ABSTRACT

MOUALLEM, PIERRE. Fault Tolerance and Reliability in Scientific Workflows (Under the direction of Dr. Mladen Vouk).

The emerging technologies of web services, agents and service-oriented workflows will enable scientific projects and experiments to be conducted on a larger scale than ever before. Data used and produced in such projects and experiments become increasingly complex and heterogeneous. Thus the need for a tool (or a set of tools) to efficiently design, manage and maintain problem solving flows (scientific workflows) using various components. The DOE Scientific Data Management (SDM) initiative aims to develop a framework that helps scientists to manage data in distributed and collaborative environments. It also provides tools that help them create and manage scientific workflows that use network-based (web) services, agent technologies and semantic mediation techniques. The current SDM's framework is known as SPA/Kepler and is Ptolemy II based.

One of the vulnerabilities of service dependent workflows is that they require that the web services they use to be available whenever the workflow is run. If key web services are not available, the workflow cannot finish successfully. At that point a scientist using such as service would have to wait for it to be restored, This, of course, impacts workflows reliability and availability, and may be sufficient for an end-user to stop using workflows that use those services.. The work reported here uses the SPA/Kepler framework to explore the issue of reliability of service-based scientific workflows.

For example, a workflow that invokes 3 services in a series may have .an acceptably high overall failure probability. This thesis explores the issues related to improvement of the overall workflow reliability using fault tolerance. Specifically, the work focuses on failure-masking and fail-over through redundancy, and in the context of individual services, rather than on provision of checkpointing and recovery.

Analyses show that even a relatively simple redundancy based fault-tolerance approach, such as duplication of key services, can provide an order of magnitude or better reliability. In the context of an actual implementation, one option is to find locations of alternative (functionally equivalent) services during workflow design, and then use that information at run-time if the primary service fails. A more practical method is to publish the list of services used by the workflow to a UDDI type service and have a way of dynamically matching needed services with functionally equivalent ones if a fail-over is required. A prototype solution of the latter, based on a commercially available brokering service, was developed for one of the SDM pilot workflows to show its viability. It is discussed in detail.

FAULT TOLERANCE AND RELIABILITY IN SCIENTIFIC WORKFLOWS

By

PIERRE MOUALLEM

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

COMPUTER SCIENCE

Raleigh

2005

APPROVED BY:

Dr. Munindar Singh	**Dr. Peter Wurman**

Dr. Mladen A. Vouk
(Chair of Advisory Committee)

1.1.1.1 DEDICATION

I dedicate this thesis to my parents and my brother. I could never have done it without their constant support, encouragement and faith in me, especially when I decided to switch from Architecture to Computer Science. Thank you for everything you have done for me, I will never forget it.

1.1.1.2 BIOGRAPHY

Pierre Mouallem was born in Beirut, Lebanon. He completed his primary education in Lebanon. He received his Bachelor's degree in Computer Science from Notre Dame University, Lebanon. He joined the Master's program in Computer Science at NC State University in Spring 2003, and joined the SDM group in the Fall 2003.

1.1.1.3 ACKNOWLEDGEMENTS

I express my sincere gratitude towards the Chair of the Advisory Committee of my Thesis, Dr. Mladen Vouk, for his guidance, support and encouragement. I also would like to thank Dr. Munindar Singh and Dr. Peter Wurman for serving on my thesis committee and guiding me through various parts of it.

Many thanks to my colleagues at the SDM group at NC State for their support and help throughout the research and writing of my thesis. Special thanks to Zhengang Cheng who has been an excellent person to work with and has helped me with various aspects of this project.

Last but not least I would like to thank my friends for their help, support and faith in me.

This work has been supported in part by the DOE SciDAC grant DE-FC02-01ER25484, IBM Shared University Program, and StrikeIron.

1.1.1.4 TABLE OF CONTENTS

1.1.1.5 LIST OF FIGURES

1.1.1.6 LIST OF TABLES

1. Introduction

"Scientific research is exploratory in nature. Scientists carry out experiments, often in a trial and error manner, wherein they modify the steps of the task to be performed as the experiment proceeds. As technology advances, more and more scientists are relying on computing systems to aide them in such explorations and problem solving. The scientist typically divides up the overall task into smaller sub-tasks, each of which can be considered to be an individual step in an experiment. The results obtained from each such step are either analyzed and/or stored for dissemination to other sites or individuals, or are used as an input to the next step in an experiment or exploration, or both. Such a reuse of data can be done repeatedly until the overall task is completed to scientist's satisfaction. We call such a chaining of smaller tasks together to achieve desired results from the experiment or explorations, using various data and analysis services, a workflow". [Singh96, Allen01]

Workflows can be naturally mapped onto graph representations. A graph could be translated then into source code and compiled, However, it is beneficial to keep high level graph representation of workflow so non-programmers can modify application logic without the need to become programmers or script writers. Workflows are well established technology in business domain, and recently they started gaining in popularity in the scientific domain.

Examples of workflows can be found in almost any domain of scientific discovery [DOE04]. There are many commonly occurring scenarios in all scientific domains that can be automated using a sufficiently robust and flexible workflow infrastructure.

Workflows have many synergies with web services. In fact, web service based workflows are quickly becoming a requirement of a wide range of new service-oriented applications. This is one of the reasons why the Software Process Automation project [SPA05] has been started. Many domain experts, particularly in the life sciences, do not wish to construct workflow by coding them at the level of individual platforms – beyond what is necessary to do research in their domain, e,g., to develop appropriate algorithms. They would like to considerably reduce the overhead currently required by some information technology solutions (sometimes as much a 50% of the activity). Therefore, workflow automation and higher-level specification fits naturally into the trends towards increased domain specialization as application developers move to become web service providers, and computer scientists seek to provide reusable, general-purpose, scientific tool, rather than custom made scientific application. Service orchestration may also play an important role fulfilling the promises of on-demand, service-oriented computing.

This thesis discusses a possible enhancement to the services aspect of the modern scientific workflows. A problem occurs when a workflow component tries to invoke a web service and fails. Currently, more often than not if a run-time service failure occurs, the workflow would simply terminate. Today, there is a sufficient proliferation of diverse but functionally equivalent services that introduction of a redundancy-based per service recover may be economical This is the solution that is discussed in this thesis..

The thesis is organized as follows. Chapter 2 gives an overview of web services (WS), their history, motivations, architecture, and components. It also discusses service oriented architecture, WS-Policy, and implementation of web services. Chapter 3 introduces Fault Tolerance and reliability concepts. It overviews different failure and fault models that may apply, it discusses fault tolerance models how they can be applied in the case of web services. Chapter 4 gives an overview of scientific workflows, then it discusses the Software Process Automation (SPA) project. Chapter 5 proposes a model that addresses the reliability and fault tolerance aspect of the workflows. It also proposes several methods for implementing that model, and discusses their implementation, advantages and limitations. Finally chapter 6 concludes and discusses possible future work.

2. Background: Web Services

2.1 History and Motivation

Web services are services offered via the Web. In a typical Web services scenario, a business application sends a request to a service at a given URL using the SOAP protocol over HTTP. The service receives the request, processes it, and returns a response. An often-cited example of a Web service is that of a stock quote service, in which the request asks for the current price of a specified stock, and the response gives the stock price. This is one of the simplest forms of a Web service in that the request is filled almost immediately, with the request and response being parts of the same method call.

In a more technical description, IBM describes web services as: “Web services are a new breed of Web application. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes” [IBM01]. In other words, web services are interoperable building blocks for constructing applications. As an example, we can imagine a distributed digital library infrastructure built on web services providing functionality such as distributed search, authentication, inter-library loan requests, document translation and payment. These web services would be combined by a particular digital library application to offer an environment for reaching information resources that is tailored to its particular user community.

Another advantage of web services is that they make no assumptions about the platform or implementation language. Java can communicate with C# or legacy systems. IBM and

Microsoft have both been instrumental in the definition of web service standard, making it possible to communicate between J2EE and .NET applications.

2.2 Web Services Architecture and Components

The Web Services architecture is the logical evolution of object-oriented analysis and design, and the logical evolution of components geared towards the architecture, design, implementation, and deployment of e-business solutions. Both approaches have been proven in dealing with the complexity of large systems. As in object-oriented systems, some of the fundamental concepts in Web Services are encapsulation, message passing, dynamic binding, and service description and querying. Fundamental to Web Services, then, is the notion that everything is a service, publishing an API for use by other services on the network and encapsulating implementation details. [W3C04]

Several essential activities need to happen in any service-oriented environment:

1. A Web service needs to be created, and its interfaces and invocation methods must be defined.
2. A Web service needs to be published to one or more intranet or Internet repositories for potential users to locate.
3. A Web service needs to be located to be invoked by potential users.
4. A Web service needs to be invoked to be of any benefit.
5. A Web service may need to be unpublished when it is no longer available or needed.

Web Services architecture then requires three fundamental operations: publish, find, and bind. Service providers publish services to a service broker. Service requesters find

required services using a service broker and bind to them. These ideas are shown in the following figure 1.

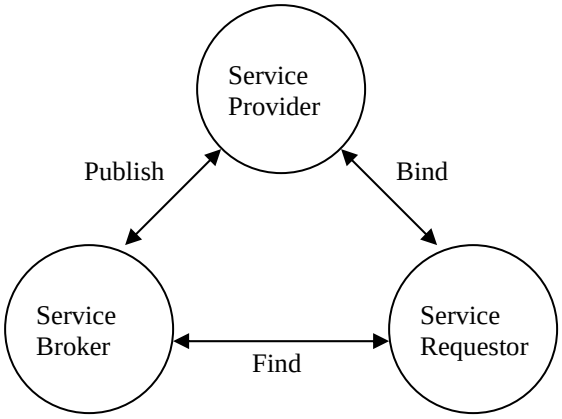


Fig 2.2.1: Web Service Architecture [W3C04]

In this architecture a service is an implementation of a service description and a service description is the metadata describing the service. This metadata must include sufficient information for a service requestor to access the service it describes including its interface and location; resource discovery metadata such as classification may also be included.

A service provider publishes a service description to a service registry. A service requester then finds the service description via the service registry; the services description contains sufficient information for the service requestor to bind to the service provider to use the service.

Although the web services architecture can be considered independently of any particular standards, clearly interoperability is required for large-scale adoption of the architecture.

A number of key industry leaders have been working to develop a set of XML-based open standards that enable the web services architecture to be implemented.

That includes, a standard way of capturing service descriptions is necessary, the web services description language (WSDL). The Simple Object Access Protocol (SOAP) is a standard for XML-based information exchange between distributed applications. And Universal Discovery, Description and Integration (UDDI) is a specification for distributed registries of web services.

2.2.1 Web Services Description Language (WSDL)

If a web service is to internet programming what a method is to programming in Java, a WSDL document is like a method signature. A WSDL document is XML, and a XML schema exists to define the structure of valid WSDL documents at http://schemas.xmlsoap.org/wsdl/. [WSDL01]

Below is an example of WSDL for a service which takes a term and returns a definition for that term. The example defines two messages, getTermRequest and getTermResponse. A single web service operation is defined called getTerm which takes as input a getTermRequest SOAP message and returns as output a getTermResponse SOAP message. The getTermRequest and getTermResponse messages are defined as having one part each, which is a string. Each part is analogous to a method parameter. Here, the types of the parts are defined as conforming to the xs:string schema. The author of the web service may designate a custom schema for the structure of the parts included in a message. In this way, the SOAP message can encapsulate arbitrary input and output for a service.

```
<message name="getTermRequest">
   <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
   <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
     <input message="getTermRequest"/>
     <output message="getTermResponse"/>
  </operation>
</portType>
```

Figure 2.2.2: An example WSDL document [WSDL01]

2.2.2 Simple Object Access Protocol (SOAP)

SOAP is an XML-based standard developed to simplify communications between applications over a distributed environment. Prior to the development of SOAP, applications communicated via remote procedure calls (RPC) using CORBA (Common Object Request Broker Architecture) or DCOM (Distributed Component Object Model) standards. Remote procedure calls, however, are proprietary binary formats that may pose logistical and security challenges in implementation.

A SOAP message is a specially formatted XML document. The advantages of using the XML framework for a SOAP message include human-readability, the availability of existing parser technology, and the fact that XML is an open standard. In a SOAP formatted XML document, there are three major components (with default namespace format in parentheses): [SOAP04]

- a mandatory SOAP envelope (env:Envelope)

- an optional SOAP header (env:Header)
- a mandatory SOAP body (env:Body)

In addition to the structure of a SOAP message, it is important to understand what protocols can be used to transmit SOAP messages. One of the main objectives with SOAP was to provide a message exchange system that could sit on top of existing, open technologies. Currently, one of the most common protocols in use is the HTTP connection model. Because HTTP transactions contain a request and a response, a SOAP message sent in the HTTP request can be easily associated with a SOAP message returned in the HTTP response. Thus, the HTTP binding provides an efficient and effective transfer protocol for SOAP message exchange. It is important to note that, thanks to its XML foundation, a SOAP message may also be transmitted via SMTP, FTP and other similar protocols.

2.2.3 Universal Discovery, Description and Integration (UDDI)

If web services are self-describing via their WSDL definitions, then callers need a way to discover these descriptions and find a required service. UDDI is a protocol for finding businesses and the services they provide. WSDL is the description provided by a UDDI registry. A registry is a web service itself, so communication with the registry is through SOAP. [UDDI]

UDDI is not a single particular registry or registry implementation though. It is a specification for a registry. Organizations may maintain their own registries. In general, a business must explicitly register itself and its services with a particular UDDI registry. An example of such Registry is the one provided by StrikeIron Inc [SI04]. It is at least

theoretically possible to automate the population of a registry or make a registry more dynamic, since UDDI only specifies the input and output requirements for a registry and not the registry implementation. Like any web service, a particular registry may be public or private. Today there are 6 major implementations of UDDI 2.0, which include:

- IBM WSTK and UDDI4J
- Systinet WASP UDDI
- jUDDI.org
- UDDI 2.0 in Java
- Microsoft UDDI SDK
- Trenian Web Services Directory

UDDI exposes the concept of a locator. A locator accepts key/value pairs and returns web services which satisfy those criteria. Locators may be used to find services based on geography, business purpose, industry, capabilities, or anything imaginable.

Without UDDI, calls to web services would have to be hard coded into applications and processes. With UDDI, a required web service may be dynamically located. In the case of a travel agency, the travel agency may dynamically locate the services for rental car agencies in a particular geography by using a UDDI locator for geography and a locator for business sector. A web service programmer may still need to hard code references to the locators or at least one initial registry.

UDDI and web services are not a panacea for the problem of dynamic code execution. While these standards provide protocols for communication, they rightly leave the structure of the data up to the owner of the web service. So, some agreement is needed between the two parties involved in a web service call. They must agree on the structure,

or XML schema, of the message in advance. Some industries and organizations have set or proposed standard XML schemas for their domains, but there is no requirement that a publisher of a service accepts certain input or returns certain output, and for good reason. Web services are not data integration technology, they are component integration technologies. So, this issue is left to other technologies. The Resource Description Framework (RDF) [RDF98] and the semantic web hold some promise of being able to communicate more dynamically by translating data.

Trust and security concerns are left to other technologies as well. In particular, trust and security may be addressed by using certificates for authentication, SOAP over HTTP for secure communication, and trusted third parties for establishing relationships. In some cases, the UDDI registry may be a trusted third party for the two parties involved in the service, so a service may be trusted if its discovered through a trusted registry, but in the case of a free and public registry, another party would be required.

2.3 Service Oriented Architecture

Service Oriented Architecture (SOA) is a programming model for building applications from service components. Although the SOA model does not apply strictly to web services, it could be made applicable to an alternative service model, web services are the most popular services to which it is applied.

The SOA model allows application components to be tied together only loosely. Individual web services can be developed be domain experts, leaving the interaction of the components to be determined, possibly at a later time, in a higher level language that

is more accessible to non-programmers. Components can also communicate in a platform independent manner. J2EE components can talk to .NET components.

Services for airline flight availability, prices, rental car info, hotel rates, and reservations for all these could be tied together to form an application to make travel reservations. Some of these services might be public, and some private. For example, a private service might be used to obtain user preference data from the user store of the originating travel agency and preference information for the same user from the companies providing the actual services. This preference data may form part of the input into the services to make the reservations.

The possible flow of the communication and interaction of the individual services can be specified in a higher level language. In particular, WS-CDL can be used to specify "choreographies" [WSCDL]. Choreographies are not executable. WS-CDL is used to describe possible interactions, including definitions for participants and roles, but it does not specify logic. Existing choreographies and services can be combined to form new choreographies. In the travel agency example, choreography for a particular rental car agency may define the interaction between a user requesting a car - where the "user" may be another application or service - and the rental car agency itself.

Since WSDL-CDL is not executable, a language is needed which can explicitly relate web service definitions in WSDL into a business processes. One such language is Business Process Execution Language (BPEL). BPEL is not particular to web services, but BPEL4WS is [Andrews03]. BPEL may be applied to Java executables or applied at a more abstract level to document workflow for instance.

BPEL allows the exact flow and execution of the applications to be defined in terms of its constituent web services. In the travel agency example, a BPEL process may define the following simple linear process.

1. Request information from the user (travel dates, source, destination, maximum price)
2. Combine user input with stored preferences (food preference for flight, car type, etc)
3. Make an airline reservation
    a) Query all airlines for availability
    b) Feed airline data into a service to find the best match for the user
    c) Reserve the best flight
4. Rent a car
    a) Query all rental car companies
    b) Feed airline data into a service to find the best match for the user
    c) Reserve the car

This is a simple example. In a more complex example, branching may be required based on conditionals such as user age or insurance information for the rental car, or whether the flight is domestic or international. Numbers three and four may be defined as sub-processes. Note that although the web services may be implemented over the web, the user facing part of this flow does not need to be implemented as a web application.

Some graphical tools exist to build BPEL flows. IBM has graphical tools to build UML and relate it to BPEL.

2.4 WS-Policy

With all these web services interacting in a web services programming model, there is a need not only to describe the web service input and output, but also to define policies for how those services interact. While WSDL describes the services, input, and output, the policy describes the capabilities and requirements of the service. WS-Policy is an emerging technology from IBM, BEA, Microsoft, VeriSign, and SAP. The policy can reflect high level business requirements like "only users over 25 may use this service to rent a car" and describe what is allowed in terms of the service. Policies can describe statements of intent or quality of service requirements like "this service is available between 9am and 5pm or when the markets are open".

Policies are generally not directly executable. As with WSDL, policies are purely descriptive. The WS-Policy implementation specifies that policies are SOAP attachments to WSDL. WS-Policy is just one possible implementation of a policy specification and is not a W3C standard, but it has become clear that there is a need to define the requirements and conditions under which services may be executed, and a policy solution fills this niche.

2.5 Implementing Web Services

While all the specifications discussed thus far are platform independent, the actual implementation of the web service is platform dependant. Through JSR-109 [JSR03], J2EE provides one standard for implementing web services. JSR 109 defines a “webservice.xml” file which defines a web service to the server much in the way a “web.xml” is able to define servlets. This differs from the WSDL file which is a

description of the web service to external agents and not a definition of the web service to the server. A definition is needed to describe how the web service is implemented, which is outside the scope of WSDL. In the J2EE implementation, a “mapping.xml” file defines how those web services bind to the Java classes which implement them.

When implemented over HTTP on a standard HTTP port, web services can also work through some of the common firewall security measures without having to change any of the filter rules. This can be an advantage so that application can communicate though firewalls, but it can also be a disadvantage in that it becomes easy for a web services programmer to open security holes. The security problem is not really intrinsic to web services, but is a problem of any remote execution protocol which makes itself publicly available.

As with J2EE, .NET also provides a mechanism for defining web services based on methods in an application. Since most of the major software industry leaders are participating, web services are starting to realize the promise ubiquitous, loosely coupled applications, even binding applications between software platforms such as J2EE and .NET.

3. Fault Tolerance and Reliability

3.1 Definition

Fault-tolerant describes a computer system or component designed so that, in the event that a component fails, a backup component or procedure can immediately take its place with no loss of service. Fault tolerance can be provided with software, or embedded in hardware, or provided by some combination. [Arora93]

3.2 Fault Models

A fault represents an erroneous state of hardware or software and an error is a manifestation of this fault. The use of an erroneous information leads to a failure. It is not necessary that every fault results in a failure. [Jag95] Depending on the system behavior after a fault has occurred, faults have been characterized into different groups or classes.

3.2.1 Byzantine Faults

The Byzantine fault model [Lamport82] represents the most adversarial model of failure. This fault model allows failed nodes to continue interacting with the rest of the system. Behavior can be arbitrary and inconsistent, and failed nodes are allowed to collude in order to devise more malicious output. Correctly operating nodes cannot automatically detect that any nodes have failed, nor do they know which nodes in particular have failed if the existence of a failure is known. This model can represent random system failures as well as malicious attacks by a hacker. It has been proven that no guarantees can be made

concerning correct operation of a system of 3m + 1 nodes if more than m nodes are experiencing Byzantine failures [Lamport82].

3.2.2 Fail Stop Faults

The fail-stop fault model [Sch83] is much simpler than the Byzantine model. This model allows any node to fail at any time, but when the failure occurs it ceases producing output and interacting with the rest of the system. Furthermore, all other nodes automatically know that the node has failed. This fault model represents common modes of failure such as a system hang or crash, but does not handle more subtle failures such as random memory corruption [Sch83].

3.2.3 Fail Stutter Faults

The Byzantine fault model is extremely broad, perhaps unrealistically so, and is therefore extremely difficult to analyze or tolerate. The fail-stop model is commonly used when presenting fault-tolerance techniques, but it is often criticized as overly simplistic due to its failure to represent many types of real-world failures. The fail-stutter fault model [Dusseau01] is an attempt to provide a middle ground model between these two extremes. The fail-stutter model is an extension of the fail-stop model. It attempts to maintain the tractability of that model while expanding the set of real-world faults that it includes. The fail-stutter model includes all provisions of the fail-stop model, but it also allows for performance faults. A performance fault is an event in which a component provides unexpectedly low performance, but to continues to function correctly with regard to its output. This extension allows the model to include faults such as poor latency performance of a network switch when suddenly hit with a very high traffic load.

Despite its advantages, however, this fault model has yet to see widespread acceptance by the community.

3.3 Specification of a Faulty System

"A system is said to be faulty if either or both of its safety and liveness properties are violated following a faulty action. A faulty system consists of a fault-intolerant system P, and one or more fault actions F executed by an adversary. A computation of P in the presence of F is a sequence of states s0, s1, … that is weakly-fair and P-maximal: for each $j>0$, s_j is obtained from s_{j-1} by executing an enabled action of P or F. Weak fairness means that each action of P that is continuously enabled is eventually chosen for execution, and P-maximality means that if the computation is finite, then the guard of each action in P is false in the final state. We assume that the number of consecutive F-actions in any schedule is bounded." [Arora98]

When a fault occurs in a system and is encountered during execution, a process experiences an unwanted state transition (into an error-state). Thus, by extending the state space of a process through addition of extra variables, every kind of fault can be simulated [Arora93]. If the erroneous state manifests itself to the end-user through an anomalous or unexpected behavior, we say that a failure occurred.

In [Arora93], the authors have formally defined what it means for a system to tolerate a class of faults based on two conditions: closure and convergence. One, if a fault occurs when the system state is within a set of 'legal' states, the resulting state is within some

larger set and, if fault continue occurring, the system state remains within that larger state space (Closure).Secondly, if faults stop occurring, the system eventually reaches a state within the legal set (Convergence).

3.4 Fault Tolerance Models

A fault is a manifestation of an unexpected behavior, and fault tolerance is a mechanism that masks or restores the expanded behavior of a system following the occurrence of faults. To confine computer failures, a system must automatically check execution results for the errors that could lead to failure. There are two main approaches to detecting error-states caused during execution by design faults:

1. Acceptance tests of the results via executable assertions. The notion of acceptance testing is essential to all fault tolerance voting or consensus arrangements. The acceptance test may determine functional equivalence or may determine correctness measured against an always-correct result, an oracle.
2. Alternate Version(s): The notion of an alternate version is essential to most fault tolerance voting or consensus arrangements. We first consider redundancy and then independent versions. Two-out-of-three majority voting is a tolerant and low cost system that can be implemented with either replicas or independent versions.

In a diversified design, the different systems produced from a common service specification are called variants. A diversified design has at least two variants plus a decider, which monitors the results of variant execution, given consistent initial conditions and inputs. The common specification must explicitly address the decision

points, that is, it must state when to make decisions and what data to base them on (the data processed by the decider).

The best-documented techniques for tolerating faults in software-based systems are the recovery block (RB) approach [Randell87] and N-version programming (NVP) [Aviz85]. In the first approach, the variants are called alternates and the decider is an acceptance test, which is applied sequentially to the alternates' results. If the results of the primary alternate do not satisfy the acceptance test, the secondary alternate executes. In the second approach, the variants are called versions, and the decider is a vote based on all versions' results.

3.4.1 Recovery Blocks

One of the common fault-tolerant software schemes is the recovery block [e.g., Randell,87, Vouk96]. As shown in figure 3.4.1, the output of a module is tested for acceptability with an acceptance test. If the acceptance test determines that the output is not acceptable, it rolls back to a state before the module was executed. It then asks the second module to execute, and so on.

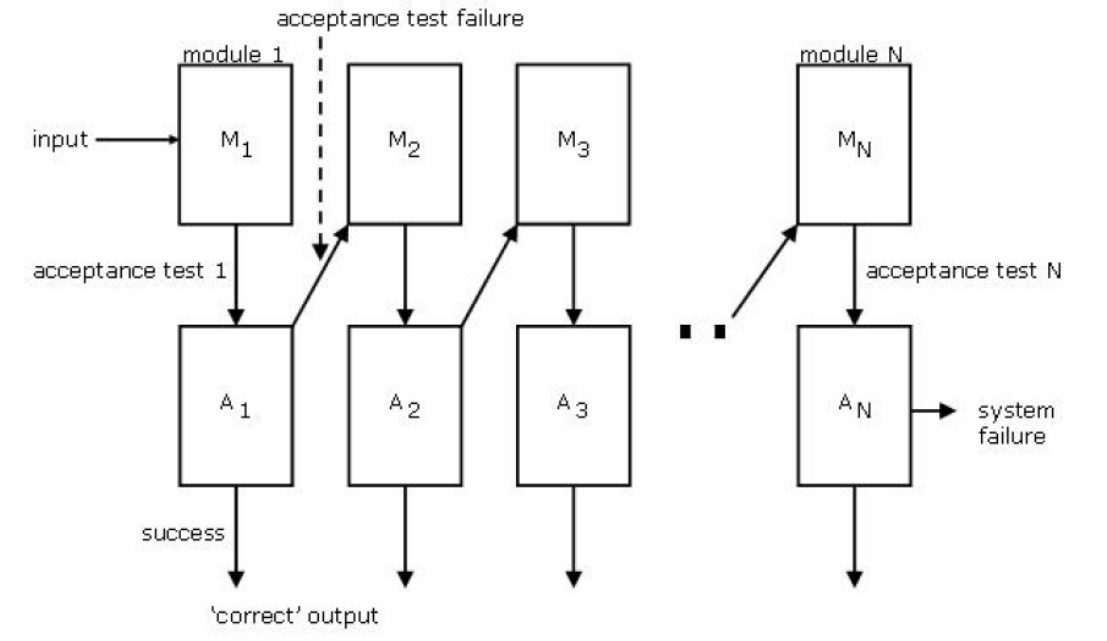


Figure 3.4.1 Recovery Blocks [Vouk96]

3.4.2 N-Version Programming

N-version programming proposes parallel execution of N alternate versions [e.g., Aviz85, Vouk96]. A module must then aggregate the N results and perform adjudication of their outputs. Part of this adjudication module can be a voter. Each version can have a weighted vote or equal vote (Figure 3.4.2).

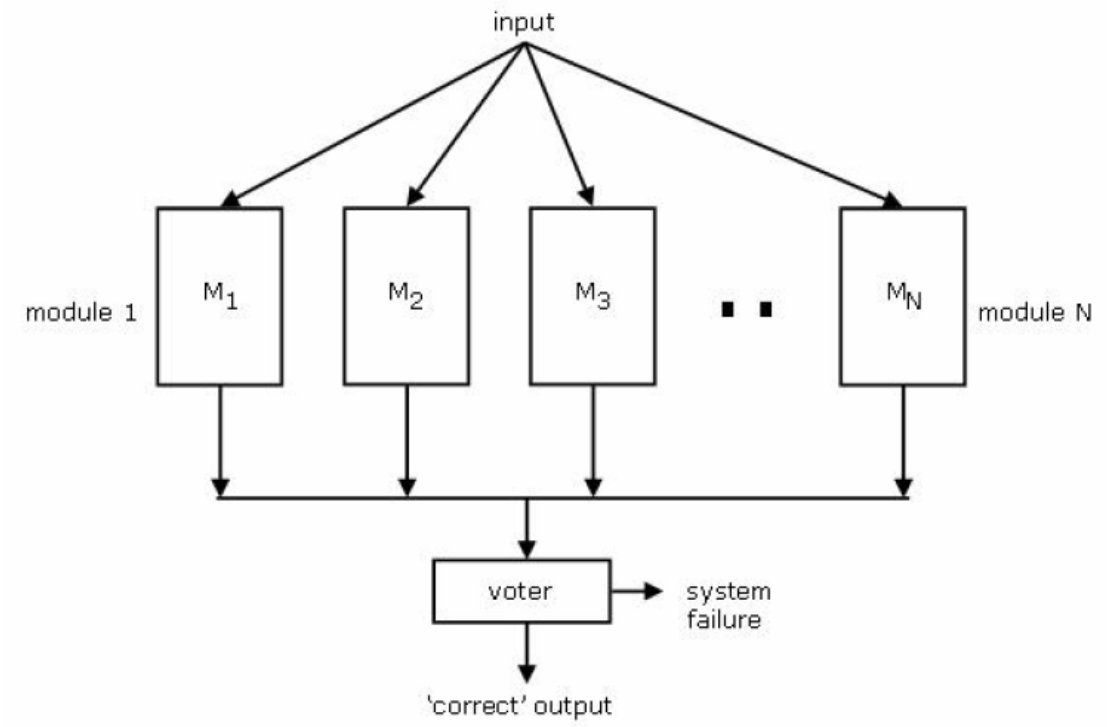


Figure 3.4.2 N-Version Programming [Vouk96]

Consensus voting is a generalization of majority voting. It uses the following Algorithm [e.g., Vouk96] to select which answer to use:

If there is a majority agreement (m >= ⌈(N+1)/2⌉, N > 1), then this answer is choose as the correct answer.

Otherwise, if there is a unique maximum agreement, but this number of agreeing versions is less then ⌈(N+1)/2⌉, then this answer chosen is the correct one.

Otherwise, if there is a tie in the maximum agreement number from several output groups, then If consensus voting is used in N-version programming, one group is chosen at random and the answer associated with this group is chosen as the correct one.

Else if consensus voting is used in consensus recovery block, all groups are subjected to the acceptance test, which is then used to choose the correct output.

There are of course many other strategies [e.g., Laprie90, Vouk96]. Table 3.4.1 summarizes the main characteristics of the two strategies mentioned above. In selecting a

strategy for a given application, pay particular attention to the method for judging result acceptability and whether service delivery is suspended when an error occurs.

Method	Error-Processing Technique	Judgment on Result Acceptability	Variant-Execution Scheme	Consistency of Input Data	Suspension of Service Delivery During Error Processing	No. Variants to Tolerate f Sequential Faults
Recovery Blocks (RB)	Error detection by acceptance tests and backward recovery	Absolute, with respect to specification	Sequential	Implicit, from backward recovery principle	Yes, duration necessary for executing one or more variants	f+1
N-Version Programming (NVP)	Vote	Relative, on variant results	Parallel	Explicit, by dedicated mechanisms	No	f+2

Table 3.4.1 Main characteristics of fault-tolerance strategies [Laprie90]

3.5 Fault Tolerance in Web Services

The classical fault tolerance models discussed above can also be applied to web services. For example when it comes to acceptance testing, there are several different types of general tests that are applicable when considering fault-tolerance in a service-orientated system. For example there is the notion of XSD Schema Validation. There are several free and commercial libraries that check a XML document against a namespace definition. We can, at minimum, check to see if all services return valid SOAP envelope documents. Often, the content of the SOAP header is XML that complies with namespace restriction to provide meta-data. The structure and content of the SOAP header is important when implementing WS-Security, WS-Availability, and related standards. The SOAP body is also usually a document, and then the service is called a document-style

service. We can compare two SOAP document for namespace equivalence, schema syntax equivalence, and check that the data contained in the response documents are identical or approximate. Now when it comes to the notion of Alternate versions, the same concept can also be applied to web services. We can have several replicas deployed on different servers, or we can have multiple implementation of the same service distributed on different servers as well.

Going back to section 3.4.1, Recovery Block is one of the most common Fault Tolerance Schemes. The output of a module, or in this case a service, is tested for acceptability with an acceptance test. If the acceptance test determines that the output is not acceptable, it rolls back to a state before the service was executed. It then asks the second service to execute, and so on. The problem with recovery block is choosing what acceptance test to use. The most straightforward acceptance test is probably to the Validate SOAP Body test describe above. This is a strong acceptance test because it guarantees that the program will be able to continue executing, however it does nothing to check the correctness of data in the output document. In section 3.4.2, we introduced the N-version programming model. N-version programming has several synergies with the web service systems. Obviously, each web service can correspond to a version. Another web service may be an adjudication mechanism.

Fault Tolerance can also be provided using Intermediary web services [Chi02]. Unfortunately, most intermediary web services, today, are point solutions to very narrow problems. Consensus voting web services hold the potential to dramatically reduce the costs of adding fault-tolerance to a web service system. Acceptance tests can be

performance on any pair (or more) of web services without any regard to the domain of the application. A general purpose web service using schema validation as the acceptance test would allow for redundancy or voting models, so long as there are multiples versions. However, one could provide these multiple versions from independent servers or they may all be hosted on the same server.

Fault Tolerance can also be implemented using the client as the side responsible for it. A service-specific acceptance test could be integrated into the client. The advantage over using a general SOAP or WSDL level acceptance test is that the client has complete control over when a result is acceptable. The client could invoke several services in parallel. Alternatively, the service could invoke services sequentially. This may be more effective for performing 2-out-of-3 consensus voting when the first two services agree and the third is only invoked as a tiebreaker.

Other Model of Fault tolerance for web service exists too, notably the Client-Transparent scheme [Tamir00] [Liang03]. These models propose an interesting approach in providing fault tolerance, and that being totally transparent to the client. A major drawback though is that they require modifications either at the Transport Layer [Tamir00], or to the SOAP protocol [Liang03]. That’s why I will not go in detail explaining those models because they don’t apply to the work being done in this thesis.

4. Scientific Workflows: Concept and architecture

4.1 Historical Perspective

The most widely used definition of term workflow is presented by the workflow management coalition [WFMC04]. Workflow can be defined as the orchestration of a set of activities to accomplish a larger and sophisticated goal, referred to as a business process. Mathematical and Computer Science academic work on workflow dates back to the mid-70s. The most significant was the work done on “Office Automation” prototypes at Xerox Park. Work in this area went on from 1975-1985, but did not make significant progress until the business process re-engineering trends of the 1990s. Several early commercial products were also created, with Lotus Domino [Domino04] being the most successful.

In the 1990s, the automation of business workflow has matured into many successfully software products such as Domino [Domino04] and Websphere [WebSphere05]. By 2000, Research on workflow systems was reopened from a different perspective when work began on Web Service Choreography Interface Specification (WSCI), and introduced the notions of web service composition and service orchestration. As part of their commitment to adopting web services and XML, Microsoft and IBM developed XLANG [Xlang05] and WSFL [WSFL05] respectively. In 2002, Microsoft and IBM jointly announced the BPEL workflow language specification. The specification [BPEL4WS04] was finalized with input from several other partners including BEA, and then rolled into a working group at OASIS [OASIS05].

Over the last 2 years, XML meta-languages have gain a lot of attention in both technical and business publications. Simultaneously, many academic projects began using these languages or systems derived from these languages.

4.2 Workflow Characteristics

The following list covers most of the principal scientific workflow requirements: [Chandra02] [Altintas03]

- **Interface:** Intuitive design and execution environment for end user.
- **Re-use:** Component reusability and exchangeability; in particular, ability to dynamically add new process nodes and data sets to a workflow (e.g., using "plug-ins" for Web services and data), or change existing ones.
- **Data transforms:** Extensive and flexible data format transformation support to mediate between consecutive processing steps and between the data sources and data sinks.
- **Interaction and Batch:** Support for user interaction (including process steering) during process execution. Ability to monitor an automated processes in real-time, to "background" it and retrieve results later, stop/resume options with long-term state memory, etc.
- **Streaming:** Support for data streaming and both data-moderate and data-intensive applications.
- **Locality**: Support for local and distributed computation and data access.

- **Interoperability:** Ability to exchange workflow descriptions. Interoperability with, and integration of, other workflow and scientific data collection, management, and analysis systems.
- • **Complexity:** Ability to handle complex control, data, and event flows.
- **Performance and Planning:** Ability to predict performance and costs of a workflow (planning) and the ability to collect data for different process, planning and audit metrics. Planning may include details such as resources on a particular host, and data transport mechanisms. When different components/tasks of the workflow have to be executed on different machines, explicit specification of how to move the data (especially large amounts of data) among nodes may be needed.
- **Dependability:** Workflow engine and resources need to be reliable, highly available, have appropriate longevity to justify investment, fault-tolerance, recoverability, etc.
- **Verification and Validation:** Ability to verify and validate constructed workflows, imported workflows, and results obtained through an automated workflow. Obviously, to properly balance all the requirements is an extremely challenging task. This has been recognized by software engineering for decades. Workflows are no different. We plan to heuristically maximize principal returns an end-user decides on, but we will not attempt to optimize the complete solution.

4.3 Services

It is efficient to have scientific tools for analysis and data manipulation distributed across various remote sites. One approach to use distributed computing is presented in Grid

Services. [Col04] offers a more detailed description of Grid Services. Another basic assumption is that scientists do want to re-use tools and facilities in performing workflow sub-tasks, and that the only programming and basic tool development will be very domain specific (e.g., software that performs astrophysics simulations). It seems reasonable that these sub-tasks be made available to the end-user as Web Services that can be used and executed seamlessly from the user's system. These services may be chained together in various different ways to design an executable workflow.

4.4 Graphical User Interfaces (GUIs)

The user interface is an important component of a workflow construction and operational management architecture. In the case of scientific workflows, it is the primary objective of the interface to hide appropriately the technical details of the underlying information technology from the end-user, and yet at the same time help the user convert the abstract ideas into executable workflows, and run those workflows, without limiting the end-user productivity. [Bha05]

If the user interface is too complex or confusing, the user will either spend too much time figuring out how to use the system, make too many mistakes, or will give up and will choose to use an alternative, such as a manual way of running the workflow. [Bha05] offers a more detailed description of Graphical User Interfaces, plus a comparison between the different GUIs used in Scientific Workflows.

4.5 Scientific Process Automation (SPA)

4.5.1 Introduction

Typically, when a scientist conducts data-driven experiments, he uses various analysis and querying tools in sequence, and in most cases, results (i.e. outputs from) one step of the experiment are fed as inputs to the next. Such tools may be local or remote for the user, and are typically used one tool at a time, although many scientists would prefer parallel execution of services wherever needed. The user has an option of trying to semi-automate this process by writing a script in a scripting language say, Perl, that connects these tools in a pre-defined sequence.

One of the primary goals of the Scientific Data Management initiative is process automation, and we wish to create a system that allows complete automation of a scientific research process as described above using a graphical user interface.

4.5.2 Background

The SciDAC (Scientific Discovery through Advanced Computing, [SciDAC04]) program, launched in 2001, is a five-year program sponsored by the U.S Department of Energy that aims to develop the Scientific Computing Software and Hardware Infrastructure needed to use terascale computers to advance its research programs in basic energy sciences, biological and environmental research, fusion energy sciences, and high energy and nuclear physics.

The SPA project [SPA05] is a project under the Scientific Data Management Center (SDM) of the SciDAC program that aims to develop tools that adapt and extend current technology for scientific workflows. It enables a scientist to create workflows in a collaborative environment using network-based tools. The SPA project is based on UC Berkeley's Ptolemy project [Ptolemy04], and is a part of a larger project, Kepler [Kepler04]. Kepler's goal is to produce an open-source scientific workflow system that allows scientists to design scientific workflows and execute those efficiently using emerging Grid-based approaches to distributed computation.

The Ptolemy II project [Ptolemy04] was developed for support of heterogeneous modeling, simulation, and design of concurrent systems. Ptolemy II is a set of Java packages that supports clustered hierarchical graphs, where each graph is a collection of entities and the relations between the entities. The executable entities are called "Actors", and the software comes with Actor Packages which is a library of pre-defined reusable components such as these with focus on embedded systems, particularly those that mix technologies. Ptolemy II also includes a number of support packages, such as graph, math, plot, data, etc. Models of computation are imposed on the relation between entities, in the form of "Directors" such as discrete-event, synchronous, sequential events, etc. Ptolemy II comes with an intuitive user-interface called "Vergil" that allows users to construct complex workflows to the desired level of abstraction using Actors and a Director for each workflow.

4.5.3 SPA System Description

The SPA (Scientific Process Automation) [SPA05] initiative of the SDM center extends the Ptolemy II system to enable it to construct and execute scientific workflows. The current capabilities of the SPA/Kepler systems have been used in various scientific domains, including molecular biology, ecology, geosciences, chemistry and oceanography. Details of these workflows and their implementations are given in [Kepler04].

The SPA system aims to include the features mentioned below:

1. An intuitive graphical user interface. The user interface should be easy to understand and use. The user should also be able to implement it as a part of his experimental research easily. The interface should have good steering capabilities, such that the user may pause and resume, or stop the execution of the workflow as and when needed. He should also be able to fully automate the process such that any intervention from his side is not required at any time, so he may run such a workflow in the background.
2. A central component or resource repository. Such a repository would be a collection for reusable components (or services) that will be used for constructing workflows. We also develop a means by which scientists can easily download “patches” to the existing version of the system to download newly created services.
3. Support for local and distributed service components, possibly attached to a grid network.

4. Support for publishing workflows, and exchanging workflows with other scientists, and support for data and workflow provenance.

Many of these features have been already implemented, and the rest are being worked on. Following are the more prominent characteristics of Ptolemy-based SPA as it exists right now:

1. Workflow Composition and Execution: SPA provides a designer to compose workflows. Tasks (or as they are called Actors here) form the components of a workflow. Each designed workflow has an engine (here Director) that initiates and monitors the execution of a workflow.
2. Retrieving Saved Workflows: In PtolemyII [Ptolemy04] workflows can be saved as XML/MOML descriptions. It can be invoked later and used with different inputs.
3. Visualization: PtolemyII has special applications (graph plot, Matlab) and libraries, which provide better visualization of data.
4. Scalability: PtolemyII is based on Java technology and scalability is achieved by developing modules or components that interface with the existing interfaces of PtolemyII.

One actor of special interest for this thesis is the Web service Actor. It is the actor responsible for invoking web services. This actor will be discussed in greater detail later in chapter 5 when we discuss the implementation of the Fault Tolerance approach.

Detailed workflows and their implementations are given in [Kepler04]. An example workflow is the Promoter Identification Workflow (PIW) shown in figure 4.5.1. This workflow is used to demonstrate Ptolemy software [Ptolemy04], and invokes three web services, which are currently provided by our research group at NC State. The Services are BLAST, ClustalW and Transfac. For a detailed description of the PIW workflow, please refer to Appendix 1 and to [Kepler04] and [Bha05].

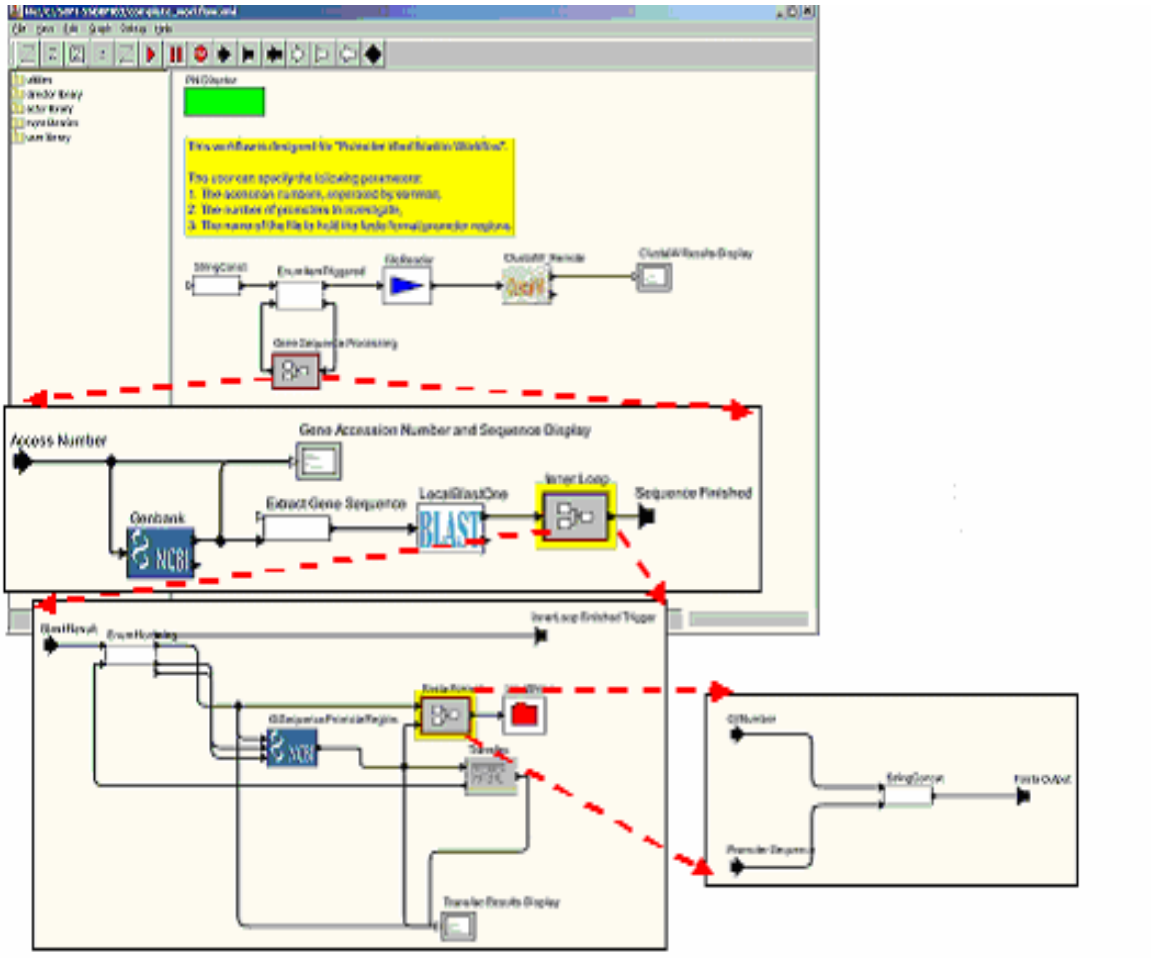


Figure 4.5.1: PIW Workflow and Sub-Workflows in the SPA system [SPA05]

5. Fault Tolerance in Scientific Workflows

5.1 Some Challenges Currently Facing Scientific Workflows

The issues discussed in the following paragraphs do not represent a comprehensive list, but are currently appear to be some of the most prominent ones. Interestingly, four out of five are in some way closely related to the issues of workflow reliability and availability.

5.1.1 Workflow Specification

Workflows can be specified over a number of ”views”. The “views” represent different aspects of the workflow, such as control flows, data flows (including I/O), event-flows, software components, computational elements, storage components etc. For example, the control layer allows the workflow to describe the sequences of tasks to be performed (expressed as actigrams or datagrams), where each task can invoke one or more software components. [DOE04]

- **Workflows – or Flow layer-** Describes execution ordering of tasks by using different views of sequencing, branching (decision making, parallelism) and feedback (loops), different constructors that permit flow of execution control. Examples of constructors are sequence, choice, parallelism, and join-synchronization. Tasks in their elementary form are atomic units of work. They may also invoke other applications and/or tools. In compound form, a task can be a ***sub-workflow,*** i.e**.,** a module consisting of an ordered execution of a set of tasks. A workflow language is needed to efficiently support execution, interoperability and exchange of this information.

- **Components - Application and Software Tools layer** – This layer describes the invoked applications and software tools used by the workflow tasks. In most cases there is a one-to-one correspondence between tasks and invoked applications but that is not necessarily always the case. Additional narrative explanations can also describe the invocation mechanism, i.e., CORBA, web services, grid services, other forms of service.
- **I/O System layer-** This layer describes the input/output (I/O) and middleware that allow efficient read and write operations by the applications. Predicted data volumes and their characteristics, such as streaming granularity, can also be described on this layer.
- **Computational, Networking and Storage Infrastructure – or Storage and Networked Resource layer** – This layer provides information about physical devices used by the tasks during their executions. Such information may include performance related issues such as the required data transfer rates, network topologies, computational and storage platforms, etc.

Graphical representations of the control and data flows found in scientific workflows have scalability limitations when the number of components becomes large. There is a need for an executable scientific workflow language that can describe effectively all layers of the workflow including:

- Inputs and outputs for each workflow component
- The metadata of the workflow
- Granularity of the tasks, sub-workflows, …
- Task invocation methods- e.g., web services, corba, wrappers, callbacks, …

- Human tasks: notifications and alerts, steering, pauses, etc.
- Dataflow streaming granularity (bytes, packets, files, buffers, …)
- Performance expectations
- Semantics of the workflow and its components
- Verification and validation hooks
- Exception and Failure management
- Other …

While a number of workflow languages have been proposed [e.g., BPEL4WS04], as yet none of them provide for all of the requirements bullets. In the context of present work, the biggest gap is the one related to verification and validation of workflows themselves, and with respect to handling of workflow exceptions and failures. This includes description of desired fail-over options. For example, web and grid-service based workflows have some failings inherent in the nature of the definition of services, such as inability to specify "heart beat" function, or query as service, or its broker, about its reliability and availability [Col04]. Determination of functional equivalency is also an open issue.

Another issue that often arises is that of graphical user interfaces (GUIs) used to access scientific workflows. Issues often revolve around usability, stability, portability, etc. A more detailed description of current graphical user interfaces and their capabilities and limitations can be found in [Bha05].

5.1.2 Workflow execution in distributed systems -

Future scientific workflows may involve manipulation of data streams from and to thousands of heterogeneous source and sinks of information. Scalability and efficiency of handlings such flows grows in importance as the number of flows grows, otherwise trivial overhead can become a significant impediment when the scale grows. There will be situations where the data sources are not reproducible (e.g., time dependent astronomical sky surveys, sensor-based input streams), or it is simply too costly to do so. In such situation, workflow implementation must be able to accept the full flow of data without failing. Optimizing the load balancing of all elements of workflows is a particular challenge in a distributed system. [DOE04], and it becomes very important not have such a workflow fail since it may result in a permanent loss of one-time data.

5.1.3 Security

The handling of security in the workflow management system is another issue. Again failures in that space can have differing consequences, some very costly. It involves a number of issues. For example, the workflow management system will need provide appropriate authentication, authorization and access (AAA) controls that span a number of institutions and groups. Only authorized collaboration groups should be able to create, modify and monitor workflows they own In practical terms, this may involved matching of different credentials of one or more of the collaborators to enable the various workflow components to access the necessary resources. The workflow management system will have to balance the difficult task of using and protecting the credentials while they are in the custody of the system, and at the same time not interrupting or slowing down the

actual execution of the workflow. In the experience of the author, this may be major issue when it comes to full automation of the workflows since currently inter-institutional distributed AAA is not a solved problem despite very active research and a multitude of practical implementations.

5.1.4 Monitoring of long-running workflows

Monitoring of workflows for auditing, recovery, meta-data collection, and other purposes is another open issue. It can be done in both online and offline mode. Online monitoring is performed while the processes are running. It is designed to provide information about the current state of the local and outsourced workflows (for user, application, and other modules). Appropriately constructed, this information can be used to checkpoint the flows and recover the workflow in the case of a failure [e.g., Laprie90]. Online monitoring typically generates log files for real-time analysis and possible additional offline monitoring. Offline monitoring is performed by analysis of the log files and can help in workflow optimization and failure detection and recovery. Some of the benefits of workflow monitoring are:

- Predicting the future behaviour of a running workflow
- Enabling flexible decisions and deadline-miss prediction
- Providing knowledge about the current workflow (state, data, and timing)
- Supporting active notification about certain conditions
- Enabling dynamic workflow optimisation
- Enabling workflow failure protection and recovery (fault-tolerance)

- Allowing for off-line workflow analysis applicable to local and distributed workflows.

Complex scientific workflows will often run for long periods of time and will need to be able to recover from dynamic changes. One issue is how to handle well-defined fault conditions such as a down server, or a resource that does not have the ability to service the request at that moment (e.g. out of storage space) when the workflow manager tries use the resource. Another issue is how to treat failures that render resources (temporarily) inaccessible, but not necessarily inoperative. The problem of monitoring resources accessed over wide-area networks (WAN) is an example of this. If the network between the WFMS monitoring service and the resource that is acting on behalf of the workflow partitions (no communication is possible between the two halves), then the monitoring service cannot determine if the resource is still acting on behalf of the workflow manager. This issue will be discussed more thoroughly in later paragraphs, and we will present a preliminary solution to the problem where a service becomes temporarily or permanently inaccessible.

5.1.5 Adapting components to the framework

One of the major issues with current scientific workflows is not related to possible run-time failures, but does relate to workflow design issues. Easy integration of existing and new data-analysis and data-mining algorithms is an important part of the workflow flexibility expected by the scientists. Such data-mining modules should take advantage of the common data model of the framework in which they operate. This implies development of a workflow framework with interface standards that enable workflow

composition, automation, interoperability of workflow components, and extendibility. Integration often involves the creation of "adaptors" and "bridges" that allow for interoperation between technologies of different provenance. [DOE04]

5.1.6 Summary

A summary of the major issues that require further research and/or deployment efforts in the area of scientific workflows, as seen by the SDM center is shown in Table 5.1.5 [DOE04]. The ellipse marks the current position of the issues relative to pure R&D all the way on the left and off-the-shelf deployment all the way on the right.

Issues	Research and Development	Hardening and Packaging	Deployment and Maintenance	Comments
Granularity of tasks, sub-workflows				
Task Invocation				Wrappers for legacy codes
Human tasks: Notifications, alerts, steering, pausing				
Dataflow streaming granularity				Research needed both in the specification and implementation phases
Performance expectation				
Workflow engine for scientific applications				
Integrated data-flow management				
Failure detection and recovery				Especially needed for distributed and long-running workflows

Data-driven flow control		
Performance-driven flow control		
Workflow optimization		
Run-time resource coordination		
Workflow Security		
Data Marshaling		

Table 5.1.1 Summary of the major issues that require further research at SDM [DOE04]

As shown in the table, failure detection and recovery is still in the early stages. The following sections discuss some of solutions to part of that problem.

5.2 Modeling Fault Tolerance

There are two basic forms of run-time fault-tolerance: forward-recovery (which includes failure masking and redundancy based fail-over), and backward-recovery (which includes checkpointing) [e.g., Laprie90]. A run-time failures of a system is in fact the result of a series of events – sometimes that of very complex and unexpected interactions. Typically, either at design time or at execution time developer or researcher makes an error, or underlying infrastructure (including invoked services) fails for some reason. An initial design error can become a fault in the initial product. This fault may propagate (as series of defects) to the final executable version of the workflow. When the workflow encounters that defect during execution, the workflow enters an error-state. If that error-state becomes visible to the end-user, it becomes a failure that may have anywhere from

no consequences to catastrophic consequences. Similarly, a call to a workflow component that fails at run-time may again force the workflow into an error-state, and [Mus87, 90, 93. 98] manifest as a failure. So, run-time workflow failures can be caused by one or more of the following non comprehensive events:

- Use errors caused by end user, for example entering incorrect data.
- Error-state caused by network difficulties, such as congestion.
- Error-state initiated by workflow component failures (actors in the case of SPA/Kepler framework), due to a programming issue..
- Hardware Faults
- Error-states caused by failures in services, such as the unavailability of a certain service (e.g., actual web service, or a remote computational or storage service)
- Other …

This work deals with one of the above events only. It tries to address the problem that occurs when a web service isn't available. It also discusses how to recover from such a failure using redundancy. In doing so, two principal assumptions are made:

1. Failures of redundant services are not correlated. It basically means that the failure of one service doesn't affect other functionally equivalent service. For example, we assume that redundant services are hosted on separate servers, so that in case one server fails, only one service will be affected. However, assumption also implies that failures are not caused by a basic algorithmic flaw that may be present in both services and may result in identical but wrong responses from all redundant services [Vou96].

2. In discussing system reliability, we may make the assumption that all of the workflow services A have the same failure and recovery probability. This, of course may not be a realistic assumptions, but it provides a vehicle for the model discussion, and it serves this purpose well. An enormous amount of work has been already done by others in modeling and simulation of redundant components under a variety of conditions, and we refer the reader to that work for further detail [e.g., Laprie90, Vou96 and references therein]

The first assumption does not affect the case when no fault tolerance exists, since we also assume that different services used by the workflow are deployed in a serial fashion. Hence, the failure of one service means the failure of the entire workflow, since no alternative services are being provided. Of course, one could actually have different functions performed in parallel (and this is often done), yet in practical terms a failure of either of those services again fails the workflow. Therefore, serialization assumptions stand.

Several studies [M&M03] [Comp01] showed that today's web servers (as opposed to actual services) have an average failure probability of 0.045. During peak operation time, that failure probability may become as high as 0.2, depending on the request type and duration. We will use these two numbers to provide an illustration of the range of service failure probabilities against which one would have to guard, and to illustrate the power of a simple redundancy-based fault-tolerance strategy.

If we consider the scientific workflows presented in the Software Process Automation project [SPA05], they usually involve invoking several services, for example the Promoter Identifier Workflow mentioned in section 4.5.3 invokes 3 different services before it is done. And if one of those services fails, the workflow won't finish successfully. At the point of failure, we could recover back to the point where the workflow was checkpointed and re-run the remaining part, or we could try to mask the failure of the component service. We focus on the latter.

Given assumptions above, the probability that the workflow fails is the probability that at least one of the services fails. In other words:

$$P_F = 1 - \prod_{i=1}^{n} (1 - P_i) \qquad [1]$$

Where P_F be the probability that the workflow will fail,

P_i is the probability that service i will fail,

n is the number of services in the workflow.

In the Promoter Identifier Workflow example, based on equation 1, we would then have $P_F = 1 - (1-0.045)^3 = 0.129$, and in the "heavy load" case $P_F = 1 - (0.8)^3 = 0.488$.

Figure 5.2.1 shows a graph of the failure probability of a workflow given our two illustration failure probability. System failure probability grows considerably with the number of services (equation 1). In the "heavy load" situation, the failure probability is very close to one once the number of services in a workflow exceeds 15.

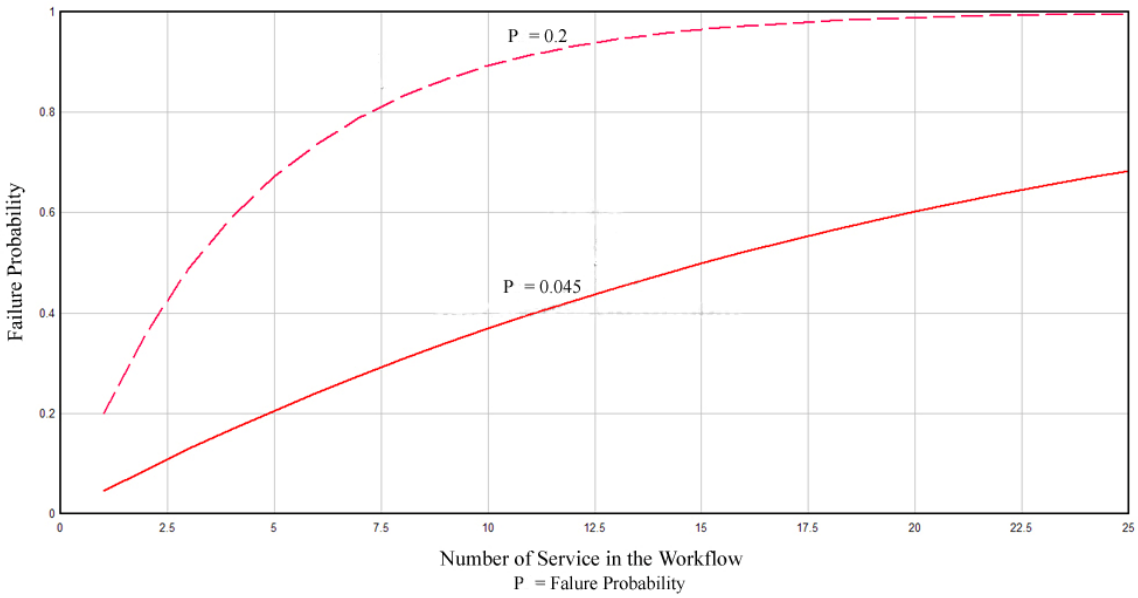


Figure 5.2.1 Failure probability of a serial system w/o fault tolerance

While the numbers in the graph may seem excessively pessimistic, it is clear that as the number of service in a complex workflow grows, service failures can have a dramatic effect on the whole system operation, especially if the workflow invokes over 10 services.

An obvious and well known solution is to use multiple parallel backup servers to counter physical infrastructure and networking failures, and/or to use alternative but functionally equivalent services if other types of failures may be suspected.. In case one of the services fails; the workflow would automatically switch to an alternative one.

Let’s assume that the probability that a service or any of its alternative fails is p. Then, the probability that such redundant service fails is that of a group of parallel components,

i.e., all of them need to fail before an end-user visible failure occurs. For example, the reliability of a service with 3 alternatives is the probability that at least one of the alternatives is operational, i.e., $R = (1-p) + p(1-p) + p^2(1-p) = 1 - p^3$ Generalizing:

$$R = \sum_{j=0}^{m-1} p^j (1-p) = 1 - p^m \qquad [2]$$

where m is the number of alternatives[1]. Applying [1] and [2] to the entire workflow, the failure probability of the workflow would be:

$$P_F = 1 - \left(\prod_{i=1}^{n} \sum_{j=0}^{m-1} p_i^{\,j} (1-p_i)\right) = 1 - \prod_{i=1}^{n} (1 - p_i^m) \qquad [3]$$

Where P_F is the probability that the workflow will fail,

n is the number of service in the workflow.

m is the number of replicas of each service in the workflow (in this example, assumed to be the same for each service).

p_i is the probability that service i will fail.

Consider again the PIW example (section 4.5.3) which has 3 "serial" services. Let each service have 2 backup services (m=3). Then the probability that at least one of the service will fail (thus the workflow will fail), and given our two illustration failures rates,

$P_F = 1 - (1 - 0.045^3)^3 = 0.00027$

and

$P_F = 1 - (1 - 0.2^3)^3 = 0.0238$

[1] Note: $(1-p) + p(1-p) + p^2(1-p) + \ldots + p^{m-1}(1-p) = 1 - p + p - p^2 + p^2 - p^3 + \ldots + p^{m-2} - p^{m-1} + p^{m-1} - p^m = (1 - p^m)$ which in turn is equal to [4].

Figure 5.2.2 illustrates different failure probabilities based on the fault tolerant service model for the PIW given assumption that p=0.045 for all services. Different lines represent the number of alternatives available for each service. We notice that, even when there is only one alternative for each service (m=2), failure probability is significantly lower that a workflow without any fault tolerance support. For example for a workflow with 25 services to invoke, the failure probability went down from almost 0.7 to 0.05 for m=2, to 0.002 for m=3 and 0.0001 for m=4. Notice that every time we added an alternative service, the failure probability went down by a factor of 20 or more.

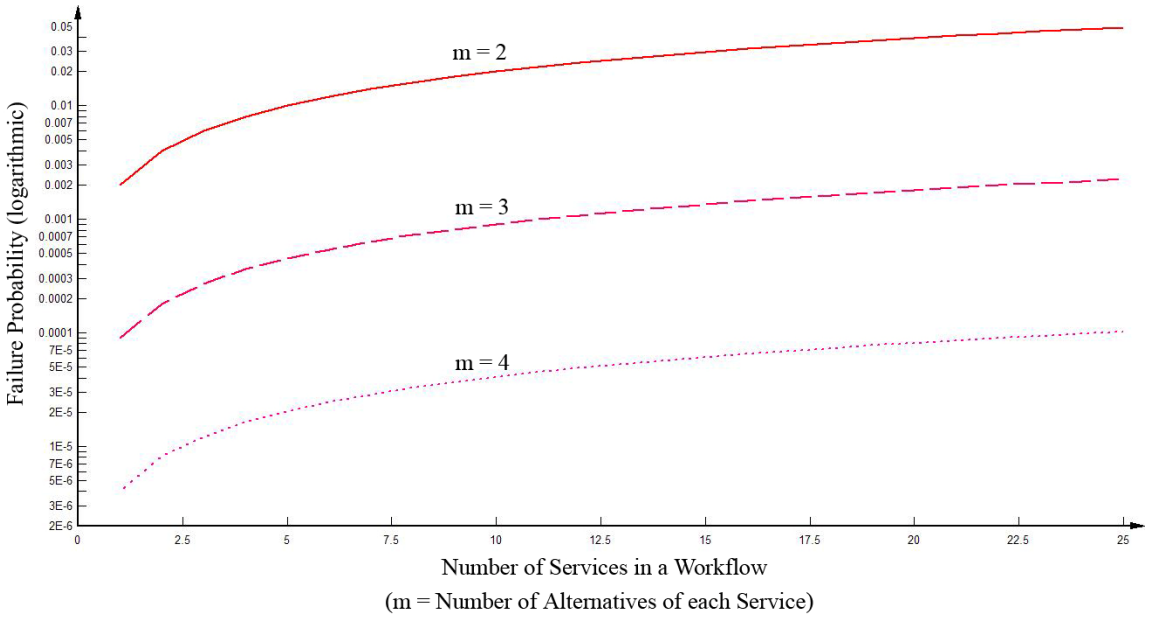


Figure 5.2.2 Failure probability of a series – parallel model representing PIW (p = 0.045)

In comparing the 2 models, we find the potential for a huge improvement using redundancy. Of course, the caveat is that redundant services must not have significant correlated failures (either due to their location, or for algorithmic or other reasons). For example, with p=0.045, n=3 and m=3, workflow failure probability is reduced from 0.129

to less than 10^{-3}. Figure 5.2.3 compares failure probability of a workflow with no fault tolerant support, and a workflow with redundancy-based service-level fault tolerance, where each service has only 1 backup service. Notice that, even with only 1 backup service, the reliability increases dramatically.

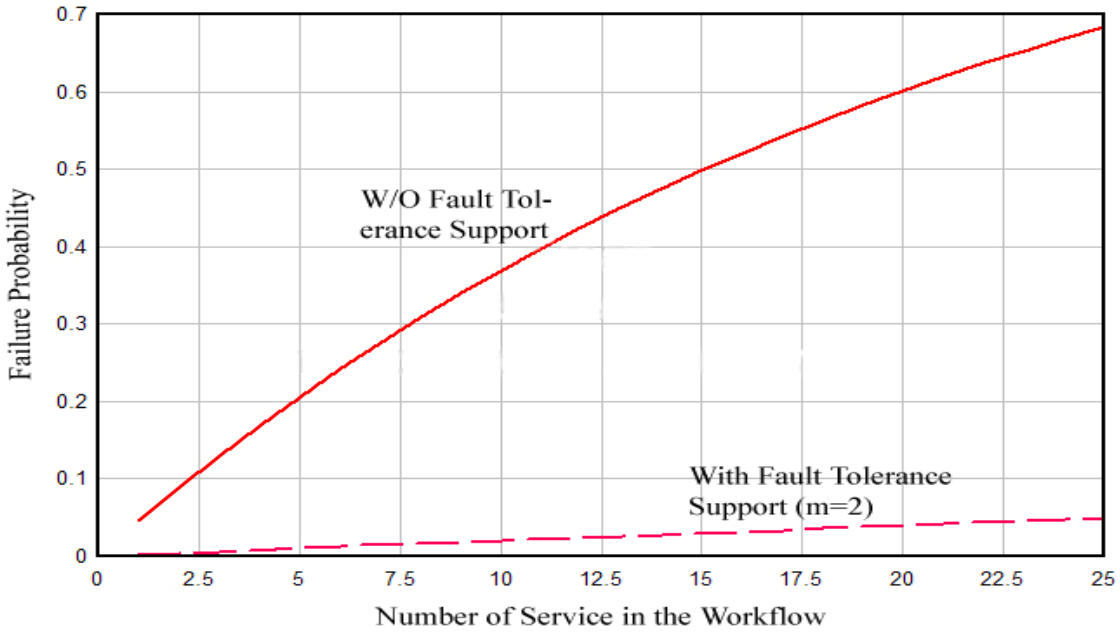


Figure 5.2.3 Comparison between the Fault Tolerant and the Non Fault Tolerant approaches

5.3 Pilot Solutions

In this subsection we discuss the different methods used to implement the model in our pilot, and their advantages. As mentioned above, the current SDM workflows and services do not support fault tolerance when it comes to handling services or server errors. So if the server running the services ever crashes or becomes unavailable, the

whole workflow would fail, and the only way to get back online is to contact the server administrators and have them manually restart the servers or correct the problem.

The following subsections discuss some solutions to the problem that are transparent to the end user. They provide fault tolerance to a certain extent. They are described in the order they were implemented.

5.3.1 Solution 1: Hard-Coding

To achieve Fault Tolerance, there should be at least one backup server running identical copies of the services or there should be another implementation of the services as discussed in section 3.5. Also, the workflow should be able to switch to that server/service in the case the primary server/service goes down.

The most obvious method to implement that was to simply encode the extra location (URL) within the code of the actors, i.e. hard code the location of the servers. This way, by using a simple control structure, the actor will try to invoke the backup services in case the primary ones failed to respond. An example of that approach is shown in figure 5.2.1 where the actor would first try to invoke the method “getDate” on “server1.csc.ncsu.edu”. In case of success, it returns the result and continues executing the rest of the code. But if it fails, instead of simply throwing an exception and exiting, thus terminating the whole workflow, the actor will try to invoke the same “getDate” method, but this time it will try the backup server, which in the examples case is “server2.csc.ncsu.edu”. If it succeeds, it will continue the rest of the code, and if it fails, it

will terminate, unless there was a loop implemented in which the actor would go back and try the first service and so on and so forth until one is successful or until the maximum number of allowed retries is exhausted.

We should note that the mechanism of the method just mentioned is transparent to the user. The only impact that it might have on the performance is a small delay, because the service would have to wait on the first server to timeout before it tries the second one, and the timeout usually takes a few seconds.

```
Service service = new Service();
Call call1 = (Call) service.createCall();
call1.setTargetEndpointAddress("http://server1.csc.ncsu.edu/axis/services/Date");
call2.setTargetEndpointAddress("http://server2.csc.ncsu.edu/axis/services/Date");
QName qn= new QName("getDate");
call1.setOperationName(qn);
call2.setOperationName(qn);
try {
        date = (String) call1.invoke(new Object[]{});
} catch (RemoteException e) {
        try {
        date = (String) call2.invoke(new Object[]{});
        } catch (RemoteException e) {
                e.printStackTrace();
        }
}
```

Figure 5.3.1 Hard-Coding Example

The advantage of using this method is that it provides a basic notion of Fault Tolerance. It makes the workflow more resilient with respect to simple server failures. But there are also several disadvantages of using this method. The most significant one is when the location of the services changes, or we would like to add more backup servers. In that case, we would need to change the code and recompile the actors and redistribute them. This complicates things. We could provide an interface for the end user to input the location of the primary and alternative services. That also is not very efficient because it requires that the end user know the location of different servers. Thus, the need for a more versatile solution emerges. The following two solutions address that.

5.3.2 Solution 2: Using an XML File

This method tries to address the limitations of the first method. Instead of simply hard coding the location of different services, it uses an XML file to store that information. An example xml file is show in figure 5.2.2.

```
<?xml version="1.0" encoding="iso-8859-1"?>

<services>
        <service-name = ="Genbank">
                <bindings>
                        <binding-name = ="main">
                                <url>http://manticore.csc.ncsu.edu/</url>
                                <namespace>urn:spa.service.Genbank</namespace>
                                <method>service</method>
                        </binding-name>

                        <binding-name = =”backup1">
                                <url> http://backup1.csc.ncsu.edu/</url>
                                <namespace> urn:spa.service.Genbank</namespace>
                                <method>service</method>
                        </binding-name>

                        <binding-name = =”backup2">
                                <url> http://backup2.csc.ncsu.edu/</url>
                                <namespace> urn:spa.service.Genbank</namespace>
                                <method>service</method>
                        </binding-name>
                </bindings>
        </service>
</services>
```

Figure 5.3.2 Sample XML file to Store the location of alternative services

This approach stores all relevant information about the services in an XML file. That XML file is then stored on a separate server to be accessed by the actors. The actor would simply parse that XML file, and store the results in an internal data structure. When the

time comes to invoke a service, system would retrieve the relevant information and invoke that service at the primary location. In case this succeeds, system will continue executing the rest of the workflow. If the invocation fails, system will try an alternative service from the list.

This approach presents a more versatile solution than the previous one since it doesn't require that the location of the services to be hard coded, and thus allows adding and modifying the location of the services without modifying the source code. The user would still need to know the location of the XML file, and which service is to be invoked. Those can be passed as actor parameters.

5.3.3 Solution 3: Using a UDDI based Repository

This approach extends the solution presented above. Instead of relying on an XML file to find alternative services, system uses a more dynamic approach by relying on a UDDI [UDDI05] based repository. Using this method, the end user needs only to supply the name of the service he is trying to invoke, and the actor would search the repository to find matching services. Note that there is an underlying assumption that a proper set of matchable keywords exists for all registered services. That service is then invoked. If this is successful, the actor will continue its execution. If invocation fails, another search will occur to find an alternative service.

The advantage of using this method over the previous one is that the repository can regularly check whether the services are still online by using a heartbeat. This feature

isn't supported in the previous method, i.e. in the case one of the services isn't available anymore, the XML file has to be manually corrected, and otherwise it will keep on returning the location of the unavailable service.

Another advantage is the flexibility in adding or modifying existing service. In order to add an additional service, all that needs to be done is to add the service to the repository through an easy-to-use web interface (figure 5.3.3). Thus no configuration files need to be modified.

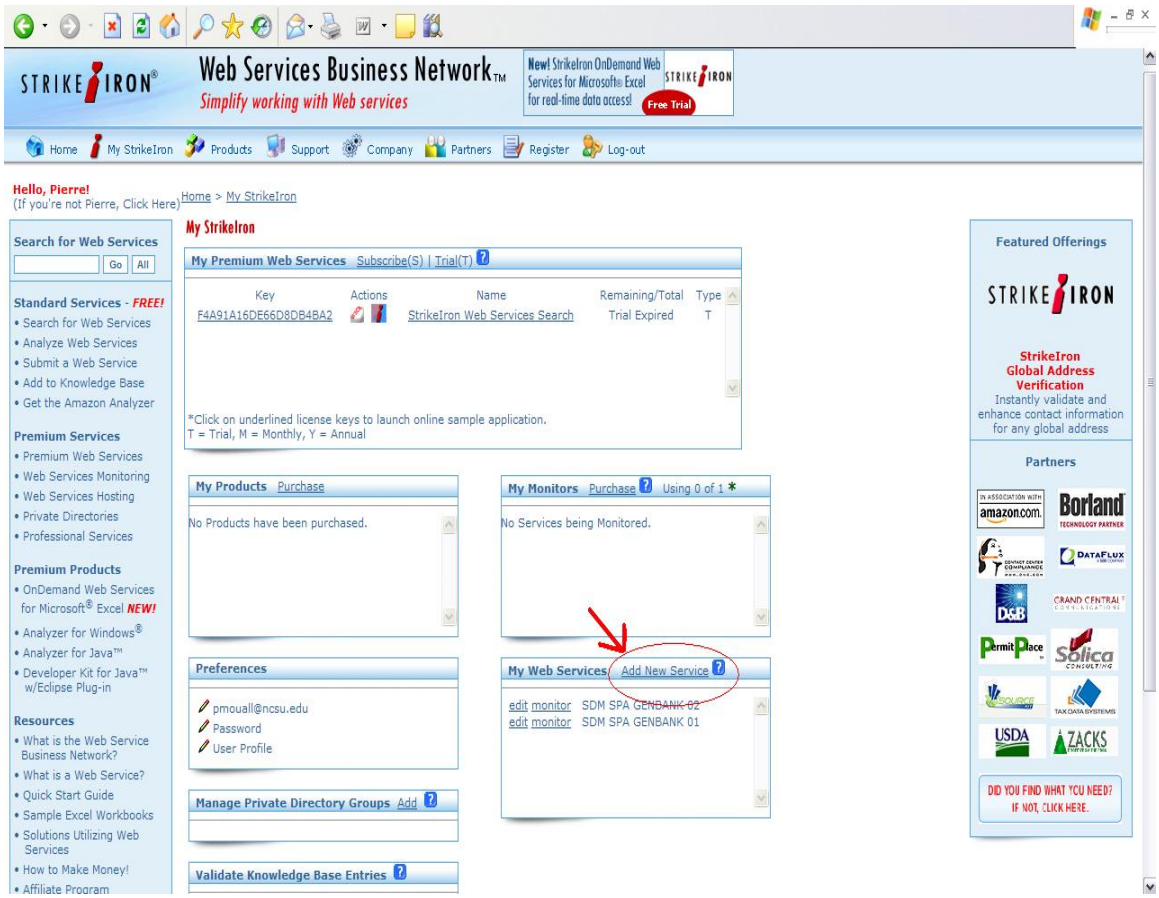


Figure 5.3.3 Screenshot of Strikeiron's UDDI Repository Management Page [SI04]

To actually implement and assess a pilot for this approach we decided to use commercially available StrikeIron's Web Service Business Directory [SI04] as our repository instead of using one of the more generic implementations of UDDI such as jUDDI [jUDDI05] or Soap UDDI [SUDDI05]. The reason behind that decision was to keep things simple, and not get into the messy details of the various UDDI implementations. As part of evaluation collaboration, StrikeIron provided a free access to their easy-to-use interface for adding services to their directory (figure 5.3.3). The readily available StrikeIron solution also provides an API, and instructions on how to search that directory.

5.4 Results and Limitations

The proposed model provides us with a certain degree of fault-tolerance. The system is now more robust when it comes to a failure in a requested web service. It can handle a failure without affecting the whole workflow operation, and with minimum impact on the end user.

However, the proposed solutions in section 5.3 have some limitations. The most important one is that they do not deal with the case when the web service is up but not behaving properly, for example not returning correct results, or taking a long time to process requests. Our solutions will only be able to handle errors caused only by the unavailability of the services.

Further improvement would include testing the validation of the results before proceeding with the rest of the workflow. That can be done, for example, by submitting a sample request and comparing the result with a saved result before submitting the rest of the results. That approach might delay the workflow execution, but it may stave a major correctness failure. On the other hand, we still need to look into finding the best way to provide run-time validation without having a big effect on the whole workflow execution. This is part of another study.

5.5 Implementation Details

5.5.1 Web service actor

The Web Service Actor's function is to invoke web services. Figure 5.5.1 shows an example of a Web Service Actor as it exists in SPA now [SPA05]. This actor invokes a service that queries the Genbank [GenBank04] database. As shown, the actor takes 6 parameters:

- NameSpace: specifies the namespace of the web service
- LocationUrl: specifies the Url where the web service is found
- Username: specifies the username required to access the service
- Password: specifies the password required to access the service
- Method: specifies which method to invoke
- ParameterName: specifies the input passed to the web service

Figure 5.5.1: Parameters of Web Service Actor

As shown, the user has to know and explicitly enter the location and namespace of the service that needs to be invoked. The service will then be invoked with the given username, password, method and input. After it is done processing, the results are forwarded to the next actor in the workflow.

5.5.2 Fault-tolerant web service actor

In order to provide Fault Tolerance, the actor responsible for invoking a service needed to be modified so that it would know how to handle a missing service. As we already described in section 5.2.4, we used a UDDI repository to store the location of the various services, and backup copies of those services. Below is an example of the way services were stored in the repository:

SDM SPA Genbank 01
http://manticore.csc.ncsu.edu:8080/axis/services/urn:spa.service.Genbank?wsdl
Description: Main Genbank Service

SDM SPA Genbank 02
http://sdm2.csc.ncsu.edu:8080/axis/services/urn:spa.service.Genbank?wsdl
Description: Identical Backup Genbank Service

Now, instead of having the user specify the location and namespace of the service to be invoked, all the user needs to do is to specify the search keyword. The actor then searches the StrikeIron repository, retrieves the matching services, and stores the service locations in a local data structure (to avoid performing another search in the case the first service on the list fails). The actor then invokes the first service on its list. If successful, it passes the results to the next actor in the workflow. If it is not successful, it then tries the second service on the list, and so on. If complete failure occurs, the actor sends an error message to the user.

Figure 5.5.2 shows a screenshot of the new parameters required for the "Fault Tolerant Web Service Actor". As shown, only a keyword, method name, username and password are required. The actor will automatically extract the namespace and location URL from the retrieved services.

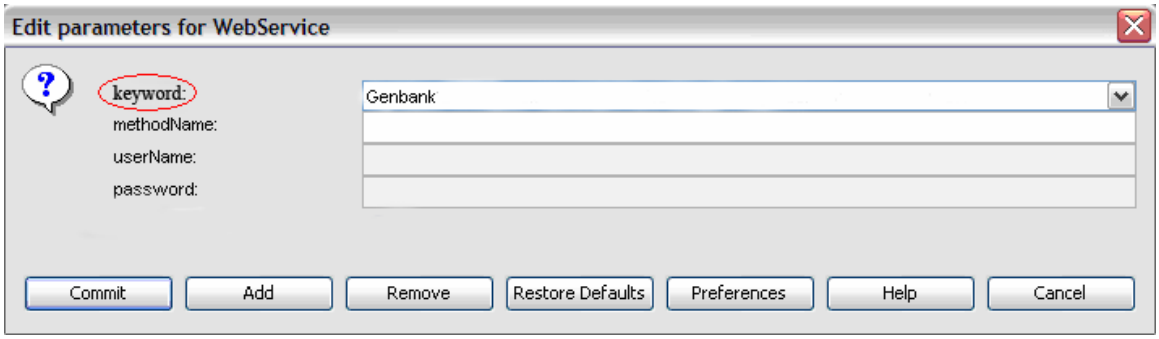


Figure 5.5.2: Parameters of Fault Tolerant Web Service Actor

We also consider the case where one of the services returned in the search isn't what we're looking for. This can be avoided by providing complex search phrase, for example in the case of the Genbank service mentioned above, the user could enter "SDM SPA Genbank" instead of "Genbank". This would solve that problem. Another approach is to

have a sample input and output, then invoke the service with that sample input and compare the outputs, to make sure that the service in question is indeed what we're looking for. This approach involves more computation, and thus might cause a slight delay. But it can also be looked at as a way to provide validation, since in the case the outputs didn't match; we can consider that the service in question is either an incorrect one, or simply not behaving properly.

6. Conclusion and Future work

A workflow management system (WFMS) is a system that allows a user to create workflows using network-based resources.

This thesis presents a solution to one of the problems facing WFMS when they are used in scientific research. Scientific research workflows tend to require frequent changes, and tend to be operated by researchers who may not wish to spend a lot of time working on general information technology issues. In this work, we are concerned with reliability of automated workflows that may use web or similar remote services. Several approaches were presented for how to tolerate failures of individual services using redundancy. It is shown that, provided frequency of correlated failures of functionally equivalent services is low, one to two alternates may provide very good protection of the workflow from individual service failures.. The more versatile solution requires use of a UDDI based repository to store the location of primary and alternate services. The SPA/Kepler framework was used to construct a proof-of-concept solution.

A number of failure related issues remain open. They include workflow validation, reliability and fault tolerance, estimation, and efficiency and degree of protection needed.

Validation is a very important issue: how can we make sure that a service, even if it is available, is responding with correct and accurate results. Current solution does not deal with that problem, and more research needs to be done in that field. One of the solutions that might resolve that issue is comparing output of a validation sample input to a pre-

computed output. Another approach is invoking several identical services and comparing their results, then choosing the most common one, similar to the N-version programming approach described in section 3.4.2. Both methods present a solution to the Validation issue, but might result in additional processing time, thus delaying the whole workflow execution. They are also not comprehensive. This issue requires more research. In this context an issue that will need further work is handling of correlated failures. This requires a much more complex model. Software rejuvenation [Bernstein04] may be a solution that can be used to provide further fault tolerance and reliability, but it's beyond the scope of this thesis.

References

[Aalst00] van der Aalst, W.M.P, ter Hofstede, A.H.M. “Advanced Workflow Patterns”, 7th International Conference on Cooperative Information Systems, volume 1901 of Lecture Notes in Computer Science, p 18. Springer-Verlag, Berlin, 2000.

[Allen01] Rob Allen, “Workflow: An Introduction”, Open Image Systems Inc., 2001.

[Altintas03] Ilkay Altintas, Sangeeta Bhagwanani, David Buttler, Sandeep Chandra, Zhengang Cheng, Matthew Coleman, Terence Critchlow, Amarnath Gupta, Wei Han, Ling Liu, Bertram Ludäscher, Calton Pu, Reagan Moore, Arie Shoshani, Mladen A. Vouk, “A Modeling and Execution Environment for Distributed Scientific Workflows”, 15th International Conference on Scientific and Statistical Database Management, 2003, p. 247, Cambridge, MA.
http://kbis.sdsc.edu/SciDAC-SDM/p1-ssdbm-demo.pdf

[Andrews03] Tony Andrews, et al. “Business Process Execution Language for Web Services Version 1.1”, IBM, BEA, Microsoft 2003
http://www-128.ibm.com/developerworks/library/ws-bpel/

[Arora93] Anish, Arora, Mohamed, Gouda. “Closure and Convergence: A Foundation of Fault-Tolerant Computing”. IEEE Trans. Software Eng. 19(11): 1015-1027, 1993.

[Arora98] Arora Anish, Sandeep S. Kulkarni. “Detectors and Correctors: A Theory of Fault-Tolerance Components”. ICDCS 1998: 436-443, 1998.

[Aviz85] A. Avizienis, “The N-Version Approach to Fault-Tolerant Systems,” IEEE Trans. Software Engineering, Vol. SE- 1 1, No. 12, Dec. 1985, pp. 1,491-1,501.

[Bernstein04] Lawrence Bernstein, Chandra M. R. Kintala “Software Rejuvenation and Self-healing“. 2004
http://www.softwaretechnews.com/stn7-2/rejuvination.html

[Bha05] Sangeeta Bhagwanani, “An Evaluation of Visual Interfaces of Scientific Workflow Management Systems”, M.S. Thesis, NC State University, 2005.

[BPEL4WS] “Business Process Execution Language for Web Services”, 2004.
http://dev2dev.bea.com/technologies/webservices/BPEL4WS.jsp

[Chandra02] Sandeep Chandra, “Service-Based Support for Scientific Workflows”, M.S. Thesis, NC State University, 2002.

[Chi02] Chi, C., Wu, Y., “An XML-Based Data Integrity Service Model for Web Intermediaries”. in 7th International Workshop on Web Content Caching and Distribution (WCW). Boulder, Colorado. August 14-16, 2002.
http://2002.iwcw.org/papers/18500021.pdf

[Col04] Daniel Colonnese, “Grid Service Data for Reliability in Scientific Workflow Systems”, M.S. Thesis, NC State University, 2004.

[Comp01] “Web Servers Comparison”. 2001
http://www.acme.com/software/thttpd/benchmarks.html

[DOE04] Department of Energy, Office of Science, “Data Management Report”. May 2004.
http://ultralight.caltech.edu/gaeweb/portal/misc/2005/05DMW/Final-report.pdf

[Domino04] “IBM Domino Lotus Workflow”, 2004.
http://www.lotus.com/products/domworkflow.nsf

[Dusseau01] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau. “Fail-stutter fault tolerance.” Proceedings of the Eighth Workshop on Hot Topics in Operating Systems, page 33. IEEE Computer Society, 2001.

[GenBank04] NCBI GenBank, 2004.
http://www.ncbi.nlm.nih.gov/Genbank/index.html

[IBM01] “Web services -- the Web's next revolution”, IBM, 1997.
http://www-105.ibm.com/developerworks/education.nsf/webservices-onlinecourse-bytitle/BA84142372686CFB862569A400601C18?OpenDocument

[Jag95] Ajeeth Jagannath. “Impact of Hardware and Software Faults on ARQ Schemes – An Experimental Study”. IEEE Reliability and Maintainability Symposium, 1995. Proceedings. January 1995.

[JSR03] Jeffery Liu, Yen Lu, “Build interoperable Web services with JSR-109”, IBM 2003
http://www-106.ibm.com/developerworks/webservices/library/ws-jsrart/

[JUDDI05] “Apache Web Services Project: jUDDI”.2005
http://ws.apache.org/juddi/

[Kepler04] Kepler Project Website, 2005.
http://kepler-project.org/Wiki.jsp?page=KeplerProject

[Lamport82] L. Lamport, R. Shostak, and M. Pease. “The Byzantine generals problem. ACM Transactions on Programming Languages and Systems”, 4(3):382–401, July 1982.

[Laprie90] Jean-Claude, Laprie, Christian, Beounes, “Definition and Analysis of Hardware- and Software-Fault-Tolerant Architectures”, IEEE Computer Society Press, Volume 23, Issue 7, Pages: 39 – 51, July 1990.

[Liang03] Deron Liang, Chen-Liang Fang. “FT-SOAP: A Fault Tolerant Web Service”. 2003
www.iis.sinica.edu.tw/LIB/TechReport/tr2003/tr03011.pdf

[M&M03] Men&Mice “Domain Health Survey for .COM”. February 2003
http://www.menandmice.com/6000/61_recent_survey.html

[Mus87] J.D. Musa, A. Iannino, and K. Okumoto, *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, New York, 1987.

[Mus90] J.D. Musa, and W.W. Everett, "Software-Reliability Engineering: Technology for the 1990s," IEEE Software, Vol. 7, pp. 36-43, November 1990

[Mus93] J.D. Musa, "Operational profiles in Software-Reliability Engineering," IEEE Software, Vol. 10 (2), pp. 14-32, March 1993.

[Mus98] J.D. Musa, *Software Reliability Engineering*, McGraw-Hill, New York, 1998.

[OASIS05] “OASIS Web Services Business Process Execution Language”, 2005.
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

[Ptolemy04] Ptolemy II Website, 2004.
http://ptolemy.eecs.berkeley.edu/ptolemyII/

[Randell87] B. Randell, “Design-Fault Tolerance,” in The Evolution of Fault-Tolerant Computing, Springer-Verlag, Vienna, 1987, pp. 251-270.

[RDF98] Eric Miller, “An Introduction to the Resource Description Framework”, D-Lib Magazine, May 1998.

[Sch83] R. Schlichting and F. Schneider. “Fail-stop processors: An approach to designing fault-tolerant computing systems.” ACM Transactions on Computing Systems, 1(3):222–238, 1983

[SciDAC04] The SciDAC Program Website, 2004.
http://www.scidac.org/

[SI04] “StrikeIron Web Services Business Directory”, StrikeIron Inc. 2004
http://www.strikeiron.com/htmls/quick_guide.aspx#quit_2

[Singh96] Munindar Singh and Mladen A. Vouk, “Scientific Workflows: Scientific Computing Meets Transactional Workflow”, NSF Workshop on Workflow and Process Automation in Information Systems: State of the Art and Future Directions, 1996, Athens, GA.

[SOAP04] Box, Don, et al. "Simple Object Access Protocol (SOAP) 1.1", *W3C* 08 May 2000. 3 October 2004
http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[SPA05] The Ptolemy II based SPA Website, 2005.
http://www-casc.llnl.gov/sdm/

[SUDDI05] “Soap UDDI”, 2005.
http://soapuddi.sourceforge.net/

[Tamir00] Yuval Tamir, Navid Aghdaie. “Client Transparent Fault Tolerant Web Service”. UCLA Computer Science Department, Los Angeles CA, 2000.

[UDDI05] “OASIS UDDI “, OASIS Open 2005
http://www.uddi.org

[Vouk96] McAllister D.F. and Vouk M.A., "Software Fault-Tolerance Engineering," Chapter 14 in Handbook of Software Reliability Engineering, McGraw Hill, pp. 567-614, January 1996.

[W3C04] W3C Working Group, “Web Services Architecture”, W3C 2004.
http://www.w3.org/TR/2004/NOTE-ws-arch-20040211

[WebShere05] “Common Component Architecture Forum, WebSphere MQ Workflow”, 2005.
http://www-306.ibm.com/software/integration/wmqwf/

[WFMC04] “The Workflow Management Coalition”, 2004
http://www.wfmc.com

[WSFL05] “Web Services Flow Language (WSFL 1.0)”, 2005.
http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

[WSCDL] “Web Services Choreography Description Language”, W3C Working Draft 17 December 2004, W3C 2004.
http://www.w3.org/TR/wsdl

[WSDL01] W3C Working Group, “Web Services Description Language (WSDL) 1.1”, W3C Note 15 March 2001
http://www.w3.org/TR/wsdl

[Xlang05] “XLANG CoverPages Technology Reports”, 2005.
http://xml.coverpages.org/xlang.html

Appendix

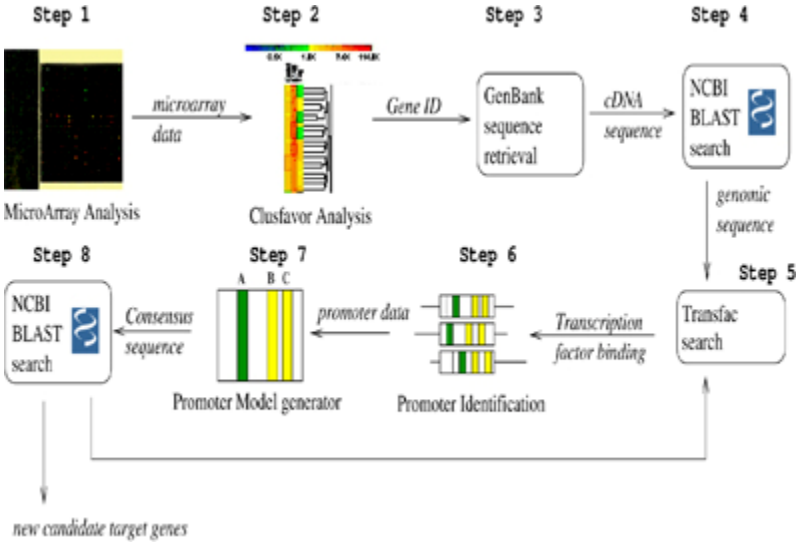
Promoter Identification Workflow

(Composed by Xiaowen Xin, Lawrence Livermore National Labs: https://www-casc.llnl.gov/sdm/documentation/casestudy-piw.php)

Background

Biologists are currently researching how an organism responds to certain environmental changes, by figuring out the effect of that change on gene expression. As an example, suppose a biologist wishes to discover the set of known genes whose expression level dramatically increases in response to radiation. One way to proceed is to first discover the effect that radiation has on the expression of a relatively small set of known genes. From that set, she can find a subset that exhibits the phenotype she's looking for (i.e. whose expression level increases significantly.) Using this subset, she can build a broader picture by finding genes, *not* in the starting set, that have similar promoter regions to genes in the subset. With the assumption that proteins that bind to promoter regions to induce transcription might bind to a set of similar promoter regions on different genes, she can hypothesize that the resulting set of genes will exhibit the same phenotype--their expression level will increase after exposure to radiation.
DNA microarray technology is used to find the expression level of a set of genes. A DNA microarray is basically a glass microscope slide printed with the sequences of thousands of genes each on a different spot in a grid. First, a sample of DNA is exposed to an environmental change, causing the transcription of certain genes. Then, the result is labelled with a fluorescent dye. When the sample is hybridized with the microarray, the amount of fluorescense at each spot in the grid indicates the abundance of sequences in the sample that match the corresponding DNA sequence. Thus, a highly fluorescent spot indicates a highly expressed gene. After a biologist finds the subset of genes that exhibits the phenotype she's looking for, she can run the Promoter Identification Workflow to find similar genes.

Overview of the Promoter Identification Workflow



The Promoter Identification Workflow

The Promoter Identification Workflow (PIW) is designed to output a set of genes that are expected to have similar expression level characteristics as each of a list of input genes. PIW takes as input from the biologist a list of Gene IDs. For each Gene ID, it finds genes with similar promoter regions. It then finds the transcription factor binding sites within each promoter region. From this data, it generates the set of promoter modules for each gene. Finally, it searches in a database for other genes that contain similar sequences as that of the promoter modules.

Finding Similar Genes
Given a Gene ID, the first step is to find the actual DNA sequence associated with it. The National Center for Biotechnology Information in the US has an online database that allows users to search for a gene sequence given a Gene ID. For PIW, we use a Web Service exported by the DNA Data Bank of Japan because it's a standard interface designed to be easily programmatically accessible.

After retrieving the actual DNA sequence, the workflow extracts the first 300 base pairs of the sequence. This is because that first portion of the gene generally contains a promoter region. PIW then executes a BLAST search using the extracted segment to find similar sequences. For this, PIW also queries a Web Service. The BLAST result is a set of similar sequences, where each sequence is represented as a Gene ID, and the start and end positions in the gene sequence that matched the input sequence.

Finding the Transcription Factor Binding Sites
Once the set of similar genes is at hand, PIW selects the two best matches, and discards the rest. For each of these results, it extracts the matching sequence including 1500 additional base pairs on the negative side and 300 additional base pairs on the positive side of it. Then PIW runs a Transfac search on that sequence to find transcription factor binding sites within it. After it has followed this procedure for both genes returned by a

single BLAST search, it has the set of likely transcription factor binding sites for the input gene.

Finding Promoter Modules

A promoter module is a series of two or more transcription factor binding sites that act synergistically or antagonistically. The presence of a single transcription factor binding site may not be sufficient to predict the expression level of the gene because its effect may be enhanced or reduced by another transcription factor binding site nearby. Using information from previous steps, PIW finds transcription factor binding sites that are close together to form a promoter module.

Finding New Genes

As a last step in the PIW workflow, it creates a sequence out of the promoter modules, which may have gaps in between, and submits this information to BLAST, which returns other genes that contain similar sequences.

As a result of this sequence of steps, PIW now outputs a set of genes it has found that have similar promoter models for each input gene. These are good candidates for genes that might have the same phenotype as the input genes because they are probably transcribed by similar transcription proteins.