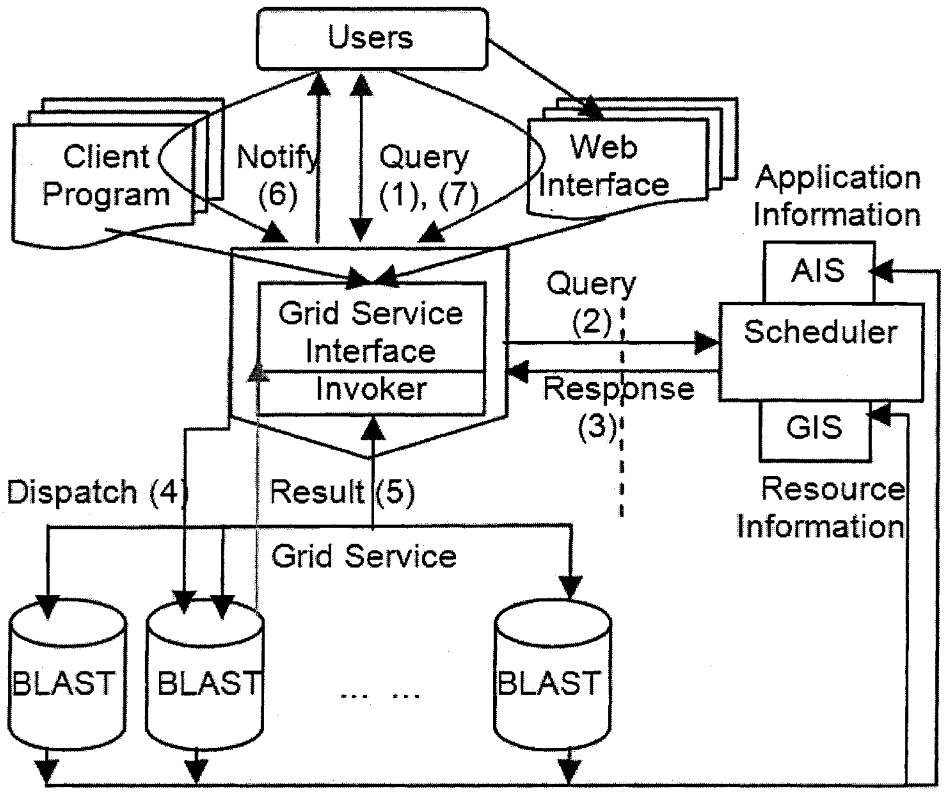


Figure 40. Overall Architecture of G-BLAST.

2. User Interfaces: Provide web and programmatic interfaces for file transfer, job submission, job monitoring and notification. These interfaces support user interactions without exposing any of the details about the grid environment and the application selection process.
3. Scheduler: Selects the best available resource and application based on user request using a two-level adaptive scheduling scheme.
4. BLAST Grid Services: Individual grid services for each of the BLAST variations that are deployed on each of the computational resources.

A typical user interaction in G-BLAST involves the following steps (indicated in Figure 40 with numbered arrows):

1. User submits queries and specifies database name.
2. G-BLAST core interacts with the scheduler.
3. Scheduler makes suggestions on resources and specific type of application to select based on number of queries, query size, database selected, and available resources.
4. Queries are dispatched to suitable computing nodes by invoking appropriate BLAST service(s).
5. Compute nodes execute BLAST search and return a job handle.
6. G-BLAST core service provides client notification.
7. User fetches the results from the G-BLAST core when job is completed.

The remainder of this section focuses on describing the G-BLAST core service, the user interface, and the BLAST Grid service. The scheduler part of G-BLAST is only in its conceptual design and initial implementation (Afgan et al., 2005). Though the scheduling mechanism is crucial for integrating the individual BLAST Grid services together as part of a job dispatching system, scheduling concerns are beyond the scope of this work; G-BLAST still provides a good example of using Grid services in Grid-Flow WFMS. A demonstrative workflow example including the use of G-BLAST is described in detail in section 8.1.4.

7.3.1 G-BLAST Core Service

A BLAST Grid Service with a uniform Grid service interface is deployed on each of the computing resources. It is located between the Invoker and each implementation of BLAST programs. No matter what kind of BLAST programs are deployed on each resource, the BLAST Grid service should cover the differences and provide fundamental features. To facilitate developers integrating individual BLAST instances into the G-BLAST framework, the BLAST Grid service defines the following methods for each instance:

1. UploadFile: Upload query sequences to the computing node.
2. DownloadFile: Download query results from the computing node.
3. RunBlast: Invoke corresponding BLAST programs on the computing node(s).
4. GetStatus: Return current status of the job.
5. NotifyUser: Notify the user once the query is complete and the result is available.

With G-BLAST, developers can easily add a new BLAST service (corresponding to the BLAST programs and the computing resources supporting it) without modifying any G-BLAST core source code. In addition to that, developers can add new BLAST services dynamically, without interrupting any of the other G-BLAST services. G-BLAST employs the creational design pattern factory method (Gamma, Helm, Johnson, & Vlissides, 1995) to enable the invoker to call newly-built BLAST services without changing its source code. To integrate their corresponding BLAST programs into the G-BLAST framework, developers should create and deploy Grid services on each of the computing resources in the Grid. The steps to generate the Grid service on G-BLAST computing resources are described as follows (Sotomayor, 2004):

1. Inherit the interface of the BLAST Grid services to define the service interface for the local computing resources.
2. Implement the local service interface by writing appropriate code to wrap local BLAST programs.
3. Define the deployment parameters for the local BLAST services in the Web Service Deployment Descriptor (Sotomayor, 2004) file.
4. Create the BLAST service GAR (Grid ARchive) file using Ant (*Ant - A Java-based Build Tool*).
5. Deploy the BLAST service on the local Grid service container.

The interface of the local BLAST service is defined in a Grid Web Service Description Language (GWSDL) (Berman et al., 2003) file. Figure 41 presents the interface definition of method TransferFile in the GWSDL file of a BLAST service. This definition describes the method TransferFile that has three input parameters (fileName, srcHost, remoteHost) and one return result (TFResponse). Other methods of the BLAST service are defined accordingly in the GWSDL file.

As described in Figure 42, Invoker and BLASTService are two abstract classes representing the invoker in the G-BLAST service core and the BLAST services on computing resources, correspondingly. When a new BLAST service (for example, mpiBLAST (Darling et al., 2003)) is added into the system, the relevant invoker (mpiInvoker) for that service must be integrated as a subclass of the class Invoker. When the invoker wants to call the new BLAST service, it can first create an instance of mpiInvoker, then let the new invoker generate an instance of mpiBLAST by calling the member function CreateService(). Thus, the invoker does not need to hard-code the instantiation of each type of BLAST service.

```
<!-- TansferFile-->
<xsd:element name="TransferFile" type="tns:TransferFile">
        <xsd:complexType name="TransferFile">
                <xsd:sequence>
                <xsd:element name="fileName" type="xsd:string"/>
                <xsd:element name="srcHost" type="xsd:string"/>
                <xsd:element name="remoteHost" type="xsd:string"/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>
<xsd:element name="TFResponse" type="tns:TFResponse">
        <xsd:complexType name="TFResponse"/>
</xsd:element>
```

Figure 41. Interface of Method *TransferFile* in GWSDL.

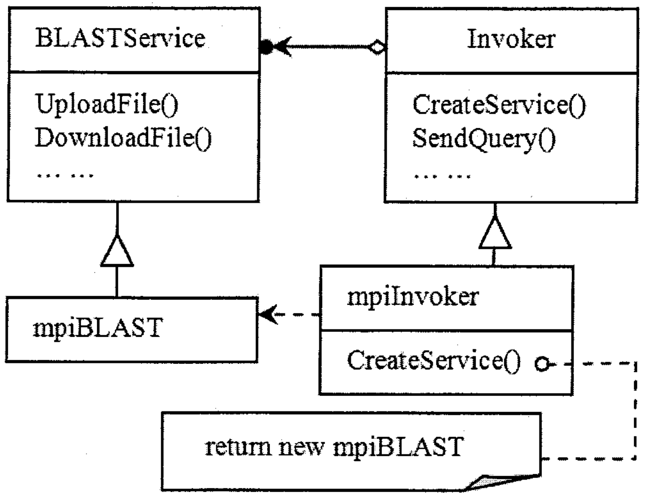


Figure 42. Factory Method for BLAST service.

This design pattern encapsulates the knowledge of which BLAST services to create and delegate the responsibility of choosing the appropriate BLAST service(s) to the scheduler. The Invoker could invoke more than one BLAST service based on the availability of resources to satisfy user requirements.

7.3.2 User Interfaces

The G-BLAST framework provides unified, integrated interfaces for users to invoke BLAST services over the heterogeneous and distributed Grid computing environment. The interfaces summarize the general functionalities that are provided by each individual BLAST service, as well as cover the implementation details from the end-users. Two user interfaces are currently implemented to satisfy different users' requirements. For users who want to submit queries as part of workflow, a programmable interface is furnished through a Grid Service. Service data and notification mechanism supported by Grid Services are integrated into the BLAST Grid service to provide stateful services with better job monitoring and notification. For users who are familiar with traditional BLAST interface (like NCBI BLAST (NCBI, 2004)) and want to submit each query with individual parameter settings, a web interface is implemented for job submission, monitoring, and file management. The programmatic interface is designed as follows:

1. FileTransfer: This method can be used to upload the local files to the servers, download the files to a local machine, or perform third-party file transfers using GridFTP (*The GridFTP Protocol and Software*). G-BLAST uses GridFTP for transferring the query files instead of directly sending the queries as SOAP (W3C, 2003) messages to avoid serialization and deserialization overheads.

2. BLASTRequest: This method defines users' request of BLAST queries via a Job Description File (JDF) – an XML file based on Job Submission Description Language (JSDL) (*Job Submission Description Language (JSDL) Specification*, 2005). JDF will be consumed by the Scheduler to generate appropriate job submission strings in the Resource Specification Language (RSL) (*The Globus Resource Specification Language RSL v1.0*). Consequently, the job submission will be taken by the Invoker and submitted to local job managers on the computing nodes.
3. JobSubmit: This method submits queries to G-BLAST Core Service and returns a job handle.
4. JobStatus: The status of the submitted job can be queried with this method using the job handle returned by the JobSubmit method. Since the order of completion may not be in the same order as that of submission this method provides a convenient way to check the status of submitted queries.
5. ResultRetrieve: This method provides an approach to fetching the results of a particular job using the job handle.

G-BLAST exploits the notification mechanism (Foster et al., 2002) provided by grid services in two aspects. One aspect is the notification of changes by BLAST services to the scheduler. The other aspect is the notification of job completion to the end-users. Both of these two instances strictly follow the protocol of notification. In notification of service changes, the BLAST services are the notification source, and the scheduler is the notification subscriber. Whenever the BLAST service on the computing node has any changes, the service itself will automatically notify the scheduler with up-to-date information. This mechanism keeps the scheduler updated with the most recent status of the

BLAST service, therefore helps the scheduler make informed decisions on the selection of computing resources. Notification for job completion has a similar implementation, except that the notification sink is the registered client program.

The notification mechanism in Grid Services is closely related to the service data. The notification sinks do not subscribe to the whole service, but to a particular Service Data Element (SDE) (Foster et al., 2002). The function addListener is called when the subscriber wants to register with a particular SDE. After registration, the subscriber waits for notification, instead of querying the services constantly. Whenever any changes occur in the BLAST service, the function notifyChange will be called to request the SDE to notify its subscribers. After receiving this request, the SDE will send the notifications to the subscribers by calling function deliverNotification. Since this notification includes the actual service data, the subscriber does not need to make any more calls to the service.

To facilitate users using G-BLAST, a programming template is also provided to guide users coding their own client program for G-BLAST service invocation. Figure 43 demonstrates the major part of a client program that invokes a G-BLAST service by creating a Grid service handler, uploading query sequence(s) to the back end server, submitting a query job, checking the job status, and finally retrieving back the query results in a sequential mode.

In addition to providing a programmatic interface for the end-user, the framework also provides a web workspace that supports the needs of a general, non-technical Grid user who prefers graphical user interface to writing code. The most common needs of a general user are file management, job submission, and job monitoring.

```
// Get command-line argument as Grid Service Handler
URL GSH = new java.net.URL(args[0]);

// Get a reference to the Grid Service instance
GBLASTServiceGridLocator gblastServiceLocator = new GBLASTServiceGridLoca-
tor();
GBLASTPortType gblast = gblastServiceLocator.getGBLASTServicePort(GSH);
  ..........
//Query sequence uploading
gblast.FileTransfer(inputFile, src, remote);
  ..........
//Submit query as a job
gblast.BLASTRequest(blastRequest);
jobid=gblast.JobSubmit();
  ..........
//Check query (job) status
gblast.JobStatus(jobid);
  ..........
//Retrieve back the query result
gblast.ResultRetrive(jobid);
```

Figure 43. Client Program to Invoke G-BLAST Service.

File management is supported through a web file browser allowing users to upload new query files or download search result files. It is a simplified version of an FTP client that is developed in PHP. The job submission module is made as simple as possible to use, the user after naming the job for easy reference only provides or selects a search query file and chooses the database to search against. Application selection, resource selection, file mapping, and data transfer are handled automatically by the system. Finally, the job monitoring module presents the user with the list of his or her jobs. It includes a date range allowing the user to view not only the currently running jobs, but the completed jobs as well. When viewing the jobs, the user is given the name of the job, current status (running, done, pending, or failed) and job execution start/end time. Upon clicking on the job name, the user can view more detailed information about the query file, database used, and start and end time. The user is also given the option to re-submit a job with the same set of parameters or after changing one of the parameters.

7.4 Summary

Grid-Flow's ability to integrate web and Grid services has been fully investigated in this chapter. The web and Grid services are mainly used to wrap remote non-interactive programs as workflow transitions. G-BLAST service, as a demonstrative example of wrapping bioinformatics tools as Grid services, is implemented and discussed in detail on the aspects of the architecture, core services, and user interface. So far all the technologies used in Grid-Flow have been presented. The next chapter illustrates the overall design and implementation issues of scientific workflows by demonstrating Grid-Flow with two use cases.

8 EXPERIMENTAL WORKFLOW APPLICATIONS

This chapter presents examples that describe how to design and execute workflow applications for biological data analyses within the Grid-Flow system. Two real workflow applications in Bioinformatics research are used to demonstrate the benefits that workflow technologies bring to the biological research. Each workflow application is presented with the overall description, the organization and its components, the design, the implementation, and the execution.

8.1 Transmembrane Region Analysis

This experiment works to introduce the basic idea of using Grid-Flow system to model and execute a bioinformatics data analysis process as a scientific workflow. It describes in detail the workflow modeling procedure with Petri nets, the operations of data and online program registration, the GFDL script generation, and the execution of the workflow including the data matching between linked programs. Here this experiment is used to demonstrate Grid-Flow system's abilities of modeling workflow processes with Petri nets, translating Petri net models into GFDL scripts, and controlling data and online resources for workflow execution.

The requirements of the Transmembrane Region Analysis workflow were described in section 3.5. This section focuses on the implementation details of this workflow process.

8.1.1 Data Analysis without Workflow

Before workflow technologies were applied to biological data analysis, biologists used to analyze the data manually following a step-by-step procedure designed before the experiment. For example, to analyze the transmembrane region of a particular DNA sequence, a biologist needs to follow these steps:

1 Data preparation for the experiment

1.1 Get the accession number of the DNA sequence;

1.2 Choose the parameters for transmembrane region analysis based on experiment plan;

2 Get the DNA sequence

2.1 Open the website http://www.ncbi.nlm.nih.gov/ in a web browser, fill in the parameters for sequence search, and click "Go";

2.2 Wait till the search result displayed in the browser, click the one that best matches the target sequence;

2.3 In the sequence viewer page, locate the DNA sequence (the "ORIGIN" part), copy and paste that sequence into a temporary file with a file editor;

3 Analyze the transmembrane regions

3.1 Open the website http://www.ch.embnet.org/software/TMPRED_form.html in a web browser;

3.2 Set all the necessary parameters for the analysis (though some parameters can use the default settings);

3.3 Copy and past the DNA sequence from the temporary file into the "Query Sequence" box, and click the "Run TMpred" button;

3.4 Wait till the result of the transmembrane analysis of the DNA sequence is displayed in the browser;

4 View and save the results

4.1 Check the result displayed in the web browser, and save the results in the file system for future reference.

8.1.2 Workflow Designers and Workflow Users

The steps presented in previous section are the base of the blueprint of the transmembrane region analysis workflow. To model such a workflow, a workflow designer should simply model each major step as a task, wrap all the functions represented by the sub-steps in the task, implement the task, and connect tasks following the designed logic as a workflow.

A workflow designer is the key person who designs the whole workflow, tests it, and delivers it to the workflow users. This workflow designer should be a domain expert, equipped with appropriate knowledge of process modeling and workflow systems. The responsibility of a workflow designer is to help the workflow users locate their user requirements, understand the whole procedure of the experiment, draw the blueprint of the workflow, implement the workflow, test it via test cases, write the documentation for the resulting workflow, teach the end user the way to run that workflow, edit this workflow according to users' requirements, and maintain the workflow. WFMS can facilitate workflow designers on designing and implementing workflows, as well as provide an easy-to-use platform to workflow users for executing and monitoring workflows. To design a workflow in Grid-Flow, a workflow designer need first to register the data and programs that will be used in the workflow, then draw the Petri net model with the Grid-Flow GUI, run the generated GFDL script within the Grid-Flow interface, and test the workflow with designed test cases. The aspects of registering data and programs, designing and implementing the workflow, and running test cases for the workflow are described in detail in the following three sections, respectively.

From the viewpoint of workflow users, running a biological workflow is much easier than analyzing the data manually following a step-by-step procedure. With Grid-Flow, a workflow user only needs to load the workflow (GFDL script file) into the Grid-Flow interface, click the "Run" button to start the workflow, feed the workflow with required user input, monitor the execution, and await the results. All the data and programs are predefined in the workflow. And all the control logic is managed by workflow itself. Users can easily change the parameters of the workflow and reuse it for analysis of other biological data.

8.1.3 Registering Data/Program

To accomplish the analysis, the following data and programs need to be registered with Grid-Flow system. Names, registration codes, and DDFs (for data only) are listed for selected data and programs. Section 6.2 describes the meaning of the data and program registration procedures in detail, as well as the format of the DDF and the semantics of the registration code.

Data:

1. *PSeq* contains the access number of particular sequence being worked on. Its content, DDF, and registration code are shown in Figure 44.

PSeq file:

```
Nucleotide
ay057452
```

DDF of PSeq:

```
DDF
<Database, String, "database name db", ".11">
<AccessID, String, "access ID term", ".12">
End DDF
FORMS
End FORMS
```

Registration Code:

```
Register Data PSeq As Text;
Set data_source="file:///C:\BioFlow\Data\PSeq.txt";
Set data_format="TEXT"
Set data_ddf="File:///C:\BioFlow\DDF\PSeq.ddf";
End;
```

Figure 44. The Content, DDF, and Registration Code of PSeq.

2. *TMPredpar* contains the parameters of the program TMPred. Its content, DDF, and registration code are shown in Figure 45.

3. Other data need to be registered are *InputSNInput, InputSNOutput, NCBISearchInput, NCBISearchOutput, TMPredInput, TMPredOutput, DisplayInput,* and *DisplayOutput.* All of these data are the input/output interfaces for the programs. These data can be registered in the same way as was just presented.

TMPredpar file:

```
plain_text
html
17
33
```

DDF of TMPredpar:

```
DDF
<format, String, "format", ".l1">
<outputmode, String, "output mode outmode", ".l2">
<minimum, Integer, "minimum min", ".l3">
<maximum, Integer, "maximum max", ".l4">
End DDF
FORMS
End FORMS
```

Registration Procedure:

```
Register Data TMPredpar As Text;
Set data_source="file:///C:\BioFlow\Data\TMPredpar.txt";
Set data_format="TEXT"
Set data_ddf="File:///c:\BioFlow\DDF\TMPredpar.ddf";
End;
```

Figure 45. The Content, DDF, and Registration Code of TMPredpar.

Program:

1. *NCBISearch* searches the protein accession number on NCBI website (*NCBI Website*), and returns the protein sequence. The registration procedure of *NCBISearch* is shown in Figure 46.

2. *TMPred* performs the transmembrane regions analysis based on provided protein sequences. The registration procedure of *TMPred* is shown in Figure 47.

```
Register Program NCBISearch As OS_Program;
Input NCBISearchInput;
Output NCBISearchOutput;
Set program_source=" http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?...";
End;
```

Figure 46. Registration Procedure of NCBISearch.

```
Register Program TMPred As OS_Program;
Input TMPredInput;
Output TMPredOutput;
Set program_source="http://www.ch.embnet.org/...";
End;
```

Figure 47. Registration Procedure of TMPred.

3. Other programs need to be registered are *InputSN* and *Display*.

8.1.4 Design & Implementation

To model this workflow process, a workflow designer may intuitively translate the graph in Figure 6 into a Petri net model as shown in Figure 48. From a structural viewpoint, the Petri net model is similar to the data and program model described in Figure 6. The Start and End places indicates the beginning and finish of the process. Each transition corresponds to a program in the data and program model. The data is represented by the token, and places are data containers connecting transitions. Transitions *Input Data* and *TMPred* are featured as AndSplit and AndJoin transitions respectively, since a parallel structure exists in this workflow process. Transition *Input Data* is the starting transition of the parallel structure, and *TMPred* is the ending transition of the same structure. With this Petri net model, the graphical user interface can generate the GFDL as shown in Figure 49, under the administration of the workflow designer.

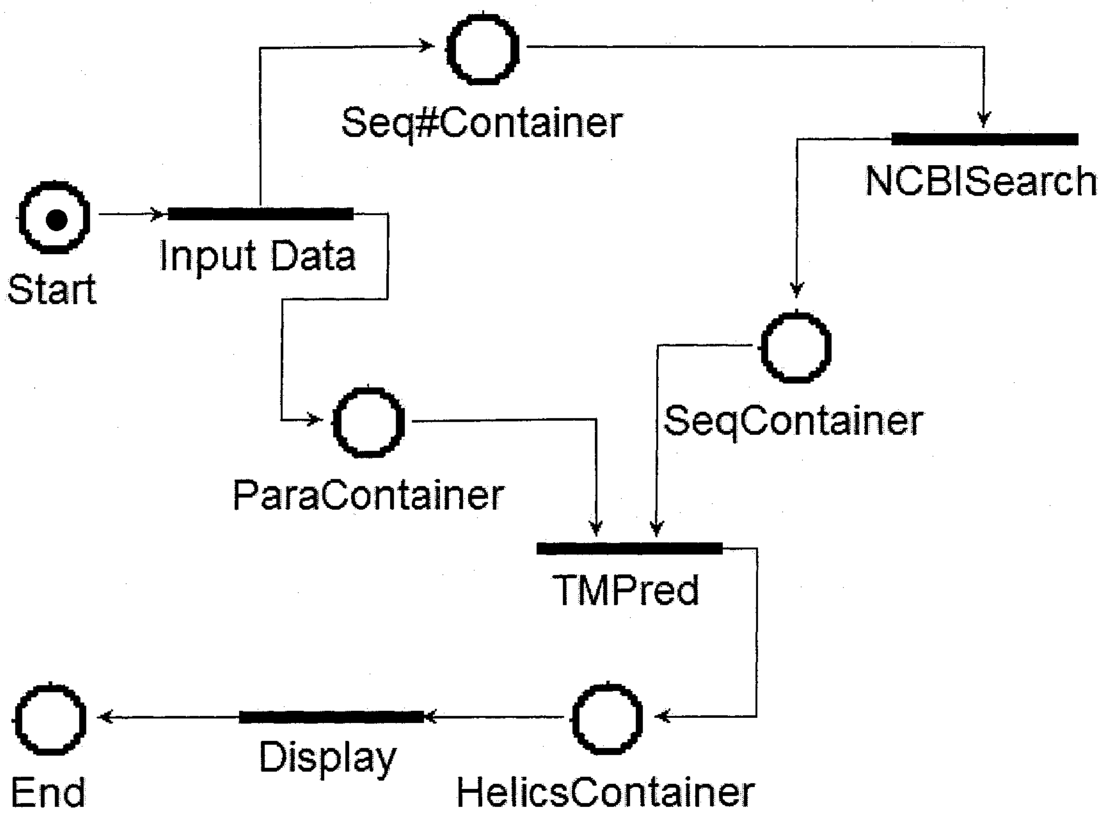


Figure 48. A Petri Net Model for Analysis of Transmembrane Regions.

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

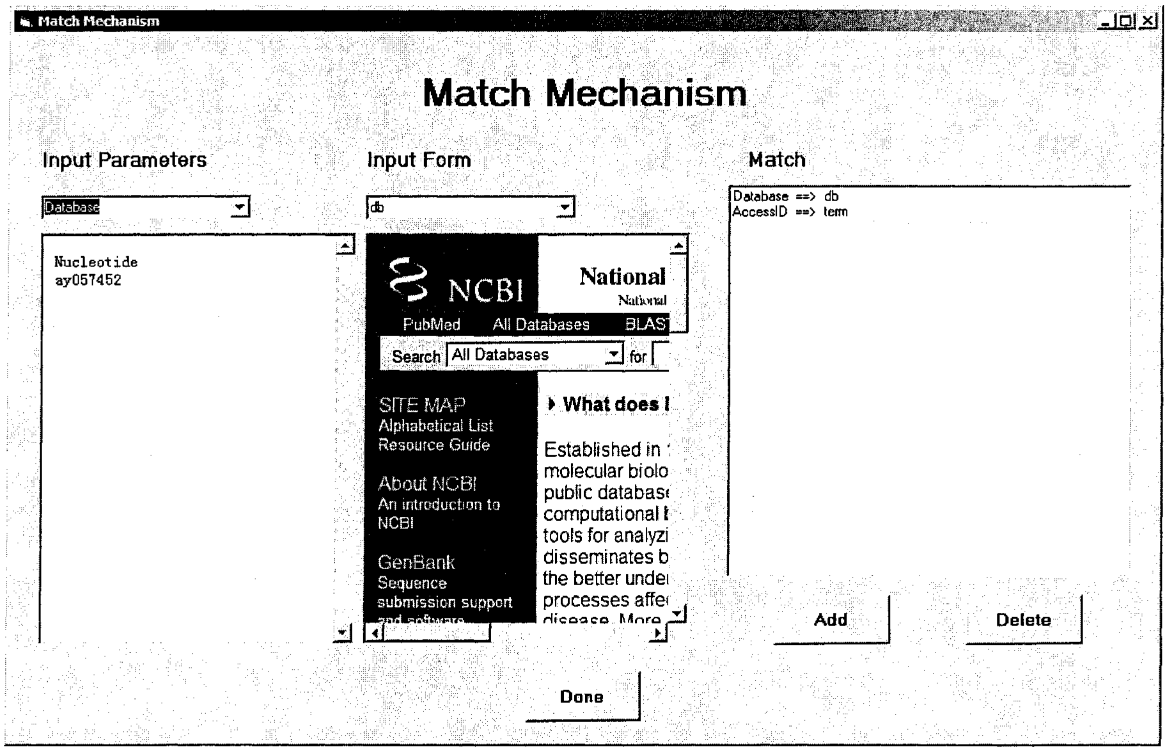
Figure 49. GFDL for the Workflow of Transmembrance Regions Analysis.

In this GFDL sentence, program *Display*, *TMPred*, *NCBISearch*, and *Input Data* have already been registered with Program Registration component before the design of the workflow process. Program *Display* and *Input Data* are programs on the local computer where the workflow is executed. Programs *TMPred* and *NCBISearch* locate on web servers which can only be accessed by Internet. The data are 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 network, the Grid-Flow engine should invoke the programs through program integration component. The input data and temporary intermediate results are transferred over the network.

8.1.5 Execution

During the execution, Grid-Flow engine first fetches the accession number of protein sequence through the program *InputSN*. Then Grid-Flow engine submits this accession number to NCBI website for searching. The accession number needs to be matched with the input interface of NCBI website. This matching is performed by Grid-Flow engine. Matching mechanism between data and program interface has been discussed in section 6.3. The matching result is provided to the end-users for review as shown in Figure 50. After approval by the user, Grid-Flow engine invokes the program *NCBI-Search* and generates the desired sequence, which is shown in Figure 51. Then Grid-Flow engine tries to match the sequence and *TMPredpar* to the input interface of program *TMPred*. The matching job is shown in Figure 52. The matching procedure has also been shown in section 6.3 as an example. If matched and approved, the protein sequence is sent to *TMPred* for analysis of transmembrane regions. The analysis result is retrieved and displayed to the end-user as Figure 53. The successful execution of this workflow process shows that users can use Grid-Flow system to design and execute workflow processes. During the design and execution, a Petri net model is generated to describe the workflow process in the graphical user interface. After that, the model is translated into

GFDL script. The Grid-Flow engine then takes over the GFDL script and maps it to the execution sequence of tasks with proper data feeding, generation, and transportation. What's more, the invocation of programs *TMPred* and *NCBISearch* proves that Grid-Flow system can schedule the data and online computing resources to support the work-flow execution. That's all that Grid-Flow system aims to achieve.


Match Mechanism

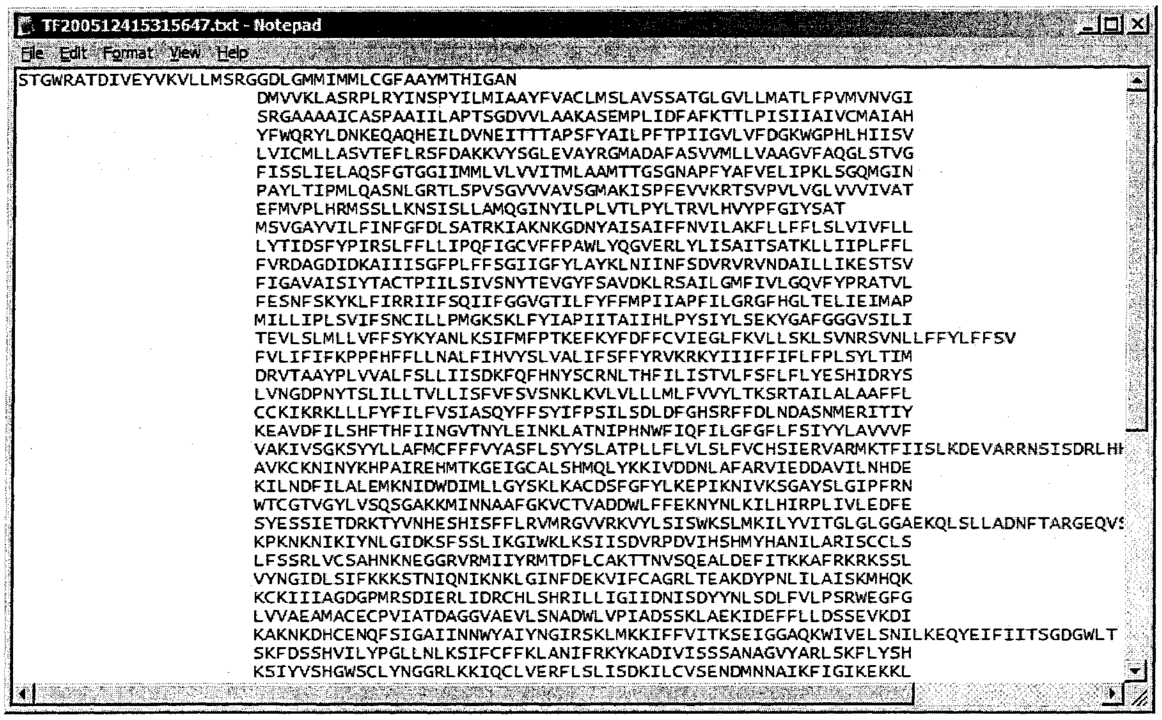
Input Parameters: Database

Nucleotide
ay057452

Input Form: db

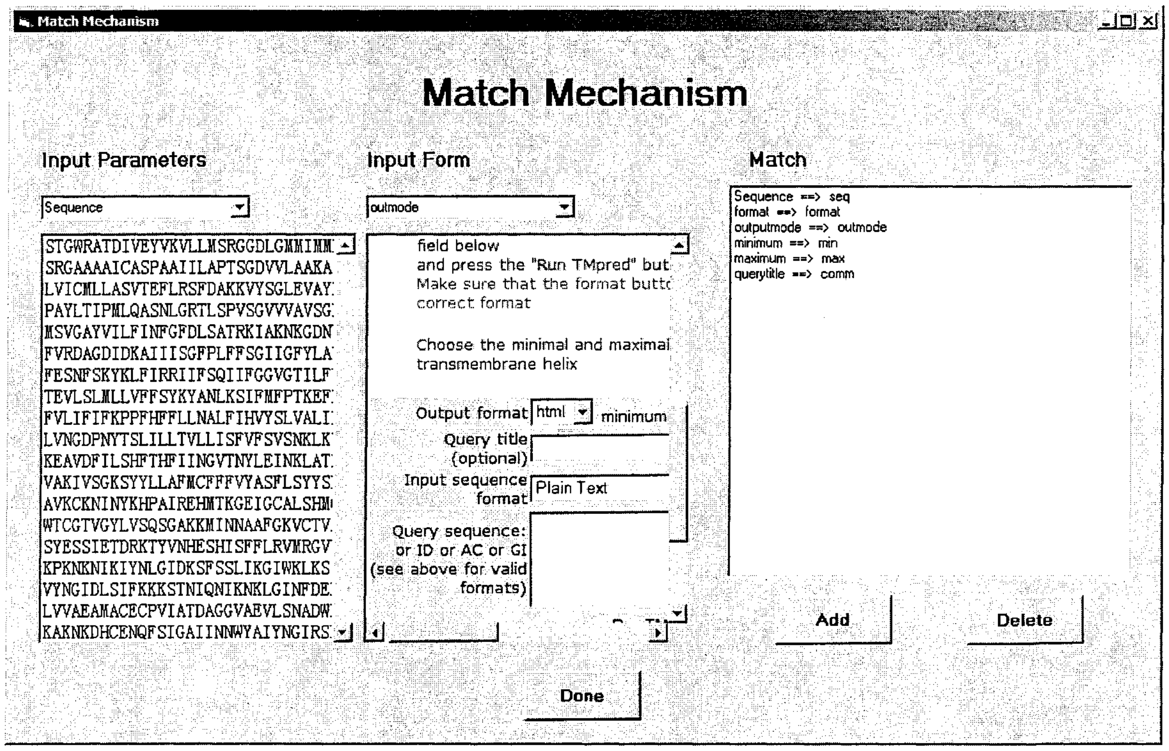
Match:
Database ==> db
AccessID ==> term

Figure 50. Matching PSeq to NCBISearch.



```
STGWRATDIVEYVKVLLMSRGGDLGMMIMMLCGFAAYMTHIGAN
                         DMVVKLASRPLRYINSPYILMIAAYFVACLMSLAVSSATGLGVLLMATLFPVMVNVGI
                         SRGAAAAICASPAAIILAPTSGDVVLAAKASEMPLIDFAFKTTLPISIIAIVCMAIAH
                         YFWQRYLDNKEQAQHEILDVNEITTTAPSFYAILPFTPIIGVLVFDGKWGPHLHIISV
                         LVICMLLASVTEFLRSFDAKKVYSGLEVAYRGMADAFASVVMLLVAAGVFAQGLSTVG
                         FISSLIELAQSFGTGGIIMMLVLVVITMLAAMTTGSGNAPFYAFVELIPKLSGQMGIN
                         PAYLTIPMLQASNLGRTLSPVSGVVVAVSGMAKISPFEVVKRTSVPVLVGLVVVIVAT
                         EFMVPLHRMSSLLKNSISLLAMQGINYILPLVTLPYLTRVLHVYPFGIYSAT
                         MSVGAYVILFINFGFDLSATRKIAKNKGDNYAISAIFFNVILAKFLLFFLSLVIVFLL
                         LYTIDSFYPIRSLFFLLIPQFIGCVFFPAWLYQGVERLYLISAITSATKLLIIPLFFL
                         FVRDAGDIDKAIIISGFPLFFSGIIGFYLAYKLNIINFSDVRVRVNDAILLIKESTSV
                         FIGAVAISIYTACTPIILSIVSNYTEVGYFSAVDKLRSAILGMFIVLGQVFYPRATVL
                         FESNFSKYKLFIRRIIFSQIIFGGVGTILFYFFMPIIAPFILGRGFHGLTELIEIMAP
                         MILLIPLSVIFSNCILLPMGKSKLFYIAPIITAIIHLPYSIYLSEKYGAFGGGVSILI
                         TEVLSLMLLVFFSYKYANLKSIFMFPTKEFKYFDFFCVIEGLFKVLLSKLSVNRSVNLLFFYLFFSV
                         FVLIFIFKPPFHFFLLNALFIHVYSLVALIFSFFYRVKRKYIIIFFIFLFPLSYLTIM
                         DRVTAAYPLVVALFSLLIISDKFQFHNYSCRNLTHFILISTVLFSFLFLYESHIDRYS
                         LVNGDPNYTSLILLTVLLISFVFSVSNKLKVLVLLLMLFVVYLTKSRTAILALAAFFL
                         CCKIKRKLLLFYFILFVSIASQYFFSYIFPSILSDLDFGHSRFFDLNDASNMERITIY
                         KEAVDFILSHFTHFIINGVTNYLEINKLATNIPHNWFIQFILGFGFLFSIYYLAVVVF
                         VAKIVSGKSYYLLAFMCFFFVYASFLSYYSLATPLLFLVLSLFVCHSIERVARMKTFIISLKDEVARRNSISDRLH
                         AVKCKNINYKHPAIREHMTKGEIGCALSHMQLYKKIVDDNLAFARVIEDDAVILNHDE
                         KILNDFILALEMKNIDWDIMLLGYSKLKACDSFGFYLKEPIKNIVKSGAYSLGIPFRN
                         WTCGTVGYLVSQSGAKKMINNAAFGKVCTVADDWLFFEKNYNLKILHIRPLIVLEDFE
                         SYESSIETDRKTYVNHESHISFFLRVMRGVVRKVYLSISWKSLMKILYVITGLGLGGAEKQLSLLADNFTARGEQV
                         KPKNKNIKIYNLGIDKSFSSLIKGIWKLKSIISDVRPDVIHSHMYHANILARISCCLS
                         LFSSRLVCSAHNKNEGGRVRMIIYRMTDFLCAKTTNVSQEALDEFITKKAFRKRKSSL
                         VYNGIDLSIFKKKSTNIQNIKNKLGINFDEKVIFCAGRLTEAKDYPNLILAISKMHQK
                         KCKIIIAGDGPMRSDIERLIDRCHLSHRILLIGIIDNISDYYNLSDLFVLPSRWEGFG
                         LVVAEAMACECPVIATDAGGVAEVLSNADWLVPIADSSKLAEKIDEFFLLDSSEVKDI
                         KAKNKDHCENQFSIGAIINNWYAIYNGIRSKLMKKIFFVITKSEIGGAQKWIVELSNILKEQYEIFIITSGDGWLT
                         SKFDSSHVILYPGLLNLKSIFCFFKLANIFRKYKADIVISSSANAGVYARLSKFLYSH
                         KSIYVSHGWSCLYNGGRLKKIQCLVERFLSLISDKILCVSENDMNNAIKFIGIKEKKL
```

Figure 51. Protein Sequence Generated by NCBISearch.



Sequence

```
STGWRATDIVEYVKVLLMSRGGDLGMMIMM
SRGAAAAICASPAAIILAPTSGDVVLAAKA
LVICMLLASVTEFLRSFDAKKVYSGLEVAY
PAYLTIPMLQASNLGRTLSPVSGVVVAVSG
MSVGAYVILFINFGFDLSATRKIAKNKGDN
FVRDAGDIDKAIIISGFPLFFSGIIGFYLA
FESNFSKYKLFIRRIIFSQIIFGGVGTILF
TEVLSLMLLVFFSYKYANLKSIFMFPTKEF
FVLIFIFKPPFHFFLLNALFIHVYSLVALI
LVNGDPNYTSLILLTVLLISFVFSVSNKLK
KEAVDFILSHFTHFIINGVTNYLEINKLAT
VAKIVSGKSYYLLAFMCFFFVYASFLSYYS
AVKCKNINYKHPAIREHMTKGEIGCALSHM
WTCGTVGYLVSQSGAKKMINNAAFGKVCTV
SYESSIETDRKTYVNHESHISFFLRVMRGV
KPKNKNIKIYNLGIDKSFSSLIKGIWKLKS
VYNGIDLSIFKKKSTNIQNIKNKLGINFDE
LVVAEAMACECPVIATDAGGVAEVLSNADW
KAKNKDHCENQFSIGAIINNWYAIYNGIRS
```

outmode

field below
and press the "Run TMpred" but
Make sure that the format butto
correct format

Choose the minimal and maximal
transmembrane helix

Output format: html minimum

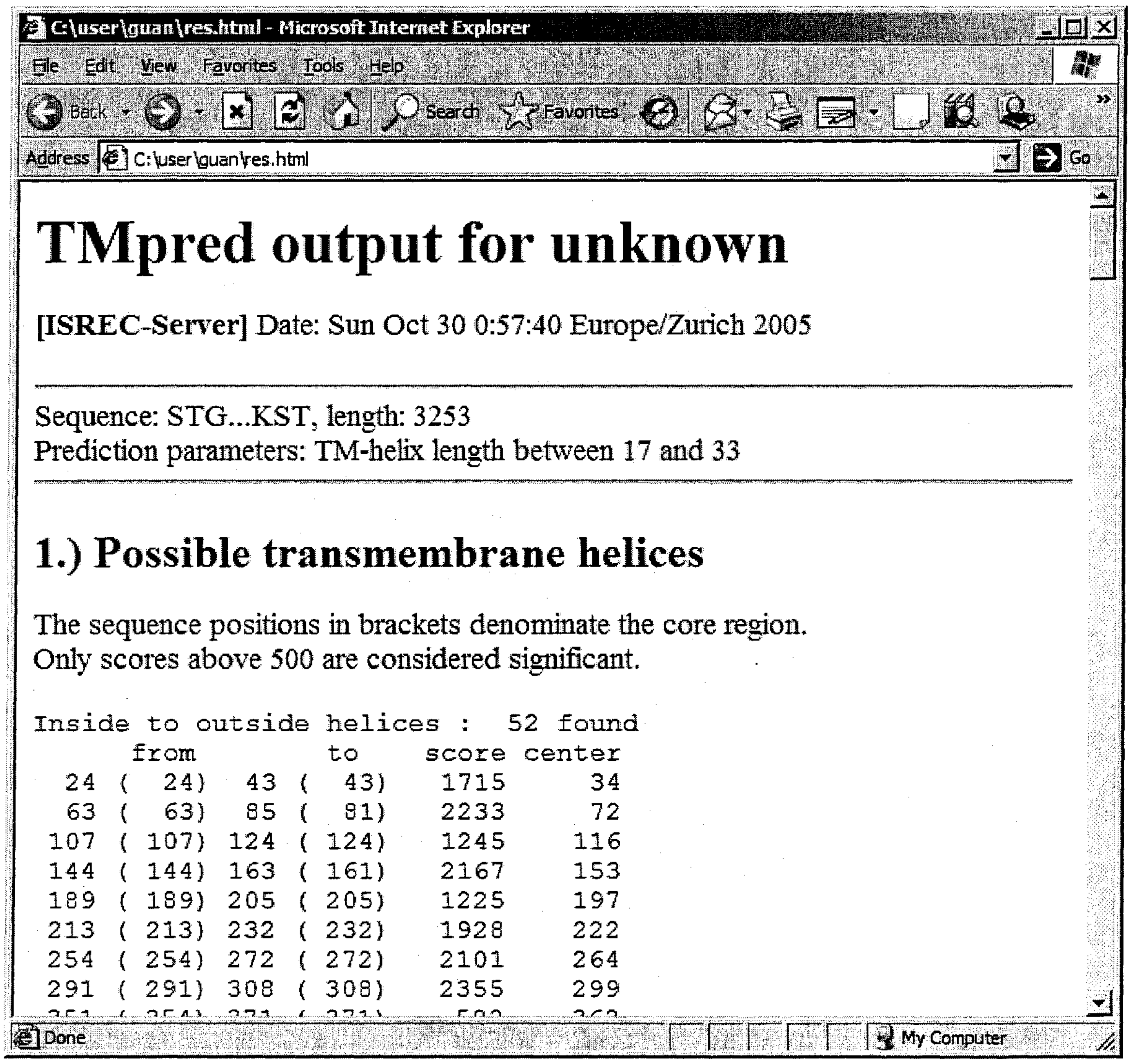
Query title (optional):

Input sequence format: Plain Text

Query sequence: or ID or AC or GI (see above for valid formats)

```
Sequence ==> seq
format ==> format
outputmode ==> outmode
minimum ==> min
maximum ==> max
querytitle ==> comm
```

Figure 52. Matching Sequence and TMPredpar to TMPred.



C:\user\guan\res.html - Microsoft Internet Explorer

Address C:\user\guan\res.html

TMpred output for unknown

[ISREC-Server] Date: Sun Oct 30 0:57:40 Europe/Zurich 2005

Sequence: STG...KST, length: 3253
Prediction parameters: TM-helix length between 17 and 33

1.) Possible transmembrane helices

The sequence positions in brackets denominate the core region.
Only scores above 500 are considered significant.

```
Inside to outside helices :  52 found
      from         to    score center
  24 (  24)  43 (  43)   1715     34
  63 (  63)  85 (  81)   2233     72
 107 ( 107) 124 ( 124)   1245    116
 144 ( 144) 163 ( 161)   2167    153
 189 ( 189) 205 ( 205)   1225    197
 213 ( 213) 232 ( 232)   1928    222
 254 ( 254) 272 ( 272)   2101    264
 291 ( 291) 308 ( 308)   2355    299
```

Done   My Computer

Figure 53. Transmembrane Regions for Protein Sequence.

8.2 Protein Function and Expression

Prediction of transmembrane regions is only part of the job for protein function and expression. The following is the whole illustration of the job on protein function and expression.

The prediction of the protein function and expression can be drawn based on three aspects of related information (Lawerence et al., 2003): 1) the functionalities of the other protein sequences that have similar sequence structures; 2) the transmembrane regions and orientation of the protein sequence; and 3) the promoter existed in the corresponding DNA sequence. To get this vital information, biologists need to analyze the protein sequence with some web-based tools, find the similar protein sequences via BLAST search (Darling et al., 2003), and collect the literature of similar protein sequences. This is a typical data analysis experiment scheduled by biologists.

This experiment describes in detail the workflow modeling approach using the Data/Program Chart and Petri net in Grid-Flow system. The demonstration in this section focuses on how to use Petri net structures modeling complex workflow process and how to mapping the Petri net structures into GFDL scripts. The generation of a coordinating GFDL script based on a Petri net model is also described with this experiment. In addition, the usage of the G-BLAST service provided by WebRun system is presented in this experiment, partially revealing the flexibility and portability that Grid computing techniques bring to the Grid-Flow system.

8.2.1 Problem Description

Intuitively, one can divide the whole experimental process into three independent threads according to the different goals they strive to achieve. Each thread collects one aspect of related information. The flowchart of the threads is shown in Figure 54. The respective goals of each task are described as the following.

Thread 1 Literature Search. In this thread, the workflow process first input the protein sequence into a BLAST search to find similar protein sequences. Then the system retrieves all of the accession number of those protein sequences whose similarity evaluation code is less than a threshold. After that, the system accesses each similar protein sequence through NCBI PubMed website (NCBI), and extracts the literature-related information from web pages. Finally, system puts all of the information together and displays it to the user.

Thread 2 Transmembrane Regions Finding. In this thread, the workflow process puts the protein sequence into the web-based tool TMPred (*TMpred - Prediction of Transmembrane Regions and Orientation*), collects the prediction result, and then shows the result to the user. Since the functionality of the workflow is similar to the workflow of transmembrane region analysis, some parts of the workflow described in 8.1 can be reused in this thread.

Thread 3 Promoters Predicting. In this thread, the workflow process finds the corresponding DNA sequences from the NCBI (*NCBI Website*) website, puts it into a web-based tool "Neural Network Promoter Prediction" (*BDGP: Neural Network Promoter Prediction*) (shown in the web browser window of Figure 55), collects the prediction results, and displays the results to the end-users.

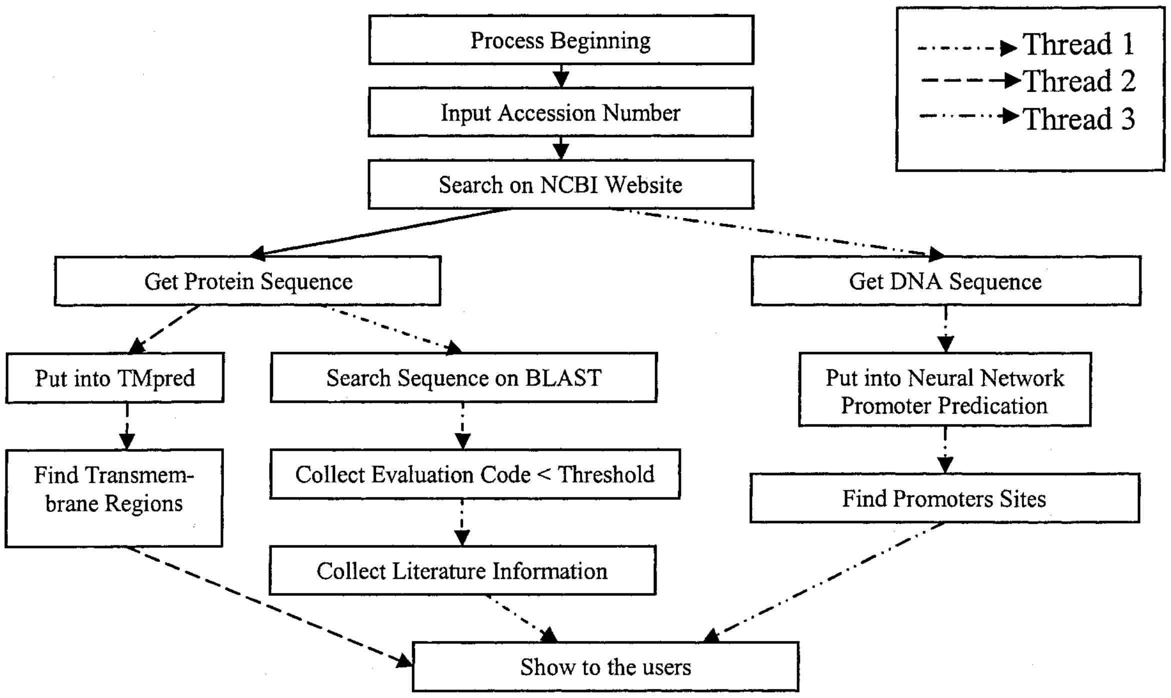
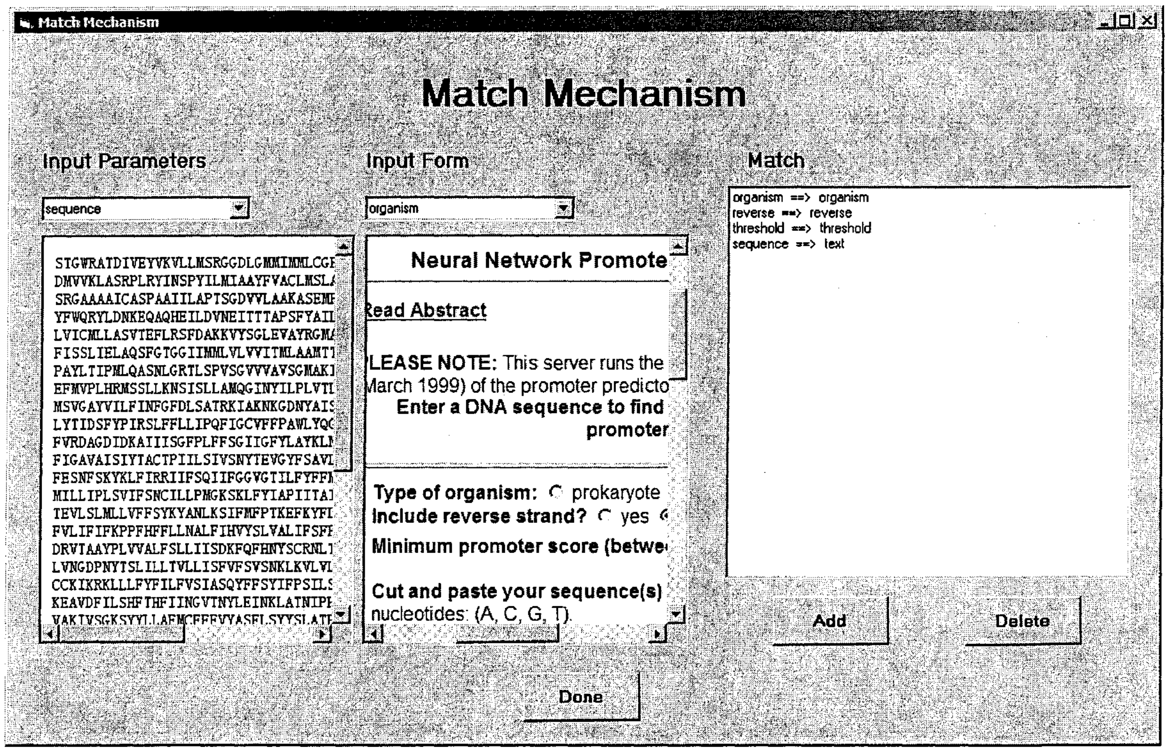


Figure 54. Protein Function and Expression.



Match Mechanism

Input Parameters

sequence

```
STGWRATDIVEYVKVLLMSRGGDLGMMIMMLCGF
DMVVKLASRPLRYINSPYILMIAAYFVACLMSLA
SRGAAAAICASPAAIILAPTSGDVVLAAKASEMF
YFWQRYLDNKEQAQHEILDVNEITTTAPSFYAIL
LVICMLLASVTEFLRSFDAKKVYSGLEVAYRGMA
FISSLIELAQSFGTGGIIMMLVLVVITMLAAMTT
PAYLTIPMLQASNLGRTLSPVSGVVVAVSGMAKI
EFMVPLHRMSSLLKNSISLLAMQGINYILPLVTL
MSVGAYVILFINFGFDLSATRKIAKNKGDNYAIS
LYTIDSFYPIRSLFFLLIPQFIGCVFFPAWLYQC
FVRDAGDIDKAIIISGFPLFFSGIIGFYLAYKLN
FIGAVAISIYTACTPIILSIVSNYTEVGYFSAVL
FESNFSKYKLFIRRIIFSQIIFGGVGTILFYFFN
MILLIPLSVIFSNCILLPMGKSKLFYIAPIITAI
TEVLSLMLLVFFSYKYANLKSIFMFPTKEFKYFL
FVLIFIFKPPFHFFLLNALFIHVYSLVALIFSFF
DRVTAAYPLVVALFSLLIISDKFQFHNYSCRNLT
LVNGDPNYTSLILLTVLLISFVFSVSNKLKVLVL
CCKIKRKLLLFYFILFVSIASQYFFSYIFPSILS
KEAVDFILSHFTHFIINGVTNYLEINKLATNIPF
VAKIVSGKSYYLLAFMCFFFVYASFLSYYSLATF
```

Input Form

organism

Neural Network Promote

Read Abstract

PLEASE NOTE: This server runs the
March 1999) of the promoter predicto
Enter a DNA sequence to find
promoter

Type of organism: ○ prokaryote
Include reverse strand? ○ yes ●
Minimum promoter score (betwe

Cut and paste your sequence(s)
nucleotides: (A, C, G, T).

Match

```
organism ==> organism
reverse ==> reverse
threshold ==> threshold
sequence ==> text
```

Figure 55. Matching Input Data to Neural Network Promoter Prediction.

8.2.2 Design and Implementation

Since this is a somewhat more complex workflow process, a Data/Program Chart describing all of these three threads is needed to help better communication between end-users and the workflow designer. The Data/Program Chart drawn by the user is shown in Figure 56. Basically the whole process starts from the task *InputSN* that asks the user for the identification number of the target protein sequence. Then the task *NCBISearch* is invoked to search the protein sequence in NCBI database and retrieve it back to the local machine. Three tasks, *TMPred, BLAST,* and *Promoter*, need to use the protein sequence retrieved by the task *NCBISearch.* Task *TMPred* marks the transmembrane regions of the sequence. Task *Promoter* predicts the promoter regions of the sequence by using a neural network. And task *BLAST* searches the sequence database to find sequences that are similar to the target sequence on structure. The outputs of the tasks *TMPred* and *Promoter* will be displayed directly to the user. The process following the task *BLAST*, which collects similar sequences, is more complex than the other straight forward display tasks. The output of task *BLAST* is a set of similar sequences associate with their similarity criteria value, called e-value (Altschul et al., 1990). This set of sequence will be fed to the next step – task *Threshold.* Task *Threshold* filters the set of sequences according to their e-values. That is, sequences with e-values lower than the threshold will be kept while the others are discarded. The qualified sequences are then sent one-by-one to the task *LitInfo* . The function of task *LitInfo* is to search each input sequence against the NCBI PubMed database (NCBI), extract the literature-oriented information from the resulting web pages, then send back the literature-oriented information to the *Threshold* task. After collecting all of the literature-related information for all similar sequences, task *Threshold* puts them together into a HTML file and sends the file to the *Display* task (for display to the user). Note that the sequence is sent one-by-one from task *Threshold* to task *LitInfo*. Furthermore, the literature-related information is collected one-by-one from the *LitInfo* task for each sequence. Thus a loop structure is employed here to address the iteration issue.

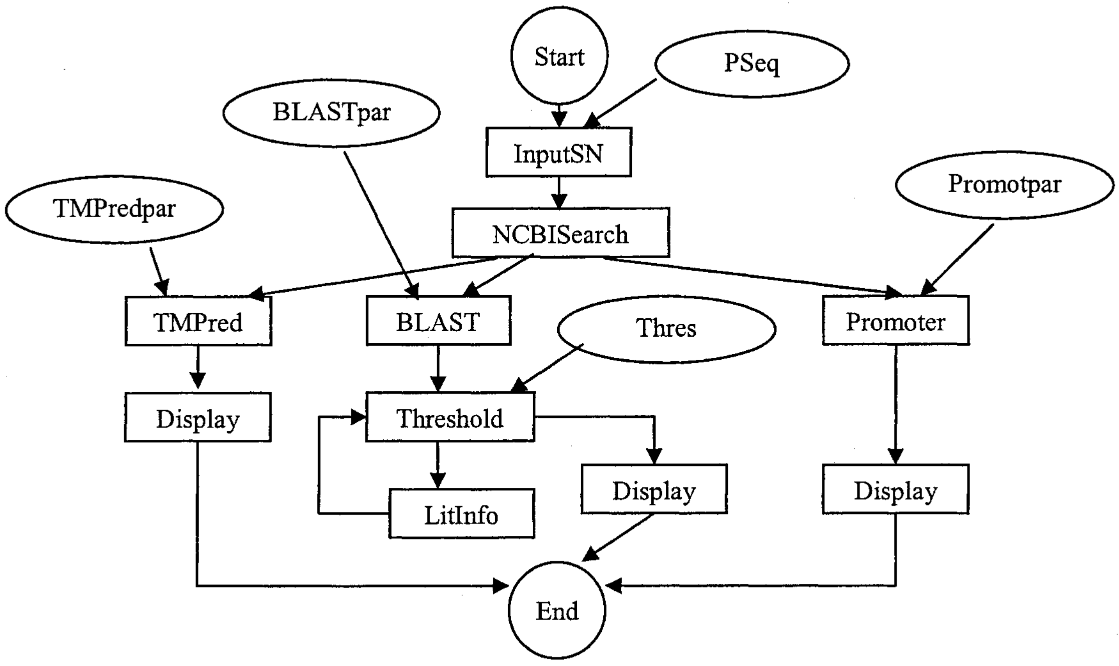


Figure 56. Data/Program Chart for Protein Function and Expression.

According to the Data/Program Chart, workflow designers, with their workflow modeling expertise, can draw the Petri net model for this workflow process with the Petri net-based user interface. The Petri net model is displayed in Figure 57. Transitions *InputTMPredpar, InputBLASTpar, InputThres,* and *InputPromoterpar* are added into the model to explicitly represent the parameter input tasks. The other part of the Petri net model is a straightforward mapping between the Data/Program Chart tasks and the Petri net transitions and places, except for the iteration part for literature information searching. To describe the iteration of transition *LitInfo*, an OrSplit place is used to dispatch the

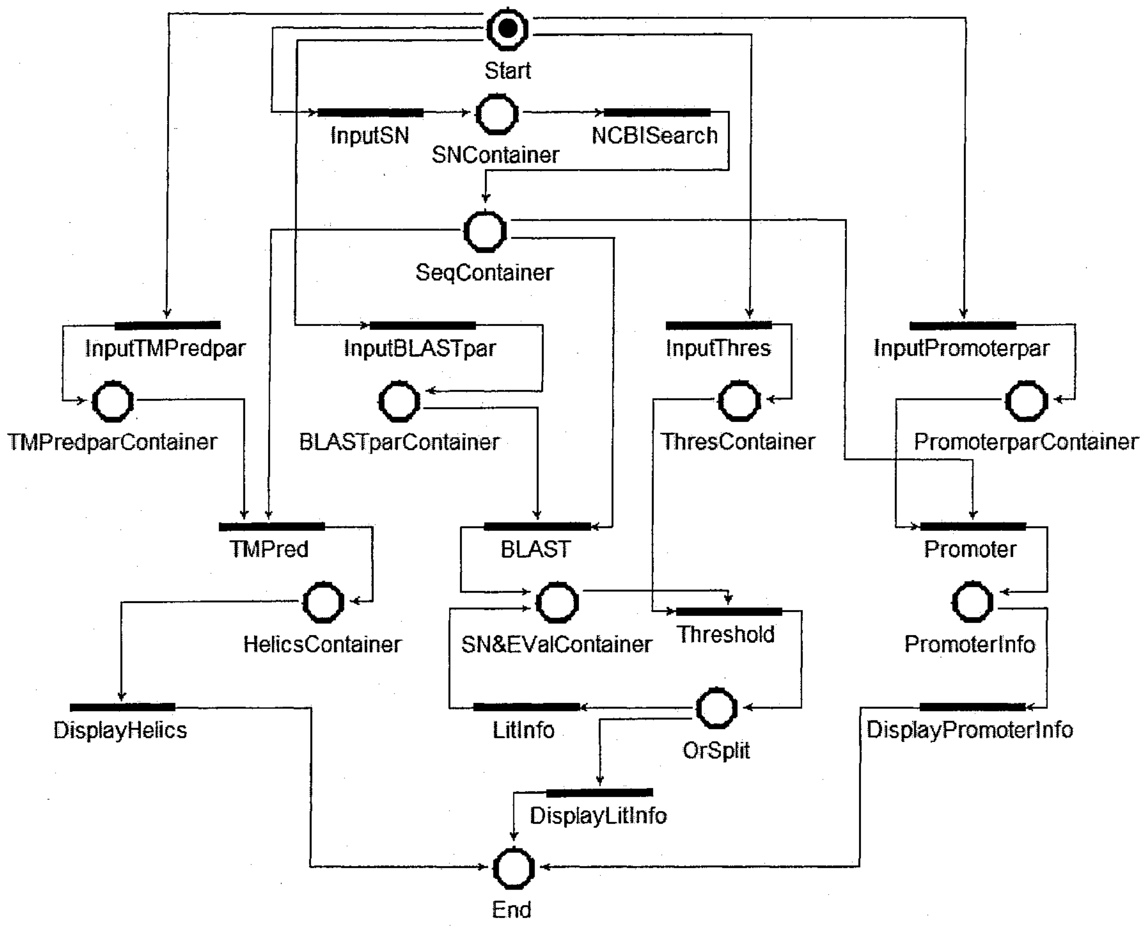


Figure 57. Petri Net Model for Protein Function and Expression Workflow.

output of transition *Threshold* to appropriate transition (*LitInfo* or *DisplayLitInfo*) based on the features of the output. If the output is a sequence, it will be directed to transition *LitInfo* by the OrSplict place. If the output is a text file containing all of the literature information for all the sequences, it will be forwarded directly to transition *DisplayLitInfo*. The mechanism of using OrSplit to construct loop structure in Petri net model is described in section 5.4. Note this loop structure is hard, if not impossible, to be represented in a DAG. Data in a DAG is not explicitly expressed. So basically any processes expressed in a DAG are static processes. No dynamic decisions based on data features can be made during the execution. It is the Petri net model that enables the modeling of iteration in this workflow case.

After been input with the Petri net workflow model, the Graphic User Interface can, with the help of the workflow designer, translate this workflow model into GFDL script as shown in Figure 58.

```
Set x=NCBISearch(InputSN(PSeq));
Set DisplayHelics(TMPred(x,InputTMPredpar()));
Set y=BLAST(x,InputBLASTpar());
Set z=NULL;
Set z=LitInfo(l) While l=Threshold(y,z,Thres);
Set DisplayLitInfo(l);
Set DisplayPromoterInfo(Promoter(x,InputPromoterpar()));
```

Figure 58. GFDL Script for Protein Function and Expression Workflow.

The translation procedure goes as follows. When the interpreter first analyzes the whole model, it detects that the transitions *InputSN* and *NCBISearch* are in a sequence structure with all the other parts of the process. So the interpreter generates the sentence

Set x=NCBISearch(InputSN(PSeq)); (the variable name *x* is randomly selected) and goes recursively into the exploration of the rest part (lower part in Figure 57) of the process model. After going further into the lower part, the interpreter detects that the lower part is consisted of three parallel routines. So the sentences *Set DisplayHelics(TMPred(x, InputTMPredpar()));* and *Set DisplayPromoterInfo(Promoter(x,Input-Promoterpar()));* are generated and the interpreter goes further into another parsing recursion. In this recursion, the most inner structures are revealed: a Begin-controlled loop structure is embedded into a sequence structure. Thus the interpreter can generate the last part of the GFDL script. That is, the code for the first step in the sequence structure is generated as sentence *Set y=BLAST(x,InputBLASTpar());* and the following loop structure is interpreted as GFDL scripts shown in Figure 59.

```
Set z=NULL;
Set z=LitInfo(l) While l=Threshold(y,z,Thres);
Set DisplayLitInfo(l);
```

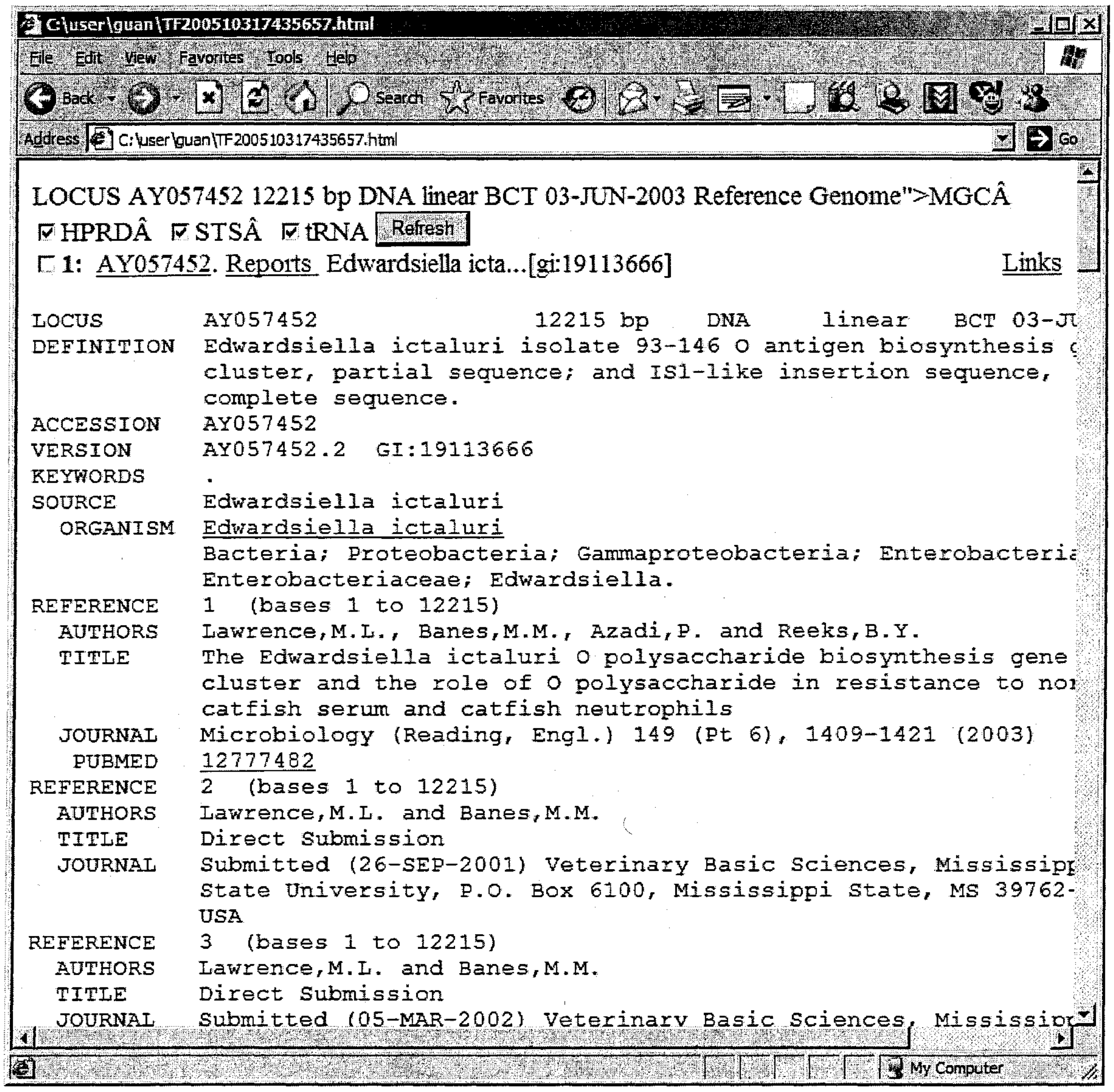
Figure 59. The Loop Structure in Protein Function and Expression Workflow.

8.2.3 *Execution*

Before the GFDL script can be executed, all the data and programs used in the sentences must be registered with the Grid-Flow system. There is only one data *PSeq* need to be registered by the user. All the other data are generated as intermediate variables during the workflow execution and thus automatically registered by the system. On the contrary, all the programs used in the script needs to be manually registered by the user. Programs *InputSN, InputTMPredpar, InputBLASTpar, Threshold, InputPromoter-par, DisplayHelics, DisplayLitInfo,* and *DisplayPromoterInfo* are all local programs located in the local operating system. Programs *NCBISearch, TMPred, Promoter,* and *LitInfo* are Internet programs which are based on webpage format CGI programs.

Program *BLAST* is different from the other programs since it is based on the G-BLAST services provided by the WebRun system. This G-BLAST web service locates on the Titanic cluster of the High Performance Computing Lab (HPCL) in UAB. It provides the service of searching single genomic sequence against predefined sequence databases. Since the G-BLAST service has a unique interface definition, it is easy to replace this G-BLAST service with other services without changing any GFDL code. The G-BLAST service could also be extended to other service providers for better performance if more computing power should be desired. This service-oriented architecture enhances the flexibility, extensibility, and performance of the Grid-Flow system.

Most of the workflow process can be automatically executed by the workflow engine. The only part that requires human interference is the matching of input/output data between programs (shown in Figure 55). The workflow process will finally present users with three types of data. They are information about sequence transmembrane regions, sequence promoter regions, and the literature-related information for the sequence in the genomic research area. Figure 53 offers a good example of the information for sequence transmembrane regions, generated by the workflow process. Figure 60 and Figure 61 display the literature-oriented information and the result of promoter prediction of a particular protein sequence.



C:\user\guan\TF200510317435657.html

File Edit View Favorites Tools Help

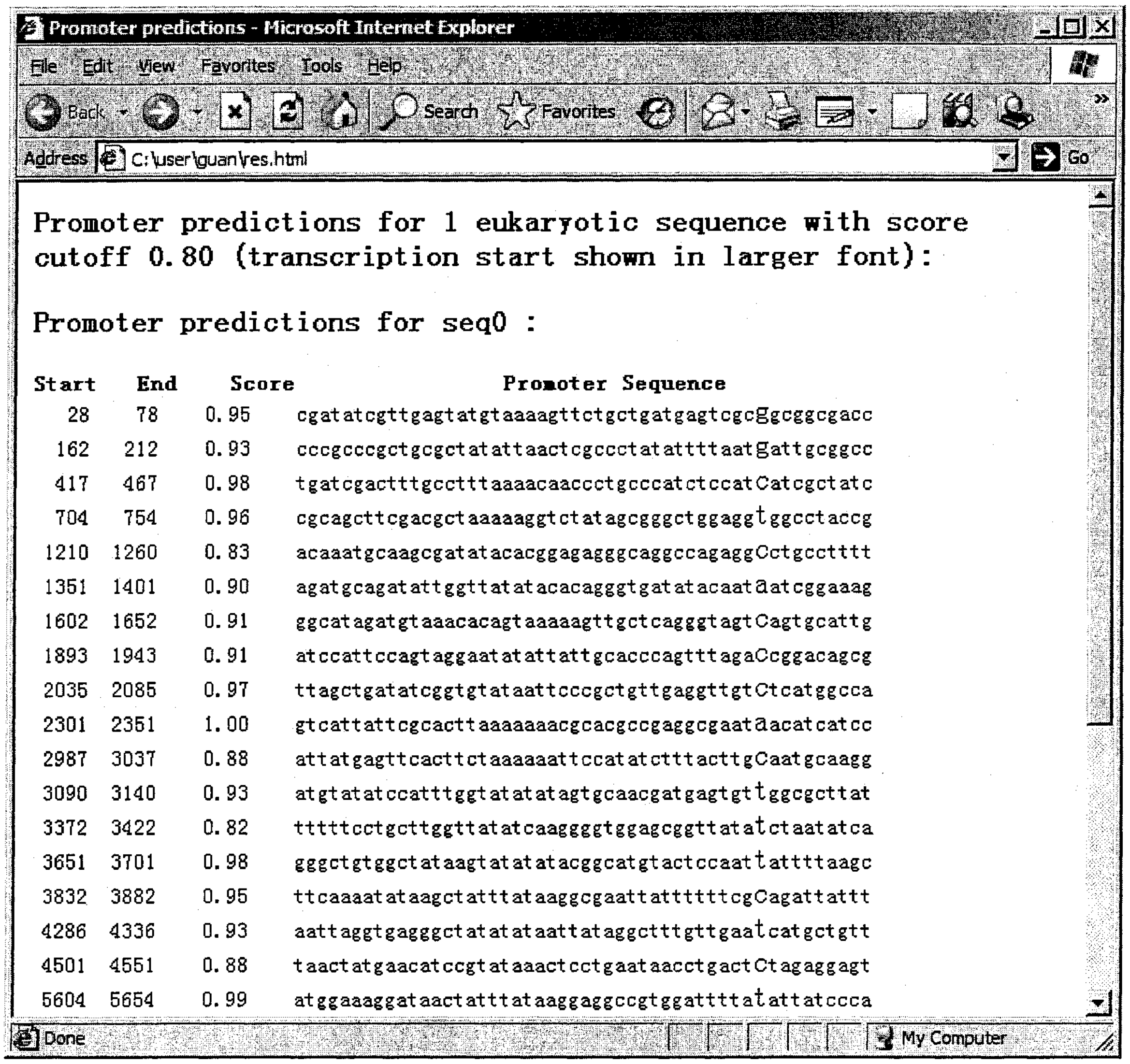
Address C:\user\guan\TF200510317435657.html

LOCUS AY057452 12215 bp DNA linear BCT 03-JUN-2003 Reference Genome">MGCÂ
☑ HPRDÂ ☑ STSÂ ☑ tRNA Refresh
☐ 1: AY057452. Reports Edwardsiella icta...[gi:19113666] Links

```
LOCUS       AY057452                 12215 bp    DNA     linear   BCT 03-JU
DEFINITION  Edwardsiella ictaluri isolate 93-146 O antigen biosynthesis g
            cluster, partial sequence; and IS1-like insertion sequence,
            complete sequence.
ACCESSION   AY057452
VERSION     AY057452.2  GI:19113666
KEYWORDS    .
SOURCE      Edwardsiella ictaluri
  ORGANISM  Edwardsiella ictaluri
            Bacteria; Proteobacteria; Gammaproteobacteria; Enterobacteria
            Enterobacteriaceae; Edwardsiella.
REFERENCE   1  (bases 1 to 12215)
  AUTHORS   Lawrence,M.L., Banes,M.M., Azadi,P. and Reeks,B.Y.
  TITLE     The Edwardsiella ictaluri O polysaccharide biosynthesis gene
            cluster and the role of O polysaccharide in resistance to no
            catfish serum and catfish neutrophils
  JOURNAL   Microbiology (Reading, Engl.) 149 (Pt 6), 1409-1421 (2003)
   PUBMED   12777482
REFERENCE   2  (bases 1 to 12215)
  AUTHORS   Lawrence,M.L. and Banes,M.M.
  TITLE     Direct Submission
  JOURNAL   Submitted (26-SEP-2001) Veterinary Basic Sciences, Mississipp
            State University, P.O. Box 6100, Mississippi State, MS 39762-
            USA
REFERENCE   3  (bases 1 to 12215)
  AUTHORS   Lawrence,M.L. and Banes,M.M.
  TITLE     Direct Submission
  JOURNAL   Submitted (05-MAR-2002) Veterinary Basic Sciences, Mississipp
```

My Computer

Figure 60. Literatures Information of a Protein Sequence.



```
Promoter predictions for 1 eukaryotic sequence with score
cutoff 0.80 (transcription start shown in larger font):

Promoter predictions for seq0 :

Start   End    Score                  Promoter Sequence
   28    78    0.95   cgatatcgttgagtatgtaaaagttctgctgatgagtcgcggcggcgacc
  162   212    0.93   cccgcccgctgcgctatattaactcgccctatattttaatgattgcggcc
  417   467    0.98   tgatcgactttgcctttaaaacaaccctgcccatctccatCatcgctatc
  704   754    0.96   cgcagcttcgacgctaaaaaggtctatagcgggctggaggtggcctaccg
 1210  1260    0.83   acaaatgcaagcgatatacacggagagggcaggccagaggCctgcctttt
 1351  1401    0.90   agatgcagatattggttatatacacagggtgatatacaataatcggaaag
 1602  1652    0.91   ggcatagatgtaaacacagtaaaaagttgctcagggtagtCagtgcattg
 1893  1943    0.91   atccattccagtaggaatatattattgcacccagtttagaCcggacagcg
 2035  2085    0.97   ttagctgatatcggtgtataattcccgctgttgaggttgtCtcatggcca
 2301  2351    1.00   gtcattattcgcacttaaaaaaacgcacgccgaggcgaataacatcatcc
 2987  3037    0.88   attatgagttcacttctaaaaaattccatatctttacttgCaatgcaagg
 3090  3140    0.93   atgtatatccatttggtatatatagtgcaacgatgagtgttggcgcttat
 3372  3422    0.82   tttttcctgcttggttatatcaaggggtggagcggttatatctaatatca
 3651  3701    0.98   gggctgtggctataagtatatatacggcatgtactccaattattttaagc
 3832  3882    0.95   ttcaaaatataagctatttataaggcgaattattttttcgCagattattt
 4286  4336    0.93   aattaggtgagggctatatataattataggctttgttgaatcatgctgtt
 4501  4551    0.88   taactatgaacatccgtataaactcctgaataacctgactCtagaggagt
 5604  5654    0.99   atggaaaggataactatttataaggaggccgtggattttatattatccca
```

Figure 61. Promoter Prediction for Protein Sequence.

8.3 Discussion

These two workflow use cases offer good demonstrations of the benefits that WFMS can bring to bioinformatics research. Compared with analyzing the protein trans-membrane regions in 10 steps without workflow (described in section 8.1), a user can easily go through the first workflow process by clicking several buttons and monitoring the execution under the control of WFMS. The significant number of functions provided by input/output tools, such as *InputSN* and *Display*, greatly facilitate the interaction between workflow users and the management system. Users are now shielded from the complicated implementation details of the data transfer and transformation. They need not even bother to learn the usage of various web interfaces for different web programs. A uniform interface for data input and output has been provided by Grid-Flow for users to communicate with the system efficiently. The flexible flow controls supported by the GFDL also enable users to describe advanced streamlining workflow processes. In addition, the powerful data and program integration engine, via novel technologies like Grid/web services, extends the scope of workflow execution over the traditional system-level boundary. More workflows can now be executed in heterogeneous and distributed environments, which fully exploit the power of distributed computation.

With the help of the WFMS, users can reduce the amount of errors and mistakes they may make while doing the data analysis manually. Current data analyses usually involve large amount of data input and output, as well as complicated procedures with numerous tasks using multiple tools. Users are prone to make mistakes when facing the large amount of data and various tools. WFMS reduce the users' burden by providing a systematic way to handle the data, tools, and procedures. As shown in the second workflow use case, the whole workflow process is predefined and recorded in GFDL scripts.

Users can check the workflow definition thoroughly beforehand based on the design of the data analysis protocol. Once the workflow is defined, WFMS would strictly execute this workflow without further human interaction. Thus, mistakes and errors introduced by human operations can mostly be avoided in WFMS. On the other hand, the complete definition of workflow process can also help users on bookkeeping and communicating their works and ideas with their colleagues.

From the perspective of performance, WFMS can improve the throughput of data analyses. The author of this dissertation made a comparison of the time used by an experienced user (manually) and WFMS to complete the data analysis defined in section 8.2.1. The user took 125 minutes to complete the whole procedure, including the waiting time of program execution and the time used for copy-and-paste data between programs. Grid-Flow spent less than two second to interpret the GFDL script, and about 67 minutes to finish the whole data analysis on the same input sequence. The improvement of performance would be more significant if a batch of sequences need to be analyzed: Grid-Flow system can parallelize the processing of multiple sequences and dramatically reduce the time used by slow human input and output operations, like copy-and-paste of data between programs.

This chapter described two workflow use cases performed with the Grid-Flow system and discussed briefly the benefits of using WFMS in bioinformatics research. Each workflow use case is demonstrated with its definition, design, implementation, and execution. The benefits of using Grid-Flow system are described based on three aspects: usability, reliability, and performance. The next chapter will summarize the dissertation and offer conclusions of the Grid-Flow research and its practice.

9 SUMMARY AND CONCLUSIONS

The Grid-Flow system is implemented as a proof-of-concept prototype to investigate the feasibility of applying Petri net modeling and Grid computing techniques to the traditional workflow management system in order to support design and execution of advanced scientific workflows in Grid environment. Briefly, the Grid-Flow system can be divided into three subsystems with their relatively individual functionalities. These subsystems are as follows: workflow modeling subsystem, which uses Petri net modeling approach to help users define workflow processes and monitor the execution; workflow execution subsystem, which employs GFDL to convey process definitions and executes workflows by scheduling appropriate data and programs; and data/program integration subsystems, which provides a uniform interface to data access and program invocation. In this chapter, each individual subsystem is evaluated and summarized, followed with an evaluation of the whole Grid-Flow system and the conclusion of the whole dissertation.

9.1 Workflow Modeling Subsystem

The main functionality of workflow modeling subsystem is to model workflow processes, or in other words, to help WFMS users to design workflows. The Petri net is used in the Grid-Flow system as the modeling approach. Compared with a DAG formulation, Petri net is a more powerful modeling approach that has broader coverage on workflow structures. Petri nets can support not only the four common routines used in workflow process, but also elaborate flow control structures, like the implicit OrSplit. In addition, Petri net modeling has support for the explicit expression of data and state in workflow process, which enables the data feature-based flow control and workflow execution monitoring, bookkeeping, and fault recovering. The Grid-Flow system equipped with Petri net modeling approach thus has the ability to model advanced workflow processes and perform more subtle controls over the workflow cases.

The instantiation of Petri net modeling accomplished in this dissertation, as well as the user interface of Grid-Flow system, is based on GME. GME provides many useful features to support the model-driven design and computation. First, the meta-model of the Petri net, which is the foundation of Petri net domain models, is defined in GME. Second, after setting up the meta-model, a user can build Petri net domain models directly within GME to model the workflow process. Third, GME supports the definition of checking constraints for the domain model. This feature can help users check the validity of the workflow definition and avoid many mistakes during the modeling. Fourth, an interpreter has been implemented within GME to automatically translate the Petri net workflow definition into GFDL.

The workflow modeling subsystem also provides another modeling approach, the Data/Program Chart, for defining workflow processes. Compared with the Petri net, the Data/Program Chart is not as powerful but is easier to grasp by inexperienced users. It facilitates the communication of informal definitions of workflow processes between end-users and workflow designers.

In a summary, the workflow modeling subsystem is a powerful, agile, and convenient system supporting advanced workflow definition and modeling.

9.2 Workflow Execution Subsystem

The workflow execution subsystem provides not only a language (GFDL) to define and convey the workflow process, but also a platform to support the execution and administration of workflow cases.

GFDL is a programmable language used to describe workflow processes. It has the ability to define variables, data, programs (functional units) in the workflow process, control the data and execution flow, and support the hierarchical definition of workflow processes. Petri net models can directly be mapped to GFDL by the interpreter. Compared with other workflow languages, GFDL is a human readable, agile, light-weight, and extensible workflow description language. Experienced users can furthermore define workflow process directly with GFDL, like using a programming language. This particular feature can help to avoid the overhead of mapping between the Petri net models and GFDL scripts, therefore improve the effectiveness of the Grid-Flow system and the performance of workflows.

The workflow execution subsystem is responsible for interpreting user-defined workflow process in GFDL, and coordinating the execution of user workflows using appropriate services provided by the data/program integration subsystem. It is based on a service-oriented architecture, which is organized as communicating as services through well-defined interfaces without knowing the implementation details. The workflow execution subsystem is implemented as an assembled system of five modules; each module has its clear defined functionality. The interfaces among functional modules are well-defined. Modules communicate with each other by using services. This architecture modularizes the function of the whole subsystem and simplifies the implementation of each module.

9.3 Data/Program Integration Subsystem

Data and program integration subsystem can actually be divided into two components: the data integration component and the program integration component. Data used in scientific workflows come from different resources in diverse formats. So, the major function of data integration component is to retrieve data from different data sources, and transform data among various formats if necessary. Similarly to data, programs could also locate in various environments and have their special execution mechanisms. Program integration component covers all of these heterogeneities by providing a uniform program invocation service to Grid-Flow engine.

The data and program integration subsystem employs several strategies to handle distributed and heterogeneous scientific data and programs. The online data and program registration mechanisms provide a formal approach to record information and tools in the Grid-Flow system, and therefore Grid-Flow effectively integrates CGI programs, which can only be invoked manually in other workflow systems. Since the online CGI programs provide such abundant functions for data analysis, scientists, especially bioinformaticians, urgently need such a mechanism that can enable them to automate the streamlining analysis involving CGI programs. On the other hand, the data transformation and matching components can transform the data and link the input and output data sets together, which ease the connection between programs in workflows and facilitate setting up the data/program pipeline. All of these mechanisms help to reduce the burden of end-users therefore improve users' productivity.

WebRun system is also used in data and program integration subsystem to provide a methodology and reference model for wrapping applications into web/Grid services. It proposes a novel approach to utilize data resources and computing power which otherwise cannot be accessed. The G-BLAST services, which were established based on the architecture of the WebRun system, provide a good example of integrating distributed computing power to provide better performance and system portability, extensibility, and address implementation details.

9.4 Evaluation Based on Use Cases

Two bioinformatics workflow cases, 1) protein transmembrane region analysis and 2) protein function and expression, are modeled using this enabling workflow management system. The success of these two use cases motivates strongly that the Grid-Flow system is a non-trivial contribution to both modeling and executing scientific workflows. The two workflow processes depicted using the Petri net-based user interface illustrate that Petri nets can be used to model advanced workflow structures. In addition, the fact that one workflow model can be reused in the other workflow case presents the ability of building hierarchical workflow models in Grid-Flow. On the other hand, the real-world workflow applications demonstrate the requirements of a powerful modeling approach to model complex workflow patterns. Compared to a DAG-based formalism, Petri nets are more capable to meet such requirements. GFDL scripts used to convey workflow definitions demonstrate the usability and efficiency of the light-weighted workflow language on describing control flow structures and the data flow. Several programs located in distributed and heterogeneous environments are integrated into the Grid-Flow system as workflow tasks. The effectiveness and utility of the data and program integration component are therefore motivated.

9.5 Conclusions

In a summary, the GridFlow system is a non-trivial contribution to scientific workflow modeling and to orchestrating programs in Grid computing environment. With prototyping of Grid-Flow, as well as running real-world workflow applications on it, one can reasonably accept that the hypothesis of this dissertation is demonstrated. That is, it is possible to applying Petri net modeling, Internet computing, and Grid computing techniques together, to create a Grid service-based workflow management system that can model workflow process with Petri nets, and execute workflows over distributed and heterogeneous environments, without sacrificing any capability and generalizability of the workflow system. The Grid-Flow system is just such a system, constructively demonstrating the hypothesis. Furthermore, Grid-Flow system and associated experiments demonstrate that the Petri net is qualified to model scientific workflow processes, and the data and program integration component in Grid-Flow has the ability to integrate newly emergent, disparate and distributed data and computing resources to provide new capabilities for workflow applications.

This dissertation analyzes three critical issues that distinguish the utility of Grid workflow systems and provides a novel solution to address these in Grid-Flow system. A brief survey was 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 light-weight workflow language (GFDL), and integrates heterogeneous data and distributed analysis tools through a unified platform (data and program integration components). The meta-model that describes the Petri net-based workflow process in GME is a novel achievement of this work. To the best of the author's knowledge, no similar meta-model has been developed in GME (or equivalent systems) to address the modeling problem of workflow processes. Furthermore, the architecture of the Grid-Flow system comprises its workflow language GFDL, its graphical user interface, its workflow engine, its system repository, and its data and program integration mechanism; these are described in detail. Finally, with the help of two illustrative workflow examples, the feasibility of the proposal of the Grid-Flow system is motivated and further shows advantages of using the Grid-Flow system.

10 FUTURE WORK

The Grid-Flow system is an on-going project. It needs further work to adapt it to the ever-changing Grid computing environment. Possible future work may focus further on the Petri net modeling approach. As a modeling approach, Petri nets have their own limitations. As reported in (Hoheisel, 2004), the size of the Petri net will increase dramatically when the modeled workflow process becomes progressively more complex. 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 such processes 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 the Grid-Flow language with appropriate adjustments is capable of describing business-oriented workflow processes. The Grid-Flow language will become friendlier in the near future for designing and carrying out business operational patterns and protocols. Finally, optimization and fault management features should also be considered in the future design of the Grid workflow system, including reexecution of steps in the workflow based on local failure recovery, and potentially global techniques for recovery from failure that are weak enough to work fast enough (not limit scalability).

LIST OF REFERENCES

Aalst, W. v. d. (1994). Putting Petri nets to work in industry. *Computers in Industry, 25*(1), 45-54.

Aalst, W. v. d. (1996, Nov.). *Three good reasons for using a Petri-net-based workflow management system.* Paper presented at the the International Working Conference on Information and Process Integration in Enterprises (IPIC'96), Camebridge, Massachusetts.

Aalst, W. v. d. (1998). The application of Petri nets to workflow management. *The Journal of Circuits, Systems and Computers, 8*(1), 21-66.

Aalst, W. v. d., & Hee, K. v. (2002). *Workflow Management: Models, Methods, and Systems.* Cambridge, Massachusetts: The MIT Press.

Afgan, E., Velusamy, V., & Bangalore, P. (2005, Feb. 14-16). *Grid Resource Broker with Application Profiling and Benchmarking.* Paper presented at the European Grid Conference, Amsterdam, Netherlands.

Allcock, W., Bester, J., Bresnahan, J., Chervenak, A., Liming, L., Meder, S., et al.*Gridftp protocol specification.* Retrieved Sepetember, 2002, from http://www.globus.org/research/papers/GridftpSpec02.doc

Allen, M. P., & Tildesley, D. J. (1987). *Computer Simulation of Liquids.* New York, NY: Oxford University Press.

Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludaescher, B., & Mock, S. (2004, March). *Kepler: Towards a Grid-Enabled System for Scientific Workflows.* Paper presented at the Workflow in Grid Systems Workshop in GGF10, Berlin, Germany.

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). Basic local alignment search tool. *Journal of Molecular Biology, 215*(3), 403-410.

Ames, C. K., Burleigh, S. C., & Mitchell, S. J. (1997, November 16-19). *A web based enterprise workflow system.* Paper presented at the the International ACM SIGGROUP Conference on Supporting Group Work: the Integration Challenge, Phoenix, Arizona.

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., et al. (2003). *Business Process Execution Language for Web Services (BPEL4WS) Spcification*

Version 1.1. Retrieved May 12th, 2004, from ftp://www6.software.ibm.com/software/developer/library/ws-bepl11.pdf

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., et al. (2003, May 05). *Specification: Business Process Execution Language for Web Services Version 1.1*. Retrieved May 31st, 2004, from http://www-106.ibm.com/developerworks/library/ws-bpel/

Ant - A Java-based Build Tool. Retrieved May 31st, 2004, from http://ant.apache.org

Arkin, A., Askary, S., Fordin, S., Jekeli, W., Kawaguchi, K., Orchard, D., et al. (2002). *Web Services Choreography Interface (WSCI) 1.0 Specification*. Retrieved May 31st, 2004, from http://wwws.sun.com/software/xml/developers/wsci/index.html

BDGP: Neural Network Promoter Prediction. Retrieved October, 2005, from http://www.fruitfly.org/seq_tools/promoter.html

Berman, F., Fox, G. C., & Hey, A. J. G. (Eds.). (2003). *Grid Computing: Making The Global Infrastructure a Reality*. West Sussex, England: John Wiley & Sons Ltd.

Bhatia, D., Burzevski, V., Camuseva, M., Fox, G., Furmanski, W., & Premchandran, G. (1997). WebFlow - A Visual Programming Paradigm for Web/Java Based Coarse Grain Distributed Computing. *Concurrency: Practice and Experience, 9*(6), 555-577.

Booch, G., Rumbaugh, J., & Jacobson, I. (1998). *The Unified Modeling Language User Guide*. Reading, Massachusetts: Addison-Wesley.

Brumfiel, G. (2002). Astronomy: The heavens at your fingertips. *Nature, 420*(6913), 262-264.

Camarda, K., He, Y., & Bishop, K. (2001, June 25-27). *A Parallel Chemical Reactor Simulation Using Cactus*. Paper presented at the Linux Clusters: the HPC Revolution, Urbana, IL.

Cannan, S. J., & Otten, G. A. M. (1992). *SQL - THE STANDARD HANDBOOK*. New York: McGraw-Hill Book Company.

Cao, J., Jarvis, S. A., Saini, S., & Nudd, G. R. (2003, May 12-15). *GridFlow: Workflow Management for Grid Computing*. Paper presented at the 3rd International Symposium on Cluster Computing and the Grid, Tokyo, Japan.

Cardoso, J., & Sheth, A. (2003). Semantic E-Workflow Composition. *Journal of Intelligent Information Systems, 21*(3), 191-225.

Casati, F., & Discenza, A. (2000, March 19-21). *Supporting Workflow Cooperation Within and Across Organizations*. Paper presented at the The 2000 ACM Symposium on Applied Computing, Como, Italy.

Chen, L., & Jamil, H. M. (2003). On Using Remote User Defined Functions as Wrappers for Biological Database Interoperability. *International Journal of Cooperative Information Systems (IJCIS), Special Issue on Data Management and Modeling Support in Bioinformatics, 12*(2), 161-195.

Christensen, E., Curbera, F., Meredith, G., & Weerawarana, S. (2001). *W3C Web Services Description Language (WSDL) 1.1*. Retrieved May 31st, 2004, from http://www.w3.org/TR/wsdl

ClustalW. Retrieved May 31st, 2004, from http://www.ebi.ac.uk/clustalw

Common Component Architecture. Retrieved May 31st, 2004, from http://www.cca-forum.org

Condor: The Directed Acyclic Graph Manager. (2003). Retrieved May 12th, 2004, from http://www.cs.wisc.edu/condor/dagman

Cybok, D. (2004, March). *A Grid Workflow Infrastructure*. Paper presented at the Workflow in Grid Systems Workshop in GGF10, Berlin, Germany.

Czarnecki, K., & Eisenecker, U. (2000). *Generative Programming: Methods, Tools, and Applications* (1st ed.): Addison-Wesley Pub. Co.

DAGMan (Directed Acyclic Graph Manager). (2002). Retrieved May 31st, 2004, from http://www.cs.wisc.edu/condor/dagman

Darling, A. E., Carey, L., & Feng, W.-c. (2003, June, 2003). *The Design, Implementation, and Evaluation of mpiBLAST*. Paper presented at the ClusterWorld Conference & Expo in conjunction with the 4th International Conference on Linux Clusters: The HPC Revolution 2003, San Jose, CA.

Davulcu, H., Kifer, M., Ramakrishnan, C. R., & Ramakrishnan, I. V. (1998). *Logic Based Modeling and Analysis of Workflows*. Paper presented at the ACM Symposium on Principles of Databases Systems.

Deelman, E., Blythe, J., Gil, Y., & Kesselman, C. (2003). Workflow Management in GriPhyN. In J. Nabrzyski, J. M. Schopf & Jan Weglarz (Eds.), *Grid Resource Management: State of the Art and Future Trends*: Kluwer Publishing.

Dogac, A., Leonid, K., Ozsu, M. T., & Sheth, A. (Eds.). (1998). *Workflow Management Systems and Interoperability* (Vol. 164): Springer Verlag.

Dozsa, G., Kacsuk, P., Lovas, R., Podhorszki, N., & Drotos, D. (2004, March). *P-GRADE: A Graphical Environment to Create and Execute Workflows in Various Grids*. Paper presented at the Workflow in Grid Systems Workshop in GGF10, Berlin, Germany.

Earth System Grid. Retrieved Nov. 27, 2005, from https://www.earthsystemgrid.org/

Erwin, D. W., & Snelling, D. F. (2001). UNICORE: A Grid Computing Environment. *Lecture Notes in Computer Science, 2150*, 825-834.

Extensible Markup Language (XML). Retrieved May 31st, 2004, from http://www.w3.org/XML/

Forum, G. G. (2004). *The JSDL Specification*. Retrieved October, 2004, from https://forge.gridforum.org/projects/jsdl-wg/document/draft-ggf-jsdl-spec/en/8

Foster, I. (2002, July 22). *What is the Grid? A Three Point Checklist*. Retrieved May 31st, 2004, from http://www.gridtoday.com/02/0722/100136.html

Foster, I., & Kesselman, C. (1999). The Globus toolkit. In I. Foster & C. Kesselman (Eds.), *The Grid: Blueprint for a New Computing Infrastructure* (pp. 259--278). San Francisco, California: Morgan Kaufmann.

Foster, I., Kesselman, C., Nick, J., & Tuecke, S. (2002). *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. Retrieved May 31st, 2004, from http://www.globus.org/research/papers/ogsa.pdf

Foster, I., Kesselman, C., Tsudik, G., & Tuecke, S. (1998). *A Security Architecture for Computational Grids*. Paper presented at the 5th ACM conference on Computer and Communications Security, San Francisco, California.

Fraunhofer Resource Grid. Retrieved May 31st, 2004, from http://www.fhrg.fhg.de

Functional Bioinformatics System. Retrieved Aug. 23, 2003, from http://www.ppddiscovery.com/PPD_FGS_3_5.htm

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns* (1st ed.): Addison-Wesley Professional.

Genomics and Its Impact on Science and Society: A 2003 Primer. (2003). Human Genome Program, U.S. Department of Energy.

Global Grid Forum. Retrieved August, 2003, from http://www.ggf.org

The Globus Alliance. Retrieved October, 2005, from http://www.globus.org/

The Globus Resource Specification Language RSL v1.0. Retrieved May 31st, 2004, from http://www.globus.org/gram/rsl_spec1.html

Goble, C. A., Paton, N. W., Stevens, R., Baker, P. G., Ng, G., Peim, M., et al. (2001). Transparent Access to Multiple Bioinformatics Information Sources. *IBM Systems Journal, 40*(2).

Graham, G. E., Evans, D., & Bertram, I. (2003, March 24-28). *McRunjob: A High Energy Physics Workflow Planner for Grid Production Processing.* Paper presented at the Computing in High Energy and Nuclear Physics, La Jolla, California.

The GridFTP Protocol and Software. Retrieved May 31st, 2004, from http://www.globus.org/datagrid/gridftp.html

GriPhyN - Grid Physics Network. Retrieved Nov. 27, 2005, from http://www.griphyn.org/

Guan, Z., & Jamil, H. M. (2003, March 10-12). *Streamlining Biological Data Analysis Using BioFlow.* Paper presented at the the Third IEEE Symposium on Bioinformatics and Bioengineering (BIBE03), Bethesda, Maryland.

Guan, Z., Liu, Y., Velusamy, V., & Bangalore, P. V. (2004). *WebRun: A Unified Platform Supporting Grid Computing Environment* (Technical Report No. UABCIS-TR-2004-1404-1): Department of Computer and Information Sciences, University of Alabama at Birmingham.

Gupta, A., Ludascher, B., & Martone, M. E. (2002, October). *Registering Scientific Information Sources for Semantic Mediation.* Paper presented at the 21st International Conference on Conceptual Modeling, Tampere, Finland.

Harary, F. (1995). *Graph Theory*: Addison Wesley Publishing Company.

Hayes, K., & Lavery, K. (1991). *Workflow management software: the business opportunity* (Technical report). London: Ovum Ltd.

Hernandez, F. (2004). *Domain Specific Models and the Globus Toolkit* (Technical Report No. UABCIS-TR-2004-0504-1): Department of Computer and Information Sciences, University of Alabama at Birmingham.

High-Level Java Interface to GME -- Users Manual Version 1.0. (2004). Institute for Software Integrated Systems, Vanderbilt University.

Hoheisel, A. (2004, March). *User Tools and Languages for Graph-based Grid Workflows.* Paper presented at the Workflow in Grid Systems Workshop in GGF10, Berlin, Germany.

Hull, R., Llirbat, F., Simon, E., Su, J., Dong, G., Kuman, B., et al. (1999, February 23-25). *Declarative Workflows that Support Easy Modification and Dynamic Browsing.* Paper presented at the International Joint Conference on Work Activities Coordination and Collaboration.

Jacko, J. A., & Sears, A. (Eds.). (2003). *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications.* Mahwah, NJ: Lawrence Erlbaum Associates.

Jacob, J. C., Williams, R., Babu, J., Djorgovski, S. G., Graham, M. J., Katz, D. S., et al. (2004, Oct. 24-27). *Grist: Grid Data Mining for Astronomy.* Paper presented at the Astronomical Data Analysis Software & Systems (ADASS) XIV, Pasadena, CA.

Jensen, K. (1997). *Coloured Petri nets: basic concepts, analysis methods and practical use* (Vol. 3). Berlin, Germany: Springer-Verlag.

Job Submission Description Language (JSDL) Specification. (2005). GGF.

Kang, M. H., Park, J. S., & Froscher, J. N. (2001, May). *Access Control Mechanisms for Inter-Organizational Workflow.* Paper presented at the the 6th ACM Symposium on Access Control Models and Technologies, Chantilly, Virginia.

Karsai, G., Maroti, M., Ledeczi, A., Gray, J., & Sztipanovits, J. (2004). Composition and Cloning in Modeling and Meta-Modeling. *IEEE Transactions on Control System Technology (special issue on Computer Automated Multi-Paradigm Modeling),* 263-278.

Kitano, H. (2002). Computational systems biology. *Nature, 420*(6912), 206-210.

Kochut, K. J., Arnold, J., Sheth, A., Miller, J., Kraemer, E., Arpinar, B., et al. (2002). IntelliGEN: A Distributed Workflow System for Discovering Protein-Protein Interactions. *Special Issue on Bioinformatics, International Journal on Distributed and Parallel Database.*

Koulopoulos, T. M. (1995). *The workflow imperative.* New York: Van Nostrand Reinhold.

Krishnan, S., Bramley, R., Gannon, D., Govindaraju, M., Alameda, J., Alkire, R., et al. (2001, Nov. 10-16). *The XCAT Science Portal.* Paper presented at the Supercomputing (SC2001), Danver, Colorado.

Krishnan, S., Bramley, R., Gannon, D., Govindaraju, M., Indurkar, R., Slominski, A., et al. (2001). *The XCAT Science Portal.* Paper presented at the SC 2001, Denver.

Krishnan, S., Wagstrom, P., & Laszewski, G. v. (2002, July, 19). *GSFL: A Workflow Framework for Grid Services.* Retrieved May 31st, 2004, from http://www.globus.org/cog/papers/gsfl-paper.pdf

Laszewski, G. v., Amin, K., Hategan, M., Zaluzec, N. J., Hampton, S., & Rossi, A. (2004, Jan. 5-8). *GridAnt: A Client-Controllable Grid Workflow System.* Paper presented at the 37th Hawaii International Conference on System Science, Island of Hawaii, Big Island.

Laszewski, G. v., Foster, I., Gawor, J., & Lane, P. (2001). A Java Commodity Grid Kit. *Concurrency and Computation: Practice and Experience, 13*(8-9), 643-662.

Lawerence, M. L., Banes, M. M., & Azadi, P. (2003). *The Edwardsiella Ictaluri O Polysaccharide Biosynthesis Gene Cluster: Correlation between Predicted Enzyme Functions and O Polysaccharide Composition.*Unpublished manuscript, Mississippi State University.

Leach, A. R., & Gillet, V. J. (2003). *An Introduction to Chemoinformatics.* Dordrecht, Netherlands: Springer.

Ledeczi, A., Bakay, A., Maroti, M., Volgyesi, P., Nordstrom, G., Sprinkle, J., et al. (2001). Composing Domain-Specific Design Environments. *IEEE Computer*, 44-51.

Ledeczi, A., Davis, J., Neema, S., & Agrawal, A. (2003). Modeling Methodology for Integrated Simulation of Embedded Systems. *ACM Transactions on Modeling and Computer Simulation*, 82-103.

Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., Thomason, C. I., et al. (2001, May 17). *The Generic Modeling Environment.* Paper presented at the Workshop on Intelligent Signal Processing, Budapest, Hungary.

Leymann, F. (2001). *Web Services Flow Language (WSFL 1.0).* Retrieved May 31st, 2004, from http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

Lomet, D. B., & Weikum, G. (1998, June 2-4). *Efficient and Transparent Application Recovery in Client-Server Information Systems.* Paper presented at the ACM SIGMOD International Conference on Management of Data, Seattle, WA.

Lorch, M., & Kafura, D. (2002, May 21-24). *Symphony - A Java-based Composition and Manipulation Framework for Computational Grids.* Paper presented at the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2002), Berlin, Germany.

McCann, K. M., Yarrow, M., DeVivo, A., & Mehrotra, P. (2004, March). *ScyFlow: An Environment for the Visual Specification and Execution of Scientific Workflows.* Paper presented at the Workflow in Grid Systems Workshop in GGF10, Berlin, Germany.

McCune, D., Parashar, M., Beck, M., Klasky, S., Bhat, V., & Atchley, S. (2004, November). *High Performance Threaded Data Streaming for Large Scale Simulations.* Paper presented at the 5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, PA.

Microsoft. *Visual Basic Language Reference: Shell Function.* Retrieved October 2, 2005, from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vblr7/html/vafctshell.asp

Mohan, C. (1998, Spetember 7-10). *Workflow Management in the Internet Age.* Paper presented at the Second East European Symposium, ADBIS'98, Poznan, Poland.

Naming and Addressing: URIs, URLs, ... Retrieved October, 2005, from http://www.w3.org/Addressing/

NCBI.*Entrez PubMed.* Retrieved Oct. 28, 2005, from http://www.pubmed.gov

NCBI. (2004, November 15). *NCBI BLAST.* Retrieved 4/21, 2005, from http://www.ncbi.nlm.nih.gov/BLAST/

NCBI Website. Retrieved August, 2003, from http://www.ncbi.nlm.nih.gov/

Neema, S., Bapty, T., Gray, J., & Gokhale, A. (2002, October 6-8). *Generators for Synthesis of QoS Adaptation in Distributed Real-Time Embedded Systems.* Paper presented at the First ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering (GPCE '02), Pittsburgh, PA.

Novotny, J., Tuecke, S., & Welch, V. (2001, August). *An Online Credential Repository for the Grid: MyProxy.* Paper presented at the The Tenth International Symposium on High Performance Distributed Computing (HPDC-10).

Oinn, T., Addis, M., Ferris, J., Marvin, D., Greenwood, M., Carver, T., et al. (2004, March). *Taverna, Lessons in Creating a Workflow Environment for the Life Sciences.* Paper presented at the Workflow in Grid Systems Workshop in GGF10, Berlin, Germany.

Papakonstantinou, Y., Garcia-Molina, H., & Widom, J. (1995). *Object Exchange across Heterogeneous Information Sources.* Paper presented at the 11th Conference on Data Engineering, Taipei, Taiwan.

Peterson, J. L. (1977). Petri Nets. *ACM Computing Surveys, 9*(3), 223-252.

The Ptolemy II Project. (1996). Retrieved May 31st, 2004, from http://ptolemy.eecs.berkeley.edu/ptolemyII

Ramakrishnan, R., & Gehrke, J. (2003). *Database Management System* (3rd ed.). New York, NY: The McGraw-Hill Companies.

Readseq: Read and reformat biosequences. Retrieved August, 2003, from http://iubio.bio.indiana.edu/soft/molbio/readseq/java/

Shields, M., & Taylor, I. (2004, March). *Programming Scientific and Distributed Workflow with Triana Services.* Paper presented at the Workflow in Grid Systems Workshop in GGF10, Berlin, Germany.

Sotomayor, B. (2004). *Chapter 3. Writing Your First Grid Service in 5 Simple Steps.* Retrieved 6/1/2005, 2005, from http://gdp.globus.org/gt3-tutorial/multiplehtml/ch03.html

Stein, L., Rozen, S., & Goodman, N. (1995). *Managing Laboratory Workflow with LABBASE.* Paper presented at the the 1994 Conference on Computers in Medicine (CompMed94).

Stevens, R., Goble, C., Horrocks, I., & Bechhofer, S. (2002). OILing the Way to Machine Understandable Bioinformatics Resources. *IEEE Transactions on Information Technology in Biomedicine, 6*(2), 129-134.

Swofford, D. L. (2002). PAUP*. Phylogenetic Analysis Using Parsimony (*and Other Methods) (Version Version 4). Sunderland, Massachusetts: Sinauer Associates.

Systinet.*Systinet Products Overview*. Retrieved May 31st, 2004, from http://www.systinet.com/products/overview

Sztipanovits, J. (2002). Generative Programming for Embedded Systems. *Keynote Address: Generative Programming and Component Engineering (GPCE), Springer-Verlag LNCS 2487*, 32-49.

TeraGrid. Retrieved Nov. 27, 2005, from http://www.teragrid.org/

Thatte, S. (2001). *XLANG: Web Services for Business Process Design*. Retrieved May 31st, 2004, from http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm

Thompson, R. D., & Perry, A. (Eds.). (1997). *Applied Climatology: Principles and Practice*. London, England: Routledge.

TMpred - Prediction of Transmembrane Regions and Orientation. Retrieved May 31st, 2004, from http://www.ch.embnet.org/software/TMPRED_form.html

Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S., Kesselman, C., et al.*Open Grid Services Infrastructure (OGSI)*. Retrieved October, 2005, from http://www-unix.globus.org/toolkit/draft-ggf-ogsi-gridservice-33_2003-06-27.pdf

TurboBench Workflow System. Retrieved Aug. 23, 2003, from http://www.turbogenomics.com/products/turbobench_example.html

uberTOOL. Retrieved Aug. 23, 2003, from http://www.science-factory.com/ubertool.html

Vossen, G., & Weske, M. (1999, June 1-3). *The WASA2 Object-oriented Workflow Management System.* Paper presented at the ACM SIGMOD International Conference on Management of Data, Philadephia, Pennsylvania.

W3C. (2003). *SOAP Version 1.2 Part 1: Messaging Framework.*

Warmer, J., & Kleppe, A. (2003). *The Object Constraint Language Second Edition, Getting Your Models Ready for MDA* (Second Edition ed.). Boston: Addison-Wesley.

Web Services Conversation Language (WSCL 1.0). (2002). Retrieved May 31st, 2004, from http://www.w3.org/TR/wscl10/

Weske, M. (1999, June). *Workflow Management through Distributed and Persistent CORBA Workflow Objects.* Paper presented at the 11th Int. Conf. on Advanced Information Systems Engineering (CAiSE), Heidelberg, Germany.

Wikipedia.*Directed Acyclic Graph*. Retrieved September 24, 2005, from http://en.wikipedia.org/wiki/Directed_acyclic_graph

Wikipedia.*Petri net*. Retrieved September 25, 2005, from http://en.wikipedia.org/wiki/Petri_net

Wikipedia.*Virtual Network Computing*. Retrieved October 8, 2005, from http://en.wikipedia.org/wiki/VNC

Workflow Management Coalition. Retrieved May 31st, 2004, from http://www.wfmc.org

Wu, J. S.-c., & Sussman, A. (2004). *Flexible Control of Data Transfers between Parallel Programs.* Paper presented at the Fifth IEEE/ACM International Workshop on Grid Computing, Pittsburgh, PA.

Zeleznik, F. J. (1968). Quasi-Newton methods for nonlinear equations. *Journal of the ACM, 15*(2), 265-271.

Zhang, S., Gu, N., & Li, S. (2004, April 5-7). *Grid workflow based on dynamic modeling and scheduling.* Paper presented at the International Conference on Information Technology: Coding and Computing (ITCC2004), Las Vegas, Nevada.

APPENDIX A

GRAMMAR OF GRID-FLOW DESCRIPTION LANGUAGE

Reserved Keywords:

Set, While, Until, When, Register, Data, Program, As, Input, Output, End, Text, Table, GridFlow, Internal_Program, OS_Program, %, ^

Syntax:

process :: = *sentence*; [*sentence*; …]

sentence :: = *set_sentence*

| *register_data_sentence*

| *register_program_sentence*

| *input_sentence*

| *output_sentence*

| *end_sentence*

set_sentence :: = **Set** *expression_set* [**While** | **Until** | **When** *expression_set*]

register_data_sentence :: = **Register Data** *data_name* **As Text** | **Table**

register_program_sentence :: =**Register Program** *program_name* **As GridFlow** | **Internal_Program** | **OS_Program**

input_sentence :: = **Input** *input_parameters_list*

output_sentence :: = **Output** *output_parameters_list*

end_sentence :: = **End**

expression_set :: = *expression* [^ | % *expression* …]

expression :: = [*variable* =]

variable |

constant |

program

program :: = *program_name*(*parameters*)

parameters :: = ε | *expression* [, *expression* …]

input_parameters_list :: = *variable* [, *variable* …]

output_parameters_list :: = *variable* [, *variable* …]

data_name :: = *variable*

program_name :: = *variable*

variable :: = A-Z | a-z [0-9 | A-Z | a-z …]

constant = A-Z | a-z [0-9 | A-Z | a-z …]

Explanation

GridFlow Description Language follows the Boyce-Codd Normal Form (BCNF) (Ramakrishnan & Gehrke, 2003). The syntax of GridFlow Description Language is explained as follows:

Italics represent meta-variables. Bold represent reserved keywords. And roman font represents terminal symbols. Particularly, ε means empty.

"^" means logic And. "%" means logic Or.

"[ ]" signifies that the language elements between the square brackets can optionally appear, but are not required. Note that these brackets are not part of the code and must not appear in the GridFlow process.

The "or" (|) symbol signifies that you may use only one of the code elements or values from the possible choices. Note that the "or" symbol is not part of the code and must not appear in the GridFlow process.

"…" symbol signifies that the code elements prior to it can repeat more than once. Note that the "…" symbol is not part of the code and must not appear in the GridFlow process.

APPENDIX B

WEBRUN SYSTEM

B.1 WebRun System Architecture

WebRun is a unified platform supporting Grid computing (Berman et al., 2003) environments. It provides users with both a web interface and a programmatic interface to non-interactive programs located on remote computing resources. The web interface, a browser and servlet implementation in WebRun, supports the finding, starting, controlling, and utilization of non-interactive application programs. The programmatic interface enabled by the web service's (Systinet) client/service model provides a Java interface for accessing the programs stored on the remote resources that are otherwise inaccessible. WebRun adopts a hierarchical architecture including three main components, the Program server, the Web Service server, and the Web server, as shown in Figure 62.

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 on the server 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) (*The Globus Resource Specification Language RSL v1.0*).

The WebRun middleware is initially centralized on one machine. It maintains a program description store, which holds the property file for each program joining the program participation. Each program profile is accessed as response to a client request, which in turn may be used to construct a job-submission form or an RSL (*The Globus Resource Specification Language RSL v1.0*) associated with a given job. In essence, for each program description, Grid services are correspondingly generated and registered with the OGSA services container. The Grid services in turn provide programmatic interface for servlets/JSP (Java Server Pages) or programmatic users.

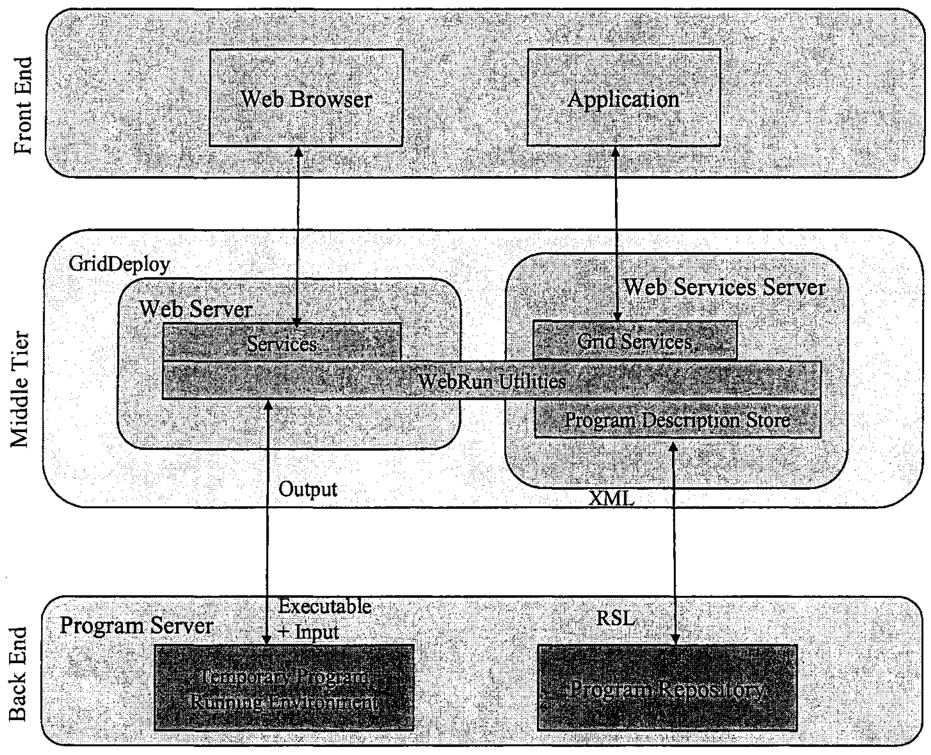


Figure 62. WebRun Architecture.

Based on the Grid service interfaces, dynamic web pages may be developed via servlets/JSP, which are in turn accessible by users via browsers. This approach facilitates remote job submission and monitoring for a chosen program. Since OGSA (Foster et al., 2002) was still under development during the implementation of this dissertation work, the implementation of WebRun creates Grid services using Java CoG-based (Laszewski et al., 2001) Grid functionalities instead of exploiting OGSA-based Grid services directly.

The 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 the web service and the usage of the Globus Toolkit 3.2 (GT3.2) (Foster & Kesselman, 1999), the domain experts are provided with a graphical modeling environment for automated generation of web services (Hernandez, 2004).

B.2 Program Description File

The Program Description File (PDF) uses XML to describe the various input arguments to the program. The various tags used are <optional>, <input>, <output> and <value>. The <optional> tag is used to specify that the argument is optional. Suppose one has a value that is associated with the argument, the <value> tag is used to mark the starting and ending limits for the value. The <input> tag is used to indicate that an input file is required for executing the program. The <output> tag indicates that an output file specified by the entry will be generated by the program. The PDF will be modified to support the Job Description Language (Forum, 2004), and other tags would be added in the future.

Two examples for PDF using XML are provided as the following.

Figure 63 shows possible XML tags with associated entries for ProgramA. ProgamA takes in a mandatory value as input and an output file that can be specified with the –l option, followed by any optional input file.

Format: `ProgramA <value> [-l] <outfile> [input]`
Program Description:
```
<value>value</value>
<optional>l</optional>
<output>outfile</output>
<optional><input></input></optional>
```

Figure 63. ProgramA - Program Description.

Figure 64 shows the XML tags associated with ProgramB. ProgramB takes in an optional input filename.

Format: `ProgramB [filename]`
Program Description:
```
<optional><value>filename</value></optional>
```

Figure 64. ProgramB - Program Description.

B.3 Wrapper Generation

The WebRun middleware provides users with programmatic access to programs distributed on heterogeneous hosting environments. To adopt programs in various environments, WebRun employs Grid services as a uniform access interface to the programs. Grid services hide the execution details of different programs; manage the execution of programs on the back end computing platforms, and transfer input and output of the program between the users and the computing resources. Whenever a user invokes a program through the WebRun middleware, the program is executed in a specific running environment created by the Grid service. All of the communication between the user and the running program is handled by the corresponding Grid services. Grid services act just like a wrapper that packs the program and its running environment into an independent tunnel between users and computing servers, without any interaction with the execution of other programs. Since different programs have different configuration for execution, various kinds of Grid services should be designed and implemented to facilitate users' invocation of programs. Two different deployment strategies are used to design Grid services according to different configuration of programs, as described below.

B.3.1 Application-Specific Grid service

Application-specific Grid services are designed for programs that have already been deployed on back end program servers. Those programs usually have complex configurations, so expertise is needed to install and tune them on the back end computing resources. These kinds of programs can be easily found in a broad applied area, from the genetic analyses over huge amount of DNA data, to the large-scale parallel simulation projects on compute farms.

The strategy of deploying this kind of application with WebRun middleware is depicted in Figure 65. In this architecture, the front-end is the user's application that invokes the program via Grid services. The back end is a group of servers that provide computing power for the execution of programs. Each program server has a program repository working as a container of programs and their running environments.

WebRun middleware is composed of a Program Description Store, a WebRun factory and some WebRun Utilities. Before being deployed, each program on the program servers needs to be registered with the Program Description Store. This registration process generates a record in PDF format for the registered program and stores the record into the Program Description Store.

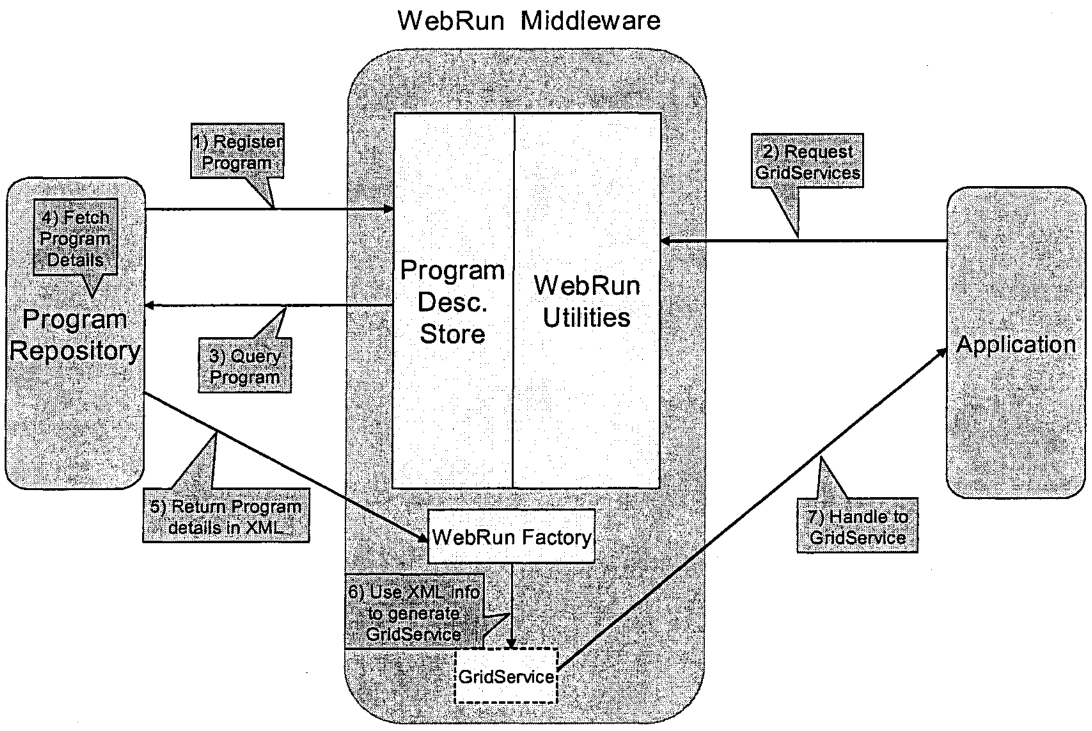


Figure 65. WebRun: Application Specific Deployment.

The deployment process is described as follows: When an application requests the Grid services for a program execution, the WebRun Utilities accept the request and query the Program Description Store. Once a match between the request and the records are found, WebRun Utilities determine the name and address of the corresponding program repository. By querying the program repository, WebRun Utilities fetch the PDF of the target program. Two aspects of information about the program are obtained by parsing the PDF. One is a complete list of all possible program arguments, including argument name and argument property (mandatory/optional). The other aspect of information is about the execution environment, such as the name and IP address of the hosting server, and the physical path of the program in the file system. All these information are transferred to the WebRun Factory, which builds up a specific Grid service for the target program, and redirects the user application's request to this Grid service. After the execution of the program, the specific Grid service is cleaned up automatically by the WebRun Utilities.

Several Grid services are developed to facilitate the deployment of application-specific Grid services. They are as follows:

1. ProgramMatchPort, which matches user application request to the records in the Program Description Store;
2. ProgramListPort, which queries the program repository on the computing server and returns the PDF of the target program;
3. PDFParsing, which parses the PDF and generates the corresponding descriptions of the program; and
4. WebRunFactory, which generates the Grid services and redirect users' application.

To explain the operation of WebRun, consider an executable HelloWorld that takes in an optional addressee. The format for executing the program on command line and the corresponding XML Program Description are given in Figure 66. Application-specific Grid services can be applied to deploy the HelloWorld program as a Grid service. Prior to deployment, the HelloWorld program should first choose a hosting back end server and register itself with the server's program repository. A PDF is generated during the registration and saved in the repository. Whenever a user application tries to invoke the HelloWorld program, the PDF file will be transferred to WebRun Utilities and parsed to enable the WebRunFactory Grid service to generate the specific Grid service. This specific Grid service for the HelloWorld program is called the HelloWorldService, as shown in Figure 67.

Format: `HelloWorld [-to addressee]`
Program Description:
<optional>to<value>addressee</value></optional>

Figure 66. HelloWorld - Program Description.

```
public interface HelloWorldService {
     java.lang.String hello();
     java.lang.String hello(java.lang.String message);
}
```

Figure 67. Grid Service for HelloWorld.

The user application which invokes this Grid service may appear as in Figure 68. For initial demonstration purposes, WASP (Systinet) is used as the web services server to serve as the corresponding registry service.

B.3.2 General Grid Service for Application Invocation

A general grid service for application invocation is designed for programs that are only temporarily executed on back-end program servers. Those programs are usually provided directly by the end-users, executed several times by the same users, and cleaned up by the servers immediately after their execution. To build up application-specific Grid services for each of these programs may be an inefficient approach. For example, a user may only want to execute his/her own program in a Grid environment once, then the overhead of creating a Grid service to deploy the program may be too costly compared to program execution itself. Moreover, the license and security issues of software may also prevent users to establish the Grid services and publish them permanently. Users expect a unified platform that can let them stage their executables, invoke the programs, and transfer the input/output between back end servers and front-end applications. Thus, an agile, general-purpose strategy for invocation of programs in a Grid environment is developed to accommodate users' requirements.

```
import org.systinet.wasp.webservice.Registry;

public class HelloWorldClient {
    public static void main(String[] args) throws Excep-
tion {
        String serverURL = args[0];

        // lookup of HelloService
        HelloWorldService helloWorldService =
(HelloWorldSer-
vice)Registry.lookup(serverURL,HelloWorldService.class);

        // call HelloWorldService and print out a re-
sponse message
        System.out.println(helloWorldService.hello());
        System.out.println(helloWorldService.hello("-to
World"));
    }
}
```

Figure 68. User Application Invoking HelloWorldService.

The architecture of general Grid services for application invocation is shown in Figure 69. The basic idea of this strategy is to execute the user-provided program in a temporary running environment, aided by a set of Grid services. When a user submits a request for program execution, WebRun Utilities first accepts the request and tries to find an appropriate computing resource for the execution. Once a program server is chosen, it is asked to construct an independent, temporary execution environment to accommodate user's program. With such an environment established, executable and input files will be staged to the program server by WebRun utilities using GridFTP. With the help of WebRun utilities, users can invoke, monitor, and manage their programs via various Grid services. After obtaining the output of their programs, users call Grid services to destroy the temporary execution environment and end the communication.

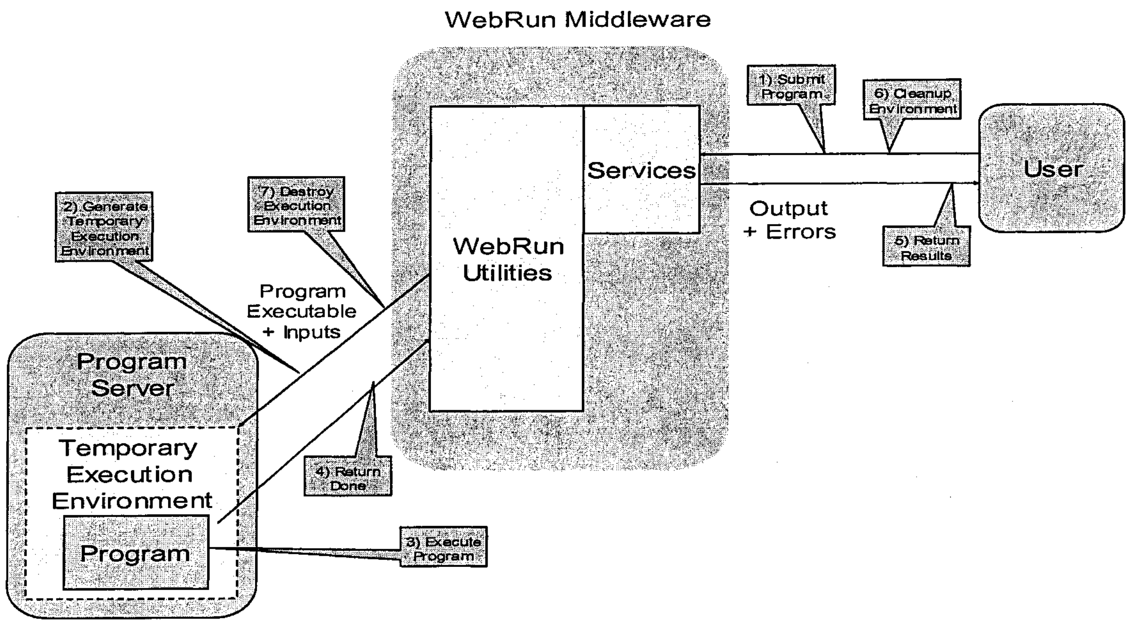


Figure 69. WebRun: General Application Invocation.

To enable the application invocation of general programs, the following Grid services are designed and implemented.

1. SetCredential and DestroyCredential, which setups and terminates users' certification.
2. SubmitJob, which invokes the programs on the back end server side.
3. GetJobStatus, which checks the status of the program execution.
4. WaitUntilDone, which executes null operations until programs finish.
5. SetFTP and StopFTP, which sets up and terminates GridFTP transformation.
6. UploadFile, which transfer files from user side to server side.
7. DownloadFile, which transfers files from server side back to user side.

With the help of these Grid services, the WebRun is able to provide the user a demonstration program, by which users can learn how to use the Grid services. This demo program, using Java Cog infrastructure (Laszewski et al., 2001), indicates what Grid services should be used, and in which order.

GRADUATE SCHOOL
UNIVERSITY OF ALABAMA AT BIRMINGHAM
DISSERTATION APPROVAL FORM
DOCTOR OF PHILOSOPHY

Name of Candidate	Zhijie Guan
Graduate Program	Computer Information Systems
Title of Dissertation	Grid Flow: A Grid-Enabled Scientific Workflow System With a Petri Net-Based Interface

I certify that I have read this document and examined the student regarding its content. In my opinion, this dissertation conforms to acceptable standards of scholarly presentation and is adequate in scope and quality, and the attainments of this student are such that he may be recommended for the degree of Doctor of Philosophy.

Dissertation Committee:

Name		**Signature**
Anthony Skjellum	, **Chair**	[illegible]
Purushotham V. Bangalore		[illegible]
Jeffrey G. Gray		[illegible]
Tracy P. Hamilton		Tracy Hamilton
Elliot J. Lefkowitz		[illegible]

Director of Graduate Program [illegible]

Dean, UAB Graduate School [illegible]

Date JUN 16 2006