

Workflow Engine with Multi-Level Parallelism Supports

Qifeng Huang, Yan Huang

School of Computer Science, Cardiff University
{q.huang, yan.huang}@cs.cf.ac.uk

Abstract

This paper presents the SWFL workflow engine, a general workflow framework that meets the needs of business processes as well as scientific computing processes with fine multi-level parallelism supports. The workflow description language, SWFL, follows a graph-oriented model to specify workflow processes composed of services. The workflow engine provides an efficient enactment environment for SWFL flow model composed of services. It provides multi-level parallelism supports: a server-level parallelism support using flexible server-level schedule algorithms, a flow-level parallelism support by partitioning a workflow into sub-flows and running the sub-flows in multiple job processing servers in parallel, and a message-passing parallelism support by using the MPFL, an extension language to SWFL and its associated tools. The architecture of the workflow engine and some other related implementation issues are also presented in the paper.

1. Introduction

Web services and grid services, are emerging as important paradigms for distributed computing. A service usually encapsulates information, software or other resources, and makes it available over the network via some standard interfaces and protocol. Complex services or specific applications may be created by aggregating the functionality provided by simpler ones instead of writing new monolithic code. This is referred as service composition or service workflow. The current growing interest in service workflow languages and corresponding enactment environments has led to the appearance of many languages for specifying the composition of services to form complex applications, such as WSFL, BPEL4WS, WSCI, BPML and GSFL [1-5]. However, they are mostly limited to business processes and simple scientific processes. They usually just provide limited support for parallel processing, and currently do not go further to meet the requirements for specifying more sophisticated scientific workflow applications that may require support for fine granular parallelism.

The GSiB (Grid-Service-in-a-Box) project [6], funded by EPSRC, presents a novel general service workflow framework, which meets the needs of business process as well as scientific processes. It provides easy-to-use visual tools for building web service composite applications and executing them in a workflow engine based Grid system with flexible multi-level fine granular parallelism supports.

As parallel computing especially grid computing application expands, scientific workflow represents the logical culmination of the trend that constructs complex distributed composite solutions of major initiatives in high performance computing such as climate modeling and high-energy physics, to enable more cost-effective usage and efficient collaboration of distributed computational resource and existent efforts. The convergence trend of grid services and web services makes such a general workflow feasible and necessary. Some new features presented in the WS-Resource Framework [7], such as replacing the object-oriented GSH (Grid Service Handle) with standard WS-Addressing Endpoint References, are the most obvious proofs. This trend also makes clear that such a general workflow falls into the same research areas as web service composition and orchestration, where abundant techniques and results can be leveraged.

Service workflow framework in GSiB is built with another two important factors in mind: the need to exploit maximal parallelism supports for sophisticated scientific processes; follows a graph-oriented approach to make it easy for the end users to create and execute a workflow even though they have little knowledge of related techniques.

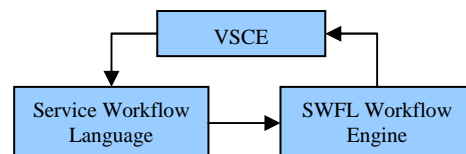


Fig. 1: Service Workflow Architecture in GSiB

Figure 1 represents the service workflow architecture in GSiB. It includes three basic components. SWFL (Service Workflow Language) is an XML-based description language that is used to specify a workflow application formed from independent services by describing how they are composed and their message exchanges, i.e. control and data dependencies among them. The SWFL Workflow Engine provides a distributed enactment environment for the automated execution of processes described in SWFL. And VSCE (Visual Service Composition Environment) provides a visual tool for a user to build his workflow model, initiate the workflow execution, retrieve the execution results from the engine, and track runtime status of executing workflow instance.

The remainder of this paper is organized as follows. Section 2 outlines the syntax and semantics of SWFL workflow description language, presents its graph-oriented features, and gives simple samples to specify SWFL processes in both XML documentation as well as directed graph. Section 3 discusses the processing of a SWFL workflow and presents the architecture of SWFL workflow engine. Multi-level parallelism supports in SWFL engine is discussed separately in Section 4. Section 5 outlines the conclusions and planned future work. However, details about VSCE and its relationship with the other two components can be found in [8].

2. SWFL Workflow Description Language

2.1 Syntax and Semantics: Basic Elements

SWFL is the main workflow language supported in GSiB[6]. It is an XML-based description language and is defined using XML schemas. By using SWFL, a normal web service composite application can be specified in an XML documentation. The basic elements defined in SWFL are listed as following:

- **Types:** A container for abstract type definitions which may be used in the message and variable definition.
- **Message:** Defines abstract messages that may be used in the workflow model.
- **Variable:** Defines global variables, which can be shared by all included activities and sub-flow throughout a workflow.
- **Activity:** Could be an atom activity or a structured activity. An atom activity is usually involving one of the operations of

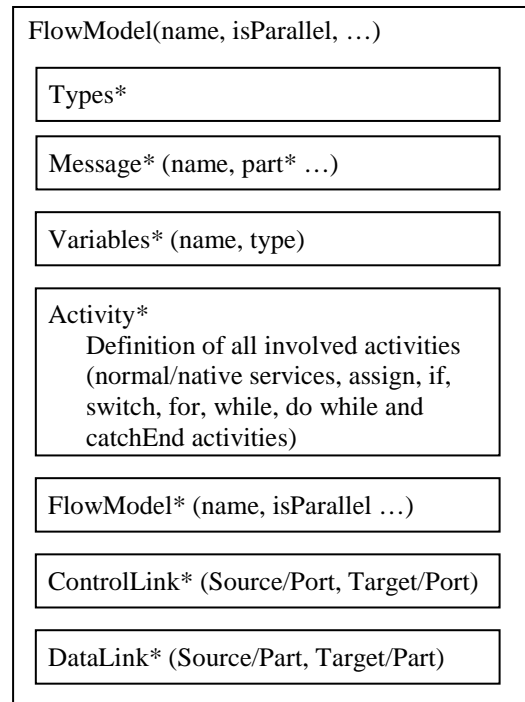


Fig. 2 SWFL Elements

an existing deployed web service, or an assignment activity, which assigns the output of an activity to a variable. A structured activity defines the execution entry point of a conditional or loop control construct, which could be *if-then-else*, *switch*, *for*, *while* or *do-while*. Another special activity defined in SWFL is *catchEnd*, which define the actions for a specific fault. This function is much required in business processes. However, as a special application of *catchEnd* activity in scientific workflows, we can use it to support checkpoints to improve the availability of distributed applications, as it may take days, or even weeks to complete such a job.

- **FlowModel:** An XML documentation representing a SWFL workflow. It could be a high-level complex flowModel which involves many other flows, in which case, its XML document could include as many individual workflow specifications as needed. Modularization is widely adopted in SWFL to simplify the definition of a SWFL flowModel and encourage the reusability the modular components used for building the flowModel.
- **ControlLink:** Defines the control dependencies among control ports of involved activities and flowModels.

- **DataLink:** Defines data dependencies in the flow model definition. It describes the data flow between involved activities and flowModels.

Figure 2 lists main elements of SWFL.

2.2 Graph-Oriented Features

As originally an extension to WSFL [1], SWFL inherits WSFL's graph-oriented approach to build a workflow model for applications composed of web services. In SWFL, a typical workflow model is described as a directed graph. Each node in the graph is an activity task node that normally involves one web service and the edges of the graph are either control or data links that represent the control or data flows between the task nodes [9].

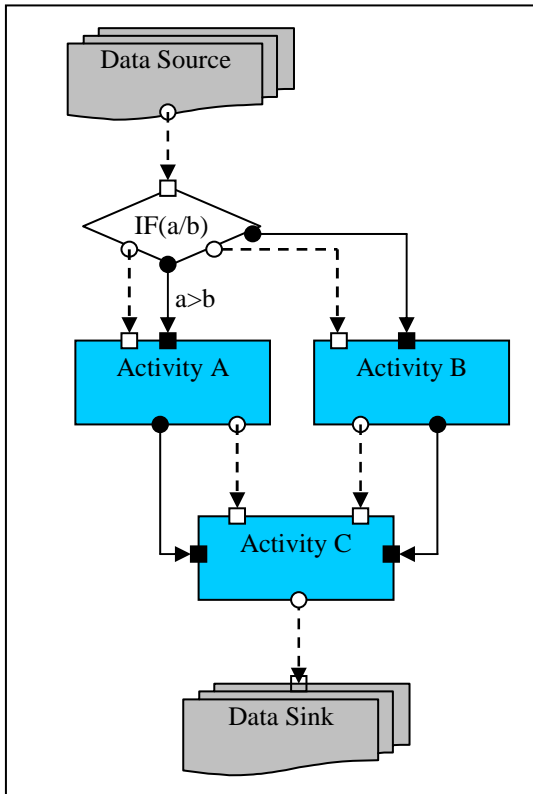


Figure 3. A SWFL workflow example represented in a directed graph

Figure 3 gives a SWFL workflow example presented in a directed graph, and Figure 4 gives a SWFL workflow example represented in an XML document. In SWFL, a typical activity node including a workflow activity node contains data input/output ports for specifying data links and control ports for specifying control links. A data link, represented in a dashed line in figure 3, represents a data flow from a source activity node to a target activity

```

.....
<flow name="sample" requireParallel="false">
  <wsdl:input message="flowInput"/>
  <wsdl:output message="flowOutput"/>
  <activity>
    <if name="ifControl">
      <input name="hehe" message="msg1"/>
      <condition id="IF" expr="msg1.a &gt;
        msg1.b"/>
      <else id="ELSE"/>
    </if>
  </activity>
  <activity>
    <normal name="ActivityA">
      <performedBy namespace="srvWSDL">
        <portType>math</portType>
        <operation>abs</operation>
      </performedBy>
    </normal>
  </activity>
  .....
  <controlLink>
    <source name="ifControl" port="IF"/>
    <target name="task2"/>
  </controlLink>
  .....
  <dataLink target="ifControl">
    <source name="ActivityA">
      <map>
        <part>
          <source part="a"/>
          <target part="f1"/>
        </part>
      </map>
    </source>
  </dataLink>
  .....
</flow>
.....

```

Figure 4. A SWFL Workflow example

node, and a control link, represented in a solid line in figure 3, represents a control flow from a source activity node to a target activity node. As an example, in Figure 4, part of the XML document of the workflow example displayed in Figure 3 is listed.

Similar to BPEL4WS, one of the workflow languages which have been widely accepted, SWFL is used to build workflows which involve peer-to-peer interactions between web services. However, SWFL adopts a graph-oriented approach to represent a workflow rather than a script-oriented approach which is used in BPEL4WS. The advantages by using a graph-oriented method rather than a script-oriented method lie in two aspects:

Firstly, this method is referred to the successes of flow chart in programming and UML modelling tools in systems design. It is easy to understand and use, and helps end users to composite their own workflow applications, especially using friendly VSCE tools, i.e. creating an application by drawing a corresponding graph using the drag-and-drop visual tools, which don't require end users to do any comprehensive low-level programming. In SWFL, activities are wired together through data and control links, and this is a flow-chart-like expression of a workflow application.

Secondly and which is more important, graph-oriented SWFL provides more scope for job partitioning and scheduling. In SWFL, there is no explicit execution order presented in a flow model graph except explicit data/control dependencies, so the execution order of the task nodes can be decided dynamically at runtime based on the available resources to achieve better performance. In addition, a workflow can be easily partitioned into multiple sub-flows according to its graphical structure to be run in parallel. In BPEL4WS, all parallelism should be explicitly expressed in the script in BPEL4WS. Although there are also efforts made to decentralize the execution of a BPEL process to allow parallel execution of a BPEL process, it has to abstract a graphical structure from its script-like workflow documentation before partitioning it to achieve efficient parallelism and performance, while abstracting a graphical workflow structure from a script-like document is rather challenging [10-12]. From this point of view, SWFL has the advantage of presenting a workflow directly using its graphical structure.

3. SWFL Workflow Engine

The SWFL Workflow Engine provides an autonomic and self-organizing enactment environment for the processing of SWFL workflow applications, either described in XML documentations or directed graphs.

3.1 XML2Graph, Graph2Java and Graph2XML

To take advantages of the graph-oriented features of SWFL, an XML-described flow model isn't directly transformed into and stored/deployed as executable programmes, but is usually stored in an intermediate form of graph objects. Two kinds of graph objects, object DataGraph and object ControlGraph, are abstracted from the workflow specification. They represent the data and control flow

structures of involved services in a workflow application respectively.

There are two main reasons for using this intermediate form of graph objects. Firstly, this is a straightforward format to be used by the VSCE tools to edit and store flow models. Secondly, according to the graphical structure stored in a graph object, the workflow engine can easily generate its executable program and schedule and execute it dynamically during runtime to get better performance. Graph objects are a much more steady form for a deployed workflow application, as different executable programs can be generated to adapt to dynamic runtime environment.

Three tools are provided by SWFL engine to accomplish the conversions between a SWFL workflow documentation, graph objects and executable program. They are XML2Graph, Graph2Java and Graph2XML. XML2Graph is used to convert a SWFL document to graph objects; Graph2Java is used to convert graph objects to an executable Java program, and Graph2XML is used to convert graph objects to a SWFL document.

Typically the processing of a SWFL flow model can be described as three steps which illustrated in Figure 5. Firstly, the XML documentation is transformed into one or more graph objects which represent the graphical workflow structure of the application. Secondly, an Java executable program is created according to the graphical workflow structure stored in the graph objects. Finally, the executable program is scheduled and executed in the enactment environment, and the final result is returned.

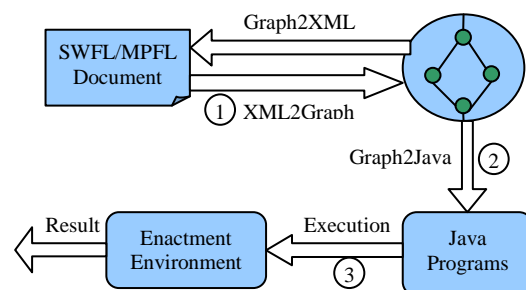


Figure 5. Processing a SWFL application

3.2 Engine Architecture

The workflow engine is an enactment environment for the automation of distributed processing of a SWFL flow model. Its basic architecture and interactions are depicted in Figure 6.

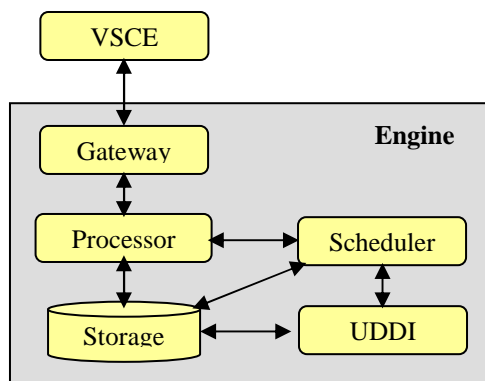


Fig. 6 SWFL Engine Architecture

The Gateway itself is a web service. It provides a standard entry point as well as API for VSCE and end users to submit a job and to retrieval the final results and runtime status of a submitted job. When a job is submitted as an XML documentation, XML2Graph tool is executed first to build the corresponding graph objects.

Job Processors are the coordinated computing resources of the workflow engine with multithread support. Each processor has a pool of worker threads, each of which is used for a specific application. A SWFL job can be partitioned and dispatched to be executed on multiple distributed processors.

The Scheduler takes fully advantage of the flexibility provided by graphical approach used in SWFL. The approach separates a workflow structure from its implementation at runtime. A SWFL workflow specification specifies the activities and the control and data dependencies among activities, but not a pre-defined execution order. Therefore, the execution order of the activities will be decided by the Scheduler at runtime according to availability of the resources and other information such as the workload of the servers.

There is no particular discovery service in the workflow engine. Since most of the components used in the engine themselves are represented as web services, such as Gateway services and job processor services, UDDI is used as the main mechanism for service registry and discovery. In addition to UDDI, a general-purpose registry server called Identical Service Registry is built on top of UDDI to provide services for registering and querying identical services. Identical services are semantically equivalent services located at different URL endpoints, such as different job processor services. Although identical services provide the same services, the services they provide may be of different quality and have various

limitations. For an example, a job processor service hosted in a fast machine provides a faster service while a job processor service hosted in a slow machine provides a slow service. In addition to the WSDL document that defines the interface and binding information for a web service, additional metadata may be needed to distinguish identical services. ISR introduces such a mechanism for associating additional metadata with services in a registry.

Finally, the Storage service provides a space for storing engine-related objects as well as API for accessing the space.

4 Multi-level Parallelism Supports in SWFL Engine

Multi-level parallelisms support is one of the main characteristics that distinguish the SWFL workflow engine from other engines. In order to support more sophisticated scientific parallel processes, the SWFL workflow engine supports server-level, flow-level and message-passing parallelisms.

4.1 Server-Level Parallelism

Service-level parallelism involves only one particular job processor and the parallelism is mainly achieved by multithreading.

In a typical SWFL workflow, each activity node represents a task involving a particular web service operation. We say the task is ready to run only when all its input data are ready and all tasks which it has control link comes from are complete. A task can be dispatched at anytime after it is ready. So using one thread for each ready task, multiple ready tasks can be run in parallel to achieve server-level parallelism.

When to start the execution of a ready task is a problem faced by the server-level scheduling algorithm. To start a task as soon as it becomes ready or to run it when it is required? To start a task as soon as it becomes ready may cause unnecessary waste of computing resource and storage especially the memory. For example, a task may be executed much earlier than is required, so its output data, which may be large, has to be buffered for a long time before it is consumed. To start a task whenever it is needed seems more efficient in terms of space-saving because the data produced by the task will be consumed immediately so no buffering is needed. However, because the execution time is unknown, there is difficulty to predict the exact time the task should start.

At present, SWFL uses a greedy algorithm which allows an activity task node to run

whenever it is ready. A top limit is set to avoid the job processor server to become overloaded with a large number of running activity tasks. The limit is set by the administrator and varies according to the performance of servers.

Since the approach which starts a task whenever it is required promises a more efficient scheduling scheme. A mechanism is under investigating for predicting the execution time of an activity task and the size of its output data from its history information, so that a task, instead of being executed as soon as it becomes ready, is executed whenever it is the best time for it to start to minimise the waiting time as well as data buffering.

4.2 Flow-Level Parallelism

The SWFL workflow engine also supports flow-level parallelism. This involves partitioning a workflow into individual sub-flows and running these sub-flows on different job processors in parallel.

Some workflow graph may not be connected graph, in which case, the workflow can be easily grouped into sub-flows and each sub-flow corresponds to an individual connected sub-graph. Each sub-flow is then submitted to different job processors to be run in parallel. However, in many cases, a workflow is a connected graph, and when parallelism is required, a partition algorithm is needed to partition the connected workflow graph into sub-flows.

The partitioning on connected workflow graph is made according to the available job processors, the control links and data dependences. The number of available job processors decides the maximum number of the sub-flows. A number of rules are followed by the partitioning algorithm with different privileges: (1) There should be minimal communication among sub-flows; (2) The execution time for each sub-flow should be minimal; (3) The total execution time for each sub-flow should be minimal; (4) The workload of each sub-flow should be roughly the same. These rules may not be followed easily because some factors may be unknown until run time. For example, the exact amount of communication between two nodes may not be known until the communication happens, and the unknown workload a web service currently has makes it difficult to predict the execution time of a activity node which involves that web service.

Normally, the owner of the job can specify whether the job should be processed in parallel and how many sub-flows it may need. But

sometimes they can leave these decisions to the workflow engine to make. The workflow engine then determines the number of sub-flows based on the total number of task nodes in the workflow graph and the number of available job processors.

Flow-level parallelism may exclude potential bottlenecks which may occur because of the limited memory space or limited CPU time of a single job processor. By coordinating execution of task nodes, data and control flows in a workflow among distributed job processors, parallelism can be achieved as well as efficiency, high throughput, and scalability.

4.3 Message-Passing Parallelism

MPI standard defines the user interface and functionality for a wide range of message-passing operations [13]. Because the similarity between a MPI program and a normal workflow application, it is possible to represent a MPI program in a workflow document in a XML language such as MPFL. A sub-routine in a MPI program can be deployed as a web service, and an invocation to the sub-routine can be taken as an activity node in a workflow. Since message passing has been widely used in high performance scientific computing, and it is important to support the function of message passing in the workflow languages for the scientific computing.

MPFL (Message Passing Flow Language) is an extension to SWFL, which provides an XML description language for describing composite scientific flow with massive message-passing [14]. MPFL not only defines MPI in XML to web service composite scientific applications that involve more sophisticated MPI-like parallel processing, but also builds a web service-based communication model that achieves most of the functionalities MPI standard provides. MPFL uses the same graphical approach as in SWFL. Most of the basic components of SWFL are kept unchanged to be consistent with SWFL. However, the types of activities are extended to support a large set of communications such as send, receive, broadcast, and etc. Data links in MPFL are extended to allow data flow between MPFL flow models to support point-to-point as well as collective communication between two workflow processes. In a typical MPFL flow model, multiple processes are involved. However, the processes are defined by using templates and communication pattern to simplify the definition which is similar to MPI programming. Moreover, the MPFL workflow

engine is just an accumulative extension of existing SWFL engine.

Message-passing parallelism is specific to MPI-like scientific processes. An MPFL job is processed by a special MPFL job processor which usually works in conjunction with a cluster of other servers. Several language converters such as MPFL2C++, MPFL2Fortran and MPFL2Java are being developed to generate machine-specific executable instances, which may include web service invocations, from a MPFL job. The MPFL job processor uses the MPFL language converters to execute an MPFL job in its local cluster environment. It is the responsibility of the engine to deal with complex communication models and synchronization between job processor. The development of MPFL job processor is still at an early stage. Further details are expected to be presented in a future paper.

5 Conclusions and Future Work

In this paper, we have outlined a general service workflow engine that can be used for business process as well as scientific workflow. The workflow engine is based on SWFL, an XML-based workflow description language, which represents service interactions with their data and control dependencies using a graphical approach. Such a graphical approach makes it easy for the user to construct sophisticated application.

SWFL workflow engine provides multi-level parallelism supports, in order to composite scientific applications with more sophisticated flow structures and advanced parallel processing requirements. It provides server-level parallelism by using flexible server-level scheduling algorithms, flow-level parallelism by partitioning a workflow into sub-flows and then having the sub-flows run in multiple job processors, and message-passing parallelism by using the MPFL extension to SWFL and associated tools.

Future work will include implementation of the MPFL workflow engine and supporting tools, evaluation the performance of each level of the parallelism supported in the SWFL engine and more study on the partition algorithms and scheduling algorithms to achieve a better performance.

References:

[1] F. Leymann, Web Services Flow Language (WSFL 1.0), [http://www-4.ibm.com/](http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf)

- [software/solutions/webservices/pdf/WSFL.pdf](http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf), May 2001.
- [2] T. Andrews, F. Curbera, H. Dholakia, Y. Golan, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic and S. Weerawarana. Specification: Business Process execution Language for web Services Version 1.1, <http://www-128.ibm.com/developerworks/library/specification/ws-bpel/>, May 2003.
- [3] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takaci-Nagy, I. Trickovic, and S. Zimek. Web Service Choreography Interface (WSCI) 1.0. <http://www.w3.org/TR/wsci/>, August 2002.
- [4] A. Arkin., Business Process Modeling Language, <http://xml.coverpages.org/BPML-2002.pdf>, November 2002.
- [5] S. Krishnan, P. Wagstrom, G. Laszewski, GSFL : A Workflow Framework for Grid Services, <http://www-unix.globus.org/cog/papers/gsfll-paper.pdf>, July 2002
- [6] Y. Huang. GSIB: PSE Infrastructure for Dynamic Service-Oriented Grid Applications. In Proceedings of 2003 International Conference on Computational Science (Part 4), June 2 - 4, 2003, Melbourne, Australia.
- [7] K. Czajkowski, D. F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe, The WS-Resource Framework, <http://www.globus.org/wsrfl/specs/ws-wsrfl.pdf>, March 2004.
- [8] Y. Huang, Q. Huang. GSIB Visual Environment for Web Service Composition and Enactment. 4th UK e-Science Programme All Hands Meeting, September 20-22, 2005, Nottingham, UK.
- [9] Y. Huang and D. W. Walker. Extensions to Web Service Techniques for Integrating Jini into a Service-Oriented Architecture for the Grid. In Proceedings of 2003 International Conference on Computational Science (Part 3), June 2-4, 2003, Melbourne, Australia.
- [10] G. Chafle, S. Chandra, V. Mann and M. G. Nanda. Decentralized Orchestration of Composite Web Services. In Proceedings of 13th International World Wide Web Conference, May 17-22 2004, New York, USA.
- [11] M. G. Nanda, S. Chandra, and V. Sarkar. Decentralizing Composite Web Services. In Proceedings of Workshop on Compilers for Parallel Computing, January 2003.

- [12]W. M. van der Aalst. Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques. In Proceedings of Business Process Management, 2000.
- [13]M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra, MPI— the Complete Reference: Volume 1, the MPI Core, 2nd ed. The MIT Press, 1998
- [14]Y. Huang, Q. Huang. WS-Based Workflow Description Language for Message Passing Interface. In Proceedings of 5th International Symposium on Cluster Computing and the Grid, May 9-12, 2005, Cardiff, UK.
- [15]H. Truong, T. Fahringer, Online Performance Monitoring and Analysis of Grid Scientific Workflows, European Grid Conference 2005, February 14 -16, 2005, Amsterdam, Netherlands.