

ABSTRACT

BHAGWANANI, SANGEETA. An Evaluation of End-User Interfaces of Scientific Workflow Management Systems (Under the direction of Dr. Mladen Vouk).

Scientific research is often exploratory in nature. As technology advances, more steps of such work may become automated. Data used and produced in such experiments becomes increasingly complex and heterogeneous. In such scenarios, a need arises for a tool (or a set of tools) that allows dynamic modeling and (semi) automatic construction and integration of problem solving flows (scientific *workflows*) using various, often network-based, components.

Various Workflow Management Systems (WFMSs) have been developed that allow modeling, construction, and execution of business-oriented workflows. Business workflows are, however, different from scientific WFMSs. They often are less dynamic and evolving in nature. Scientific workflows tend to change more frequently and may involve very voluminous data-translations. In addition, while business workflows tend to be constructed by professional software and business flow engineers, scientific workflows are often constructed by scientists themselves – experts in their domains, but not necessarily experts in information technology, the software or networking domains in which the tools and workflows operate. Therefore, the two cases may require considerably different interfaces and end-user robustness both during the construction stage of the workflows and during their execution.

SciDAC's Scientific Data Management (SDM) initiative is developing a framework that allows scientists to manage data in a more efficient manner, using tools that help them create and manage scientific workflows that use network-based (web) services. This work is part of that effort and it focuses on a comparative analysis of user interfaces of the SDM framework with some other available solutions. An approach for evaluating SWFMS end-user interfacing is presented. More than 20 criteria, based on HCI literature, and interviews, and the findings from the prototype SWMS developed as part of SDM research, are presented. The criteria are used to compare and discuss interfaces of the current version of the SDM SWFMS – the Scientific Process Automation system based on Ptolemy II framework, with some other open-source systems intended for high-end scientific workflow support, such as SCIRun, Enhydra JaWE, and Taverna.

The study finds that although a lot of effort has been put into creating user-friendly workflow systems, the complexity of the visual component is still overwhelming and unintuitive at times – even to computer science students. Many systems that we discuss here employ interesting metaphors, but as the complexity of the underlying layers increases, most visual components also become more complex and more difficult to use. Special care should be taken in presenting new or enhanced functionality to the user for the system to be well accepted. We also observe that most scientific workflow systems that may be used in the life sciences domain do not support satisfactorily decision-based construction and execution of workflows. At this time, most of these systems have difficulty in interoperating with other supporting systems. Verification, Validation and Fault-Tolerance also appear to be a challenge.

**AN EVALUATION OF END-USER INTERFACES OF SCIENTIFIC
WORKFLOW MANAGEMENT SYSTEMS**

By

SANGEETA BHAGWANANI

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. Christopher G. Healey

Dr. Peter Wurman

Dr. Mladen A. Vouk

(Chair of Advisory Committee)

BIOGRAPHY

Sangeeta Bhagwanani was born in Muscat, Oman. She completed her education in India, and received her Bachelor's degree in Computer Science from M. S. University, India. She joined the Master's program in Computer Science at NC State University in Fall 2001, and joined the SDM group in Fall 2002.

ACKNOWLEDGEMENTS

I express my sincere gratitude towards the Chair of the Advisory Committee of my MS Thesis, Dr. Mladen Vouk, for his guidance, support and encouragement since I have been a part of the research group. I also thank Dr. Robert St. Amant, Dr. Peter Wurman and Dr. Christopher G. Healey for serving on my thesis committee and guiding me through various parts of it.

Special thanks to my colleagues, past and present, at the SDM group at NC State for their support and help throughout the research and writing of my thesis. Zhengang Cheng has been an excellent person to work with and has always helped me with various different aspects of this project. I also thank my colleagues at San Diego Super Computer Center, Georgia Tech and Lawrence Livermore National Laboratory for their cooperation.

I would like to thank my family, for their constant encouragement and faith in me. Their support means a lot to me. And last but not the least, many thanks to all my friends for their help, support and confidence.

This work has been supported by the DOE SciDAC grant DE-FC02-01ER25484 and IBM SUR Award.

TABLE OF CONTENTS

LIST OF FIGURES viii

LIST OF TABLES ix

1 INTRODUCTION 1

1.1 MOTIVATION 1

- **Data post-processing: 4**
- **Real-time simulation analysis: 4**

1.2 SCOPE 5

1.3 RELATED WORK 6

1.4 ORGANIZATION OF THE THESIS 8

2 ARCHITECTURE AND CONCEPTS 9

2.1 WORKFLOW CHARACTERISTICS 9

2.2 SERVICES 10

2.3 REPOSITORY 11

2.4 GRAPHICAL USER INTERFACES 12

2.5 SCIENTIFIC PROCESS AUTOMATION (SPA) 13

2.5.1 Background 13

3 USER INTERFACE DESIGN PRINCIPLES AND EVALUATION 16

3.1 USABILITY 18

3.1.1 Windows and the WIMP interface [DFAB97] 19

3.1.2 Metaphor [All01] 19

3.1.4 Direct Manipulation 20

3.2 DESIGN APPROACHES 21

3.2.1 User-Centered Design	22
3.2.1 Iterative Design	23
4 EVALUATION HEURISTICS	24
4.1 HEURISTIC EVALUATION PROCESS	26
4.2 USABILITY HEURISTICS AND METRICS	28
4.2.1 Construction and Execution of workflows	28
4.2.2 Match between system and real world [DFAB97, NM94]	29
4.2.3 Ease of Use	29
4.2.4 Flexibility and efficiency of use	30
4.2.5 Consistency and Standards	31
4.2.6 Recognition rather than recall	31
4.2.7 Aesthetic and minimalistic design	32
4.2.8 Employing dialog to resolve key uncertainties	32
4.2.9 Interaction with the end-user and Visibility of System status	32
4.2.10 Allowing efficient direct invocation and termination	33
4.2.11 Interaction with a Central Repository	33
4.2.12 Data-Mapping	34
4.2.13 Decision-based execution	35
4.2.14 Visualization of output data	35
4.2.15 Reusability of Workflows	36
4.2.16 Verification, Validation and Fault-Tolerance	36
4.2.17 Minimizing the cost of poor guesses about action and timing	37
4.2.18 Help and Documentation	37

4.2.19 Security and Protection	37
4.2.20 Web-based GUI	38
4.2.21 Interoperability	38
5 THE PILOT SYSTEM	40
5.1 THE SDMSDWE SYSTEM	40
5.1.1 Using the SDMSWE system	43
5.1.2 The SDMSWE Graphical User Interface	45
5.2 LESSONS LEARNED FROM THE PILOT	47
6 A COMPARATIVE EVALUATION OF SCIENTIFIC WFMS	50
6.1.1 Main features of Ptolemy II	52
6.1.2 The SPA system	53
6.1.3 Some Usability Issues	56
6.1.4 Other evaluations of SPA	58
6.2 AN EVALUATION OF SCIRUN	60
6.2.1 The SCIRun User Interface	61
6.2.2 Modules	62
6.2.3 Pipes	63
6.2.4 Some Usability Issues	64
6.3 EVALUATION OF ENHYDRA JAWE	66
6.3.1 Package Level	66
6.3.2 Process Level	67
6.3.3 Some Usability Issues	69
6.4 EVALUATION OF OBJECTWEB BONITA	71

6.4.1 The Bonita Modeling Component	71
6.4.2 The Bonita Execution Component	72
6.4.3 Some Usability Issues	75
6.5 AN EVALUATION OF TAVERNA	76
6.5.1 The ScufI Workbench	77
6.5.2 Some Usability Issues in Taverna	79
6.6 A COMPARISON	81
6.6.1 Discussion	82
7 CONCLUSION	85
7.1 SUMMARY	85
7.1.1 Contribution	88
7.2 FUTURE WORK	89
REFERENCES	91
APPENDIX	98
I PROMOTER IDENTIFICATION WORKFLOW	98
II TSI WORKFLOW	100

LIST OF FIGURES

FIGURE 2.2.1: PIW WORKFLOWS AND SUB-WORKFLOWS AS IMPLEMENTED IN THE SPA SYSTEM	15
FIGURE 3.1.1: NORMAN'S EXECUTION-EVALUATION MODEL [NOR]	17
FIGURE 5.1.1: ARCHITECTURE OF SDMSWE	41
FIGURE 5.1.2A: SDMSWE SCREENSHOT	45
FIGURE 5.1.2B: EXECUTION IN SDMSWE	47
FIGURE 6.1.1: SCREENSHOT OF A SIGNAL PROCESSING WORKFLOW IN THE PTOLEMY II ENVIRONMENT	53
FIGURE 6.1.2: SCREENSHOT OF THE HIGH LEVEL PIW IN THE SPA ENVIRONMENT	56
FIGURE 6.2.1: THE SCIRUN WINDOW FRAME [SCIRUN04]	61
FIGURE 6.2.2: SCIRUN SAMPLE WORKFLOW [SCIRUN04]	62
FIGURE 6.2.3: ANATOMY OF A SCIRUN MODULE [SCIRUN04]	63
FIGURE 6.3.1: PACKAGE LEVEL IN JAWE [JAWE04]	67
FIGURE 6.3.2: PROCESS LEVEL VIEW IN JAWE [JAWE04]	68
FIGURE 6.4.1: BONITA GRAPHEDITOR TOOL [BONITA04]	72
FIGURE 6.4.2: BONITA ACTIVITY EXECUTION [BONITA04]	73
FIGURE 6.4.3: BONITA WORKLIST EXECUTION [BONITA04]	73
FIGURE 6.4.4: THE BONITA ARCHITECTURE [BONITA04]	74
FIGURE 6.5.1: TAVERNA ARCHITECTURE [TAVERNA04]	76
FIGURE 6.5.2: THE SCUFL WORKBENCH [TAVERNA04]	78

LIST OF TABLES

TABLE 6.6.1: COMPARISON TABLE (Y = YES, S = SOME, N = NO) 81

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 the 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" [SV96, All01].

1.1 Motivation

For example, an NC State astrophysicist may develop a simulation package that needs to run on a highly parallel supercomputer. The output from the simulation needs to be manipulated to produce correct time-step mergers and prepare it for visualization display. This visual output then needs to be studied, parameters may need to be changed, simulations may need to be re-run, new data may need to be stored, and so on. If such a computer is not available locally, the scientist may need to run at a remote site, perform data post-processing either remotely or locally, bring the processed data over to the scientist's site and then interactively visualize and

otherwise analyze the results. The issue that arises (in this particular example) is that a supercomputer simulation can produce as much as 1 Terabyte of data overnight. While the scientist would like to view the results in the morning, today this is not really possible for a variety of reasons (e.g., security at remote site, amount of data, software licenses). On the other hand, bringing the data over to NC State may be an equally laborious task. It may involve constant attention and re-runs of data transfer commands, and so on. A lot of someone's time may be spent on this. Automation of the whole process, from simulation submission to final preparation of the outputs for visualization, is desirable.

Similar workflows, but with other issues, can be found in almost any domain of scientific discovery [DOE04]. There are many commonly occurring scenarios in scientific domains of interest that can be automated using a sufficiently robust and flexible workflow infrastructure. The characteristics of workflows that may benefit from a proper workflow environment include the rate, coupling, and synchrony at which workflow elements (tasks) exchange data. Depending on the application, dataflow rates range from very low (in kilobits per second) to very high intensity (in gigabits per second). There is a range of coupling - judged mostly by the allowable transfer delay per unit of data - from tightly coupled, requiring microsecond delays, to loosely coupled, allowing milliseconds and even seconds of delay. Finally, there are synchronous (or blocking) and asynchronous (or non-blocking) flows. More concrete scenarios where workflows could be applied include:

Genomics data analyses often have dataflows that are of low to medium intensity, relatively loosely coupled, and in many instances are asynchronous. These scenarios are frequently good candidates for execution in highly parallel and distributed environments. Many types of *in silico*

genomics analyses, such as promoter identification, start with an initial set of data, perhaps acquired in a more mechanical way such as through fast sequencing equipment or from a microarray chip. This is followed by an ordered sequence of database queries, data transformations, and complex functional, statistical and other analyses. Such work may require computing power that ranges from a desktop to a supercomputer, but typically interchanges data only in the megabyte to gigabyte range. By defining a workflow to automatically invoke and analyze more routine parts of the process, multiple data sets can be processed in parallel without requiring a significant amount of additional effort from the scientist and can considerably increase productivity [Cha02]. The bioinformatics workflow we have used so far in this project is described in more detail in Appendix I.

Leading-edge simulation analyses typically belong to the high-intensity data flow category – they not only require high performance computers, but produce and exchange data from hundreds of gigabytes to terabytes, and perhaps even petabytes, of data. Because these scenarios are resource intensive, prudent computational planning, steering and validation of the workflows (whether manual or automated) is necessary. At the beginning, flows tend to be tightly coupled and synchronous. However, as the scenario progresses they may well be both asynchronous and more loosely coupled. We distinguish two categories:

- **Data post-processing:**

When a scientific simulation is completed, a sequence of tasks is often executed by the scientist to analyze the resulting data set(s), e.g. run an analysis to check for errors, eliminate corrupted time-steps, identify the interesting subset of the data, visualize the data, generate a movie, and transfer the data set to long-term storage or a data-distribution depot. Of course the details vary but for each scientist the sequence of tasks remains relatively consistent across simulation executions. By providing an easy way of automating, tracking and modifying this sequence of tasks, data post-processing may be more performed more efficiently in our experience as much as an order of magnitude more efficiently [Cha02].

- **Real-time simulation analysis:**

While a simulation is executing, a series of analysis steps can be performed to determine if the run is proceeding as expected or has encountered an erroneous condition (e.g. a tangled mesh, a singularity, etc.). Based on the results of this analysis, a scientist can modify or terminate the execution of the simulation and thus reduce the time spent performing wasted computations. While creating and automating a workflow will not, in itself, enable computational steering, it is an important step. As part of the coming Year 1 and 2 works, we propose to implement the TSI workflow described in Appendix II. It focuses on the large-scale data aggregation, movement and redistribution required to perform advanced visualization of the results of the supernovae simulations. This scenario is representative of high-performance computing simulation and data analysis workflows. Thus, by implementing it within the SPA workflow environment, we demonstrate the applicability of

our environment to a large class of SciDAC relevant workflows. Through support from the Link-Up project [Link04] and the Kepler initiative [Kep04a], SPA will interact with other national and international projects on scientific workflow technology, including SEEK [SEEK04], Ptolemy [Pto04], and BIRN [BIRN04] to monitor and advance the state of the art in scientific workflow management.

1.2 Scope

In such and similar scenarios as above, a scientist typically uses a networked computer connected to several other data, storage and analysis sites. For each task that the scientist performs, data may be accessed from both local and remote locations. Analytical capabilities of other sites may be used to complete smaller parts of the workflow. As workflows grow larger, and implement more sites that perform specialized functions, the end-users may become inundated with information and tools. Since most scientists that we consider here (e.g., scientists in the life sciences) are non-programmers, such an environment usually overwhelms them. This tends to result in a situation where, after the initial – and large – investment in learning a particular computer-based toolset, a scientist may become quite reluctant to try new services even when they may potentially increase productivity. On the other hand, scientists who try to learn new tools, or may not have a choice in using new tools that may help them enhance their problem-solving process, may end up spending an inordinate amount of effort figuring out the technology rather than doing their domain science.

This prompted development of client-centric infrastructures that allow the end-users (here, scientists) to design and execute their scientific domain workflows using tools and services in a

transparent manner. In such a system, it is imperative that the end user interface be quite usable and easy to understand. If the visual interface is too busy, confusing, or difficult to adapt the end user may end up spending too much time figuring out how to use it, or worse, not use it at all regardless of the effectiveness and versatility of the underlying layers.

In this study we focus on the user interface component of such a system and study its characteristics [Joh06]. We propose ways to build a usable interface for a complex system such as a Scientific WFMS [ZOO, WWV, SV96], and evaluate some existing systems available at this time.

1.3 Related Work

Performed in close collaboration with domain scientists, our research [e.g., SV96, Cha02, ABB+03] has identified number of requirements for scientific workflows, and provides DOE with a sound basis for assessment of usability and cost-effectiveness of scientific workflow support. In this study, we discuss a few workflow systems that allow the scientist to create and execute workflows in his domain. In Chapter 6 we discuss SDM's Ptolemy-based SPA system [SPA05], along with some others like SCIRun [SCI04], Enhydra JaWE [JaWE04], etc. We evaluate these systems based on their visual interfaces and overall usability.

The study that comes closest to present work is that by Peter Boersma [Boe94]. Although the author focuses on "traditional" (i.e. business-related) WFMS's, the general approach and goals of the study are the same as ours. Boersma's work focuses on conducting two laboratory

experiments researching the usability and organizational impact of workflow tools. The results of his study show the effects of introducing workflow software into an organization on productivity, consulting, and waiting time.

The work of Ramage in [Ram99] is again very similar to ours, although it exclusively explores the space of CSCW (Computer Supported Cooperative Work) systems, which, as we show later, are very different from scientific WFMS's. Ramage observes that currently existing evaluation methods are mostly inadequate as they cater to single-user systems, and may not be useful while evaluating multi-user systems. He proposes new ways of evaluating CSCW systems taking into account issues of individual, group and organizational effects as well as questions of usability. In [Gru89] Grudin discusses problems with CSCW systems, some of which are applicable to our field of research. He explains what he thinks the problems with the usability of CSCW systems are and what could be done to rectify them.

In [Bak02], Baker evaluates systems falling under the Groupware category. He explains how Heuristic Evaluation can be used to evaluate groupware systems, and how to make the process more efficient by defining heuristics systematically. In [ONG96], Wei ONG studies complexity in graphical user interfaces in general. He conducts two experiments to study the effect of Distinctiveness (of Icons or Widgets) on response time and learnability. He concludes that Distinctiveness has a major effect on speed of search of icons, but may not have an effect on the learning of those icons. He suggests that interface designers balance the speed advantage of globally distinct icons with the representational advantage of locally distinct icons while designing user interfaces.

1.4 Organization of the Thesis

Chapter 2 briefly introduces the SciDAC initiative and the vision of the Scientific Data Management center, and the general architecture and concepts of the SDM Scientific Process Automation system. Chapter 3 discusses relevant design principles for complex system user interfaces, and introduces concepts from interface design in the context of workflow systems. Chapter 4 explains usability heuristics, and defines a set of heuristics for scientific workflow management systems which we later use in Chapter 6 to evaluate existing workflow systems. It also discusses design approaches towards interactive systems design. Chapter 5 illustrates the Pilot system SDMSWE and explains its significance as a prototype that helped us understand the concepts of workflow construction and design. Chapter 6 presents a comparative analysis of several existing scientific WFMS, including the Pilot system and SDM's SPA system. Finally, Chapter 7 discusses results and suggestions for future work.

2 Architecture and Concepts

There is a need for a workflow construction framework and approach that allows a scientist, who is primarily a domain expert, to create and execute complex experimental workflows using heterogeneous data, and distributed storage and analysis resources. Such an architecture should be end-user centric and must have support for the end-user's domain. Following are the main concepts [Cha02, ABB+03].

2.1 Workflow Characteristics

The following list, while not exhaustive, covers principal scientific workflow requirements:

- **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 and automate 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). 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 (e.g., SABUL, or FastTCP, etc.).
- **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.

2.2 Services

A basic assumption is that it is efficient to have scientific tools for analysis and data manipulation distributed across various remote sites (in the spirit of the “grid” concept). It is inefficient to develop well new tools which are not workflow specific. If there is a sharing within a domain, it should be encouraged. 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. Services, as explained in [Cha02], are software applications that interact with other software applications over the network using standard protocols. In general, such interactions tend to be loosely coupled and often asynchronous.

2.3 Repository

It is important that all network-based services be made available to the end user transparently, i.e. the user should not need to access and use services differently based on their location, function or context. Hence, a need arises to shield the end-user from the technical details of the services using a common repository that hosts all such services. Such a repository may be physical, i.e. in the form of a database where all services are stored uniformly, or it may be virtual, such that the implemented system has a common architectural interface that allows seamless access to all such services.

A popular choice as a registry is IBM’s UDDI [UDDI04]. The UDDI (Universal Description, Discovery and Integration) registry is a global, public, online directory that gives organizations a uniform way to describe their services, discover other organization’s services, and understand the methods necessary to conduct information exchange with one another.

UDDI implements a distributed registry across the web where services, businesses and descriptions of both are stored in a common XML format, hence enabling documents to be easily

searched and analyzed. Typically, a client application searches for a service with certain functionality, and once found, the owner of the service mutually agrees with the client to share the service functionality.

Such an architecture enables fault-tolerance. Because a user is allowed to pick and employ services from the registry into his workflow, he also has the ability to use an alternative service with similar functionality from the registry, in case the original service fails. This ensures that no workflow terminates unexpectedly because of failure of one particular service in the flow.

2.4 User Interfaces

The user interface is one of the more important components 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.

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. Section 3 discusses in detail characteristics of a good scientific user interface.

In the next section we introduce the SDM's Scientific workflow system that implements the above-mentioned concepts.

2.5 Scientific Process Automation (SPA)

The SciDAC (Scientific Discovery through Advanced Computing, [Sci04]) 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 of the SciDAC program that aims to develop tools that adapt and extend current technology for scientific workflows.

2.5.1 Background

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 Scientific Data Management 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. Our system is described below.

2.5.2 The SPA Workflow System [SPA05]

The main aim of the SPA project is to develop a Scientific Problem Solving Environment (also referred to as WFMS, and discussed earlier) that allows scientists to create exploratory scientific workflows using an intuitive graphical user interface. The SPA system is based on UC Berkeley's Ptolemy II system [Pto04] which is an independent, open-source java-based system that allows heterogeneous, concurrent modeling, design and execution. By creating functionality over the Ptolemy II system, the SPA system aims to include the features mentioned below:

- An intuitive graphical user interface. The user interface should be easy to understand and use, and the user should 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.
- 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.

- Support for local and distributed service components, possibly attached to a grid network.
- Support for publishing workflows, and exchanging workflows with other scientists, and support for data and workflow provenance.

Below is a snapshot of the system GUI as it exists right now. Specifically, it is a snapshot of the Promoter Identification workflow. We describe the architecture and implementation of the SPA system in detail in Section 6.1, followed by an evaluation of its Usability.

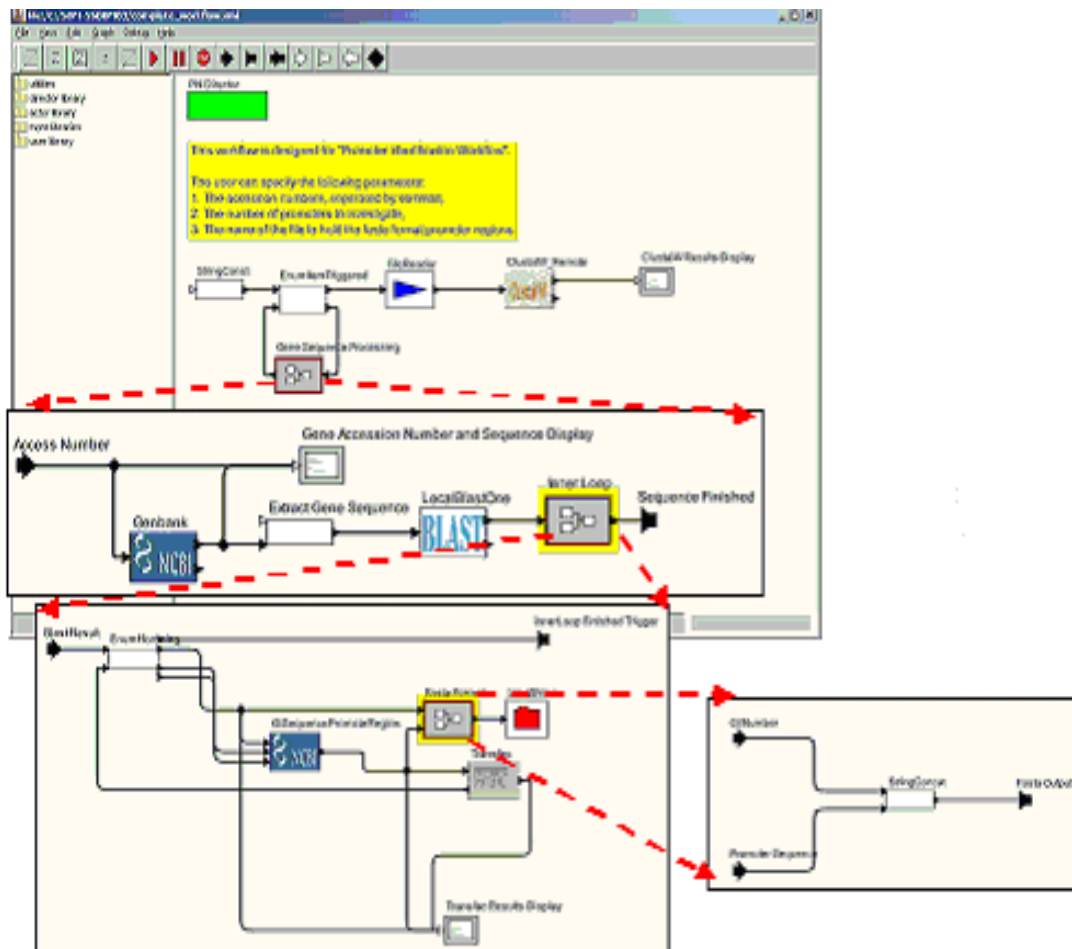


Figure 2.2.1: PIW Workflows and Sub-Workflows as implemented in the SPA system

3 User Interface Design Principles and Evaluation

Graphical user interfaces (GUI) are common today. In fact, they are expected. As already mentioned, in the context of workflow management systems, it is imperative that the user interface (graphical or otherwise) be as non-taxing, useful, and intuitive as possible [DFAB97, Sch86]. Although the underlying functionality of the system is important, it is even more important that this functionality be presented to the user in a simple and easy-to-use manner, and at the level appropriate for the end-user's expertise and domain. The user should not be overwhelmed by the complexity of the system information technology, but should easily adapt and make optimum use of it using the interface.

Humans have a great ability to remember and recall graphical concepts and visual cues. Much better, in fact, than at processing text or using command line interface [DFAB97, SCH86, Zad99]. Designers of graphical user interfaces have long realized this. A wide array of techniques is available. They include various graphical query languages, sounds, animations, virtual reality, etc. Norman's execution-evaluation model [DFAB97, Nor], and the interaction framework that extends, is an example of a successful way of analyzing interactions between a computing system and a human end-user in terms of the difficulty for the user to express what the user wants and determining whether this has been achieved by the GUI. In [Nor90] Norman provides guidelines for designing usable systems, and the rationale behind these guidelines. He explains that an interaction between a human and the system can be divided into two phases -- **Execution phase** in which a user performs actions on the world (a device or system) and the

Evaluation phase in which the user assesses the state of the world, to evaluate the results of his actions. He recommends designers consider the properties of all the system components as well as their interactions (especially with humans). He observes that failures of information systems are attributed to human error rather than to the design, and that designers should design systems keeping the end-user in mind. He encourages designers to look at bad designs and to avoid the mistakes that designers of those systems made. A diagram showing Norman's Execution-Evaluation model is given below. Although we don't explicitly follow Norman's model to guide our design, we follow User-Centered Design (as the evaluation phase) and Iterative design (as the feedback phase) as our choices of design, as will be shown in later sections.

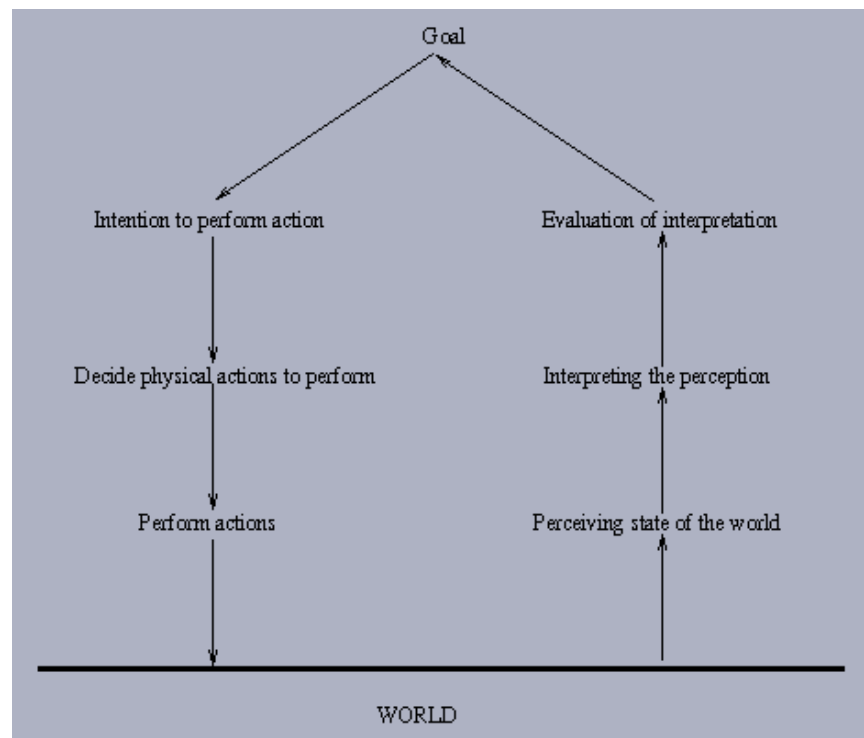


Figure 3.1.1: Norman's Execution-Evaluation Model [Nor]

3.1 Usability

One of the main factors that sets apart one interface from the other is its “usability” [NM94, Usa04, Nie05]. In very broad terms, usability is what makes a good interface effective, efficient, enjoyable, and safe. Usability is also a very strong function of the end-user expertise and training level. Designing for maximum usability is the goal of interactive systems design. However, what is a very usable for an expert may not be very usable for a beginner. A main issue any designer of an interactive user interface needs to deal with is measuring of the usability of such a system. Some of the factors that determine the usability of a system, as described in [DFAB97, Bro03], are:

- The amount of time it takes the user to learn how to use the system properly,
- The ease with which users can carry out tasks to solve their problems using the system,
- The user’s ability to customize and adapt the system to his way of use, than the other way around,
- Speed of performance of carrying out such a problem-solving activity using the system,
- The number of errors that occur while performing the activity, and the user’s ability to avoid errors and recover from them once they occur,
- User’s satisfaction with the system.

Some of the important principles that support usability are:

- Learnability: The ease with which users can use the system efficiently.
- Flexibility: The different ways that the user and system can exchange information.
- Robustness: The level of support provided to the user by the system as he tries to achieve his goals.

More details on this are given in [DFAB97]. We implement these principles in the Heuristics that we propose in Section 4.2.

Some of the principal paradigms for providing usability in WFMS GUIs are “windows”, metaphors, and direct manipulation.

3.1.1 Windows and the WIMP interface [DFAB97]

The advent and widespread use of Apple Macintosh computers and later windows-based IBM compatible PC's, has given rise to one of the more popular design paradigms for interactive computing, it is called WIMP - for windows, icons, menus and pointers. Windows are areas of the screen that behave as if they were independent terminals, and usually contain text and graphics that the user interacts with. Icons are small pictorial representations for closed windows (or rather windows left in the “background”). Icons essentially make it easier for the user to switch to different windows using their closed or “minimized” representations. Pointers help the user point and select things such as icons in a WIMP interface. Menus present choice of operations or services that the system can perform at a given time. A user usually selects the desired option using a pointer.

3.1.2 Metaphor [All01]

Metaphors are used to teach new concepts, computer-related concepts being no exception, to novices in terms of familiar concepts. They help increase and improve initial familiarity between

user and computer application. Some of the more popularly known Metaphors are the Microsoft's "Desktop" metaphor and the "Paper Clip". In the SPA system described in the previous section, Ptolemy implements services as "Actors" and the execution engine as the "Director".

However, problems arise once the initial stages of familiarizing one to the concept has gone by, and the user expects the actual concept to behave very similar to the metaphor being used, often leading to confusion and dissatisfaction. Microsoft's friendly Paper Clip was replaced by other help techniques since both end users and developers never really found the Paper clip very useful, unlike the actual office paper clip, but extremely animated and intrusive. Some designers feel the Desktop metaphor, although helpful at the start, is also inadequate up to an extent. For example, dragging floppy disks to the Recycle Bin may not come across as intuitive way to eject them. More such problems are discussed in [DFAB97].

3.1.4 Direct Manipulation

Direct Manipulation (DM) allows the user to get rapid feedback and evaluative information for every executed user action. Thus, for each action that the user performs, the user knows the new state of the underlying system immediately. The following are the main features of a DM system as described by Ben Schneiderman [Sch86]:

- Visibility of objects of interest
- Incremental action at the interface with rapid feedback on all actions

- Reversibility of all actions, so that users are encouraged to explore without severe penalties
- Syntactic correctness of all actions, so that every user action is a legal operation
- Replacement of complex command languages with actions to manipulate directly the visible objects (and, hence, the name direct manipulation)

3.2 Design Approaches

Some of the more popular approaches towards designing user interfaces are:

- User-Centered Design
- Task Analysis
- Rapid Prototyping

These approaches are usually chained into a sequence called the **Design Cycle**. The Design Cycle usually starts with obtaining user ideas about the proposed system and initial design, followed by requirements specification, and design of the prototype. This is usually followed by evaluation, redesign and reimplementation.

Task Analysis usually describes a wide range of techniques used by ergonomists, designers and users to describe human-human and human-machine interactions in a system. It is usually used as an initial technique in the design cycle process to figure out the goals of the end-user and the system. We include Task Analysis as a part of User-Centered Design.

We don't strictly follow the Design Cycle in our user interface design process because we feel that the techniques described below (User-Centered Design followed by Iterative Design) efficiently achieve the goals of the Design Cycle.

3.2.1 User-Centered Design

User-Centered Design (UCD) [VIR01] is the author's choice of the design approach for a WFMS. UCD is an approach where user requirements are the determinants in the design of a usable system. This requires a clear understanding early on in the design of the tasks that the user wishes to perform. Some guiding principles of UCD are:

- Early focus on, contact with, and understanding of users and their jobs. This ensures that the users are involved in the design process right from the requirement-gathering phase.
- Integrated Design where all aspects of usability evolve in parallel.
- Early and continual testing of prototypes by users and feedback. Qualitative and quantitative measurements determine how closely the implemented system's specifications match the requirements.
- Iterative Design, where the system is modified based on the testing of the prototype by the user and his feedback.

The problem with such an approach is that the tasks that the user will perform will be known to him once he is completely familiar and comfortable with the system. Also, the end-user and developer of the system may communicate using different jargon, and iterative design may result

in fine-tuning of the system to cater to a specific group of user's needs. These and more such problems with UCD are discussed in [DFAB97].

3.2.1 Iterative Design

Iterative Design [Usa04] is defined by the use of prototypes [DFAB97] which can be defined as examinable models of the system that simulate some but not all features of the proposed system.

The three main approaches to prototyping are:

- **Throw-away:** The prototype is used to the point where the exact requirements of the users are determined iteratively through testing, and finally the proposed system is built from scratch using the knowledge gained from the prototype.
- **Incremental:** The overall design for the proposed system is partitioned into independent and smaller components and the final product is released as a series of products, with each subsequent release containing one more component.
- **Evolutionary:** The first version of the prototype is built on iteratively until it finally meets all the requirements takes form of the final product.

4 Evaluation Heuristics

Evaluation of user-interfaces has been a topic of research for many years now. However, most of this work focuses strictly on either single-user systems [DFAB97], mixed-initiative or intelligent systems [HM93, Keo00, Hor99], or in some cases, systems that belong to the CSCW (Computer Supported Collaborative Work) category [Gre98, Gru98, McE02, Bak02]. A Scientific WFMS (SWFMS) as we describe here, is typically a system that a) borrows techniques from other systems belonging to the above-mentioned categories, and b) is intended for use primarily by domain experts who “program” it to model and implement their workflows, but who are not IT experts but their own domain experts (e.g., astrophysicists, bioinformaticians, etc.).

An SWFMS may not be a single-user system. It functions and manages data in a collaborative environment. A user may want to use network-based resources (like web-services), and share workflows (or templates) with other users either synchronously (collaboratively) or asynchronously. Such a system usually does not qualify as an Artificial Intelligence engine because it is not completely intelligent, but it may also not be only a CSCW (Computer Supported Collaborative Work) engine, since the system may not involve synchronous collaboration in the traditional sense. On the other hand for a collaboration to be possible, end-users may need to have access to the same set of tools and functions, and may need to exchange workflows.

System end-user interfaces can be evaluated using techniques borrowed from the human computer interaction (HCI) world, techniques such as heuristic evaluation, user testing, etc, and

from the cognitive and social psychology, such as lab experiments, interviews, focus groups and customer feedback, or from sociology such as ethnography [DFAB97, SCH86].

For the purposes of this study, we will use a variant of a “heuristic evaluation” [NM94, Nie92] approach. As defined in [NM94], “Heuristic evaluation is an evaluation technique for user interfaces that relies on an evaluator’s immediate reactions, intuitions and predictions, that fall under a set of Design Principles and Usability Attributes”. For most part, such an evaluation, we hope, will be formative, i.e. help towards the iterative development of a system (or a prototype) of this kind.

Evaluations can be of two kinds – Analytical or Empirical. While Analytical evaluations focus on how the system would probably function in a given scenario, Empirical evaluations focus on creating a prototype or the system itself and having users use it and provide feedback. Heuristic Evaluation is one of the better Empirical evaluation processes. It has the following advantages:

- It is easy and intuitive to perform,
- It is a good method for finding both major and minor problems in system,
- It is suitable for use in the early development process, and hence is low cost since the problems discovered are much easier to fix than if found later,
- No advance planning is required since evaluators can evaluate the system as a team,
- Since each problem discovered is in reference to an already established usability problem or heuristic, it is fairly easy to fix the problem.

4.1 Heuristic Evaluation Process

While usability-engineering methods can contribute substantially to the resulting interface, many developers find usability methods intimidating, expensive, and too difficult and time consuming to apply. Heuristic evaluation can help in such cases. It is one of the principal discount usability-engineering methods, and yet can give quite good results [NM94]. It employs methods that are cheap, fast, and easy to use [NM94, Nie92].

Heuristic evaluation can be used as part of usability engineering to finding usability problems in a user interface design. Such an evaluation is usually an integral part of an iterative design process. It involves having a small set of evaluators, an expert group, examine the interface and judge its compliance with recognized usability principles. It is a variant of the classical Delphi Method [Delphi].

Heuristic evaluation involves more than one evaluator. Each evaluator inspects the interface alone to find usability problems. After all the evaluations have been completed, the evaluators communicate with each other and aggregate their findings. Typically, a heuristic evaluation session for an individual evaluator lasts one or two hours, though longer evaluation sessions are not uncommon for more complex systems.

During the evaluation session, the evaluator goes through the interface several times and inspects the various elements of the user interface and compares them with a list of recognized usability principles. In addition to the checklist of general heuristics to be considered for all such elements, the evaluator is also allowed to consider any additional usability principles that could

possibly be relevant for any specific element in the given context. In such cases, as will be shown later, it is possible to develop category or domain specific heuristics that apply to a specific class of products as a supplement to the general heuristics.

In most cases, it is upto the evaluators to decide as to how to proceed with evaluating a user interface. A common practice is on where the evaluators go through the interface at least twice where the first pass would be intended to get a feel for the flow of the interaction and the general scope of the system, and the second pass allows the evaluator to focus on specific interface elements and evaluate them.

Although it is recommended that the number of evaluators be more than one for purposes of Heuristic Evaluation, it is not uncommon for Heuristic Evaluation to be carried out by a single expert for smaller projects, or projects where the end user has been providing constant feedback during each design phase. In our case, the author of this study is responsible for the Heuristic Evaluation for the tools mentioned in the Section 6. However, evaluations of these tools by other authors, developers, and end users were taken into consideration while creating the Usability Heuristics and conducting the Heuristic Evaluation itself. For this reason, we refrain from assigning severity ratings to the usability problems discovered, and use a more flexible way to outline the severity of the problem, as will be shown in Table 6.6.1.

4.2 Usability Heuristics and Metrics

Since the start of this project about 2 years ago, we have identified the following to be the basic features that can be expected out of a GUI for a Scientific WFMS. Heuristics such as these

define the desirable properties of a usable interface, ensure that the tool is truly intuitive and user-friendly, and alternatively can be used as metrics based on which an evaluator (or alternatively an interface designer) may assess an existing system. Some of the metrics have been borrowed from the HCI discipline [DFAB97], UI Patterns [Tal98], or Workflow Patterns [WP04, ABHK00], while others have been discovered in the requirements gathering process. While these features are necessary, they are not sufficient. Managers and end-users of the system would ideally compile a complete set of heuristics for such an evaluation during initial design phases with the help of HCI and usability experts [NM94, Nie92, Nie05]. Such a specification of heuristic metrics usually changes through various iterations of the design phases as more feedback is received and specific details of the design are deduced.

Below we present a set of such heuristic metrics for a Scientific WFMS. These are based on heuristics defined by Nielsen [DFAB97, Nm942], and were adapted to fit our description of such a system.

4.2.1 Construction and Execution of workflows

A Scientific WFMS should allow a scientist to chain different tasks or services together to form an analytical flow. The scientist should be able to specify the sequence of steps (some of which may be parallel) that the scientist wants to execute, inputs and outputs, and information about how output data of the executing task(s) maps to the input of the next-in-line task. In general, a scientist should be able to specify equivalent of a full general activity graph [Dav91]. For example, steps may be synchronous or asynchronous, interactive, collaborative, and tasks may be repetitive, and the looping control can be a function of the computation. The scientists should be

able to specify or modify task specific parameters that will make the task behave one way or the other, e.g. specific method to be used within the given web-service, name for the task, etc. The user should be able to “execute” the workflow, and he should be able to stop or pause it temporarily if needed, and to interact with it. He should be allowed to “background” this control flow and create and/or execute a new workflow separately. Thus the end user should be able to “orchestrate” the workflow the way the user wants. A workflow “recording” should be storable and exchangeable in a standard way.

4.2.2 Match between system and real world [DFAB97, NM94]

The system should “speak” the user’s language, with terminology, phrases and concepts familiar to the user, rather than be system-oriented or information technology oriented. It should follow real-world conventions, making information appear in natural and logical order. A **Metaphor** [Tal98] may be employed, to help the user draw natural comparisons to the system being used and the domain world in which the user operates.

4.2.3 Ease of Use

Every system designed for use by actual end-users (as against systems that “talk” to other systems) needs to be easy to use for the intended end-user. It has to be sensitive to the end-user information technology expertise level and domain expertise level. The two may be vastly different. This especially becomes a necessity in cases where a complex system will be used by an end-user who does not want to deal with all the internal details of the system. In our case, the scientist does not need to, and very often does not want to, know the information technology details of the services in use, e.g. what the WSDL looks like, where the service is deployed, etc.

Thus, the system must have an “appliance-like” interface that makes it easy to use in the end-user domain, but hides all the underlying information technology details of the system that the end-user does not need to know or adjust.

Ease of use also implies an uncomplicated way of installing and uninstalling the system. A user should not have to worry about how to install the software package by putting different components together and the package should have self-installation capabilities.

4.2.4 Flexibility and efficiency of use

The user should have full control of the system (although some steps and procedures may be “automated”), and have the freedom to modify the current state of execution as and when he pleases. A complex system of this kind should at all times support undo and redo functionality. Also, Accelerators [DFAB97] should be employed. Accelerators allow users to tailor frequent actions by employing shortcut keys for such actions. These would usually not be used by the novice user, but may often speed up the interaction for the expert user to such an extent that the system can cater to both inexperienced and experienced users.

4.2.5 Consistency and Standards

The user interface of a system as complex as described here should have a consistent look-and-feel, and if it employs any metaphors, they should be implemented consistently as well. Users should not have to wonder whether different words, situations and actions mean the same thing or no.

It is important that standards be implemented wherever applicable. In case of scientific workflows, not only should the user interface comply with industry standards of design and evaluation as discussed earlier, but also the underlying layers should implement a standard language for representation of data. XML has proven to be an efficient standard for this purpose, one that is widely accepted by the industry. A lot of industry standards have been recommended by the WfMC [WfMC04], and a detailed comparison of these standards is given in [ABHK00, WP04].

4.2.6 Recognition rather than recall

All options, objects and possible actions should always be made visible. The user should not have to remember information from one part of the dialog to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

Also, **automation** [DFAB97] to the extent that it adds value should be implemented. It is important to provide automated services that provide genuine value over solutions attainable with direct manipulation.

4.2.7 Aesthetic and minimalistic design

Dialogs should not contain information that is irrelevant or rarely needed. Every extra unit of information in a dialog competes with the relevant units of information and diminishes their relative visibility.

The widgets and objects used for interaction with the end user should be minimal, and under no circumstances should these distract or annoy the user in any way.

4.2.8 Employing dialog to resolve key uncertainties

If a system is uncertain about a user's intentions, it should be able to engage in an efficient dialog with the user. However, care should be taken that such a dialog does not unnecessarily bother or completely annoy the user.

4.2.9 Interaction with the end-user and Visibility of System status

The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

The user interface should have data-tracking display abilities which allow an end-user to actively monitor the execution of the workflow at all times. He should be able to see which service in the workflow is currently executing graphically (change of color, etc.) and via text messages. A user should always be informed when a given service finishes execution, so the user can inspect the output results and make modifications to the workflow if need be.

The user should be provided with status information at all times as to how far along a particular service is in execution and approximately how much longer it will take for it to execute completely, so the user can make decisions based on that. Thus the user should have the

complete capability of inspecting the execution of a workflow and its outcome, and go back and change the flow of control in the workflow if needed.

4.2.10 Allowing efficient direct invocation and termination

A system that is exploratory in nature, and is operating under uncertainty will sometimes make poor decisions about invoking, or not invoking an automated service. Providing efficient means by which users can directly invoke or terminate the automated service can enhance the value of agents providing automated services.

4.2.11 Interaction with a Central Repository

A central repository should be made available to the user, from which he can choose predefined tasks or services to chain together to form a workflow. A well-known web-service registry service is IBM's UDDI, but a local (or central) database or a flat file system can be used easily. What is important is that these tasks/services are pre-defined and available to the users for use at all times (preferably in an intuitive drag-and-drop manner), and can be modified according to the end-user's preference.

Also, the end-user should be able to publish services to this same repository. In collaborative environments, the tool will be used by various domain-specialists. It is always helpful if a user creates a task and registers it on the central repository, so another user (or the very same user) can use it in his workflow at a different time.

It is important that there be descriptions associated with such tasks in the repository so the end user does not spend time testing a service and figuring out its functionality. An architecture of this kind, as discussed in Section 2.3, enables an efficient way to find and use services. This also enables fault-tolerance, since if a given service implemented in a workflow fails or shuts down, the user can choose an alternative service that implements the same functionality. It is important that the service discovery module of the registry have an interface that allows the user to describe the functionality desired and use it in his workflow in an intuitive manner.

4.2.12 Data-Mapping

A good WFMS should provide the user with data-transformation capabilities (preferably graphical entities) that allow him to map and transform data as it passes from one service to the next. However, more basic transformations (like integers to strings) should be taken care of by the system itself, only the more workflow-specific (or domain-specific) ones should be left to the end-user.

4.2.13 Decision-based execution

In many scientific workflows, certain data flows are executed repeatedly, whereas others are executed based on the logical outcome of a given condition preceding this data flow. The WFMS should have the ability to let the user insert decision nodes and loops in the workflow. There has

to be a way to translate the user's logic into the underlying workflow language. Specifically, the following kind of nodes should be allowed and processed:

- Conditional Branching: The user should be able to specify conditions that will be evaluated and based on the outcome the workflow will take one route or the other. Support for logical operators AND, OR, XOR, NOT, etc. is important.
- (Conditional) Looping: The user should be able to execute parts of workflows (subflows) within for and while loops. It should be easy for him to specify iteration criteria and parameters.
- Splits and Joins: The user should be able to “split” the workflow by having services in the flow execute in parallel if the output of one does not depend on the output of the other. He should also be able to “merge” or join results obtained from such execution and use it as an input to the following service.

4.2.14 Visualization of output data

It is very important that the output from individual services of a workflow, and the workflow as a whole, be in a format desired by the end user of the workflow. Also, the end-user should be allowed to view the results of the workflow in various different formats permissible given the nature of output (as opposed one specific kind of output). For example, an end-user should be able to view his numeric output as points in space, tabular format, histogram, etc. If the WFMS is domain-specific, such domain-specific output formats should be possible (e.g. for a bioinformatics WFMS, a double-helical representation of the DNA strand should be possible).

4.2.15 Reusability of Workflows

The user should be able to save a workflow along with its status, and should be able to retrieve it at a later time to re-run or re-inspect it. He should have a choice of saving workflow information locally, or on the central repository such that other users can use it. It should also be possible for the user to store the outputs of various services and the entire workflow.

4.2.16 Verification, Validation and Fault-Tolerance

Each of Verification, Validation and Fault-Tolerance are features that should be provided by the underlying system that supports the WFMS. However, there always should be user-friendly ways to convey this information to the user correctly, and not confuse him. Error and system messages should have priorities, and should be relayed to the user with appropriate suggestions. If a user commits a mistake while creating the workflow, he should be corrected and advised on how to correct it. If a service fails, the user should be given a choice of alternate services that he can use.

Thus, recognizing, diagnosing and recovering from errors should not be overwhelming for the user. The system should also implement features that prevent errors on the user's part.

4.2.17 Minimizing the cost of poor guesses about action and timing

Designs for services and alerts should be undertaken with an eye to minimizing the cost of poor guesses, including appropriate timing out and natural gestures for rejecting attempts at service.

Recovering from errors should be fairly easy to achieve.

4.2.18 Help and Documentation

Even though it is better if the system can be used without documentation, it is good to provide help and documentation. Any such information should be easy to search, focused on the user's task, and list concrete steps to be carried out, and not be too large.

4.2.19 Security and Protection

Security is a concern with WFMS's used in collaborative environments, but it is usually managed by underlying layers. However, care should be taken on the user-interface and in case of extremely complex security policies so that security features do not baffle the user. It has been shown that scientists prefer easy-to-use systems with simple security features, than systems with more complex security features.

The system should implement features that allow the user to backup their workflows in a simple manner. Recovering data and partly complete workflows from failure should be easy to achieve as well.

4.2.20 Web-based GUI

It is important that Scientific WFMS's make their visual components available via the browser for users who may not want to install and run the system locally. Although systems such as scientific workflow systems are intended to be client-centric and be available to the end-users as software suites by themselves, the user should have an option of running the workflow using a browser, although it may have limited capabilities for security reasons. The browser interface may be a Java Applet, however, care should be taken about the visual interface if the user will have the capability of accessing local files and data via the applet. An intermediate servlet or Struts framework may have to be used to access such local data.

4.2.21 Interoperability

In collaborative environments, end users typically use a variety of tools and legacy systems to form various workflows. Although we describe a service-based architecture here, the scientist may use a third-party tool for other purposes such as visualization, reporting, etc. Then, the visual interface of the given WFMS (along with supporting layers) should have the functionality of letting the user conveniently hook into outside systems. One needs to make sure that the user is offered a consistent look and feel when accessing other systems to avoid confusing him, which is not a trivial task.

This is one of the more difficult and challenging features to achieve in a WFMS, and has been a topic of research for a lot of systems.

Apart from the above-mentioned features, each WFMS created for a specific domain will have different requirements for its visual component. At the very least, the user-interface should satisfy these features, and be accessible, flexible, scalable, extensible and reliable.

5 The Pilot system

We explained in the previous sections our approach of using iterative design and a prototype in our user-interface design cycle. The pilot system, created during the inception of this project was used as a prototype, based on requirements and feedback received from our pilot user, Dr. Matthew Coleman. A design process of this kind helps understand the requirements and architecture better, since both the developers and end users have a better understanding of the functionality desired after several iterations of design and development. It also helped us evaluate, select and formulate our interface evaluation criteria.

As must be obvious by now, the most intuitive way to describe a workflow is in the form of a directed acyclic graph (DAG) [DADS05, Dav91], where the vertices (nodes) represent specific tasks or services and the edges represent flow of control and data. However, a more complete general activity network (GAN) diagram could be used [Elm64, Elm66, Elm95]. The user-interface of a WFMS should allow the end-user to create an “executable” graph consisting of services and information about the sequence of execution and the data and information passed between the nodes. In this section we introduce our prototype system, SDMSWE [Cha02, LAGM03] that was based on the DAG view of the process.

5.1 The SDMSDWE System

SDMSWE stands for SDM Scientific Workflow Environment. SDMSWE has a user interface that closely manifests a user interface designer’s idealistic specification of a scientific WFMS. This was the prototype system built by the SDM team before adopting the new Ptolemy-based

architecture. Our earliest prototypes on paper were designed keeping in mind ease of use as a primary goal, hence SDMSWE has a very simple yet effective design. However, it has limited capabilities when it comes to handling heterogeneous and highly structured data, and thus it was replaced by a more robust architecture.

The SDMSWE workflow tool lets the user define, construct and execute a sequence of data and analysis services the user wants to invoke. It creates a Web Service based environment for executing and monitoring workflows.

Below is an architecture diagram of the implemented system.

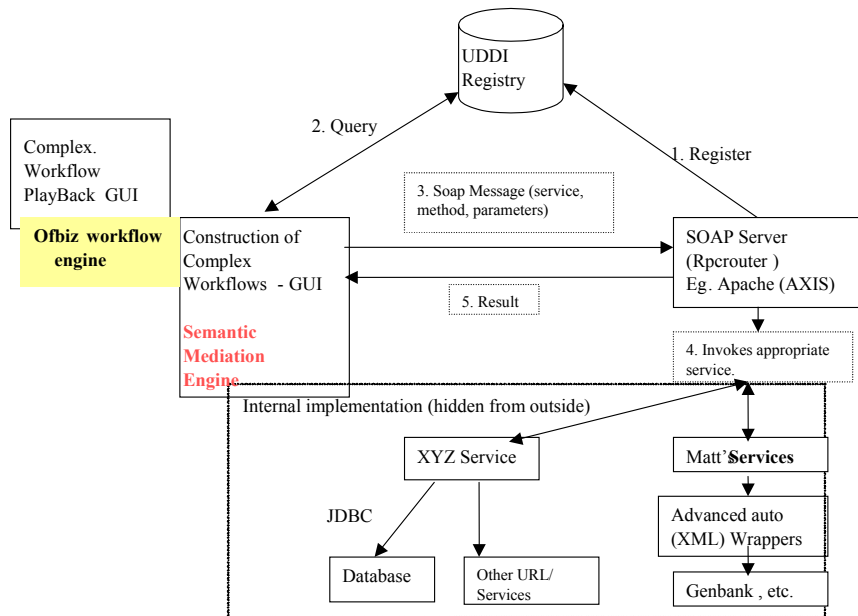


Figure 5.1.1: Architecture of SDMSWE

The workflow editor allows the user to create exploratory workflows using various tasks (web-services). These web-services are registered on the UDDI [UDDI04], and also on a backend PostGRE database associated with OfBiz [OfBiz03] workflow engine. On execution, the SOAP server (here, Apache Axis) is called which invokes each service in the sequence specified on the user interface. The service definitions are stored in a database associated with the Axis server.

This system has the following characteristics:

- Easy to use interface for creating, editing, saving and monitoring linear and non-linear workflows that involve multiple services and data resources, possibly heterogeneous in nature.
- The workflow consists of task nodes that are connected into a directed graph. Each task represents a unit of work to be done which may invoke a (web) service on the remote machine, or a service over the web, or even a local program. It may also deliver information from a web page.
- The workflow editor can dynamically locate services registered with the UDDI and extract service details. The workflow editor allows the user to select Matt's (PIW) [ABB+03] services from a service menu and draw them as tasks. In essence, this eliminates the need on the users part to enter parameters for a task. The parameters are automatically assigned to the task. All the user needs to do is select the service and click on the workflow designer to see the task drawn.
- The services (service details) are extracted from the workflow engine database. In our case it's the PostGRE database that is part of the OFBiz workflow engine.

- The **workflow engine** is central to the workflow editor architecture and is responsible for composing SOAP calls based on the service interface parameters supplied to the workflow components. It provides basic syntactical verification of workflow parameters. A more extensive verification would involve verifying the workflow description against the service description in the registry.

There are two modes of operation for the workflow GUI. One is the **workflow composer** mode used to construct workflows. The other is the **playback** mode where the end-user initiates and tracks the execution of a defined workflow. The execution engine invokes the services defined in the workflow. Alternatively the user can directly go to the UDDI registry, find the interested service, and invoke those services directly (as a web page).

SDMSWE does not use any standard language for describing workflows; it uses an XML-like description that only the SDMSWE workflow engine can understand. At this point, SDMSWE has very limited support for loops and no support for conditional execution. It does not have sophisticated verification and validation features in the most recent version.

5.1.1 Using the SDMSWE system

To create a workflow, a user would follow these steps:

- 1) The user starts with clicking the “Services” button. By default, a “Start” node is inserted in the editing area. The user can click anywhere in the editing area, and an empty Service node will be inserted there. This service node is an abstract entity that represents a web

service or a task that the user wishes to execute. The user may delete any service at any given time by right clicking on the service and clicking “remove”.

- 2) The user can “bind” these abstract service nodes to specifications of his choice, making the services capable of performing specific tasks. To do so, he would right-click on the service, and click “general properties”. Here he can enter parameters such as name, (web) service URL, method name (function to be invoked), etc.
- 3) The user can connect these service nodes using “Connectors”. Connectors can be used by dragging them through the connector point of one service to the connector point of the next. The direction of the connector arrows determine the data and control flow in the workflow.
- 4) Once done, the user can insert an “Exit” node on the editing area, and connect the last-in-sequence service node with it.
- 5) The user clicks “Run” to execute the workflow. As the workflow executes, each service that is being executed is yellow in color, and once finished, it changes its color to green.
- 6) The user can also create a “loop” in the workflow by inserting “start loop” and “end loop” before and after the sequence of services that he wants to execute iteratively. In this case, the iterator is a list of Accession ID’s created as a result of the Microarray Analysis in the first step.
- 7) As each service executes, the output is shown as XML format in a pop-up window. At any time during the execution, the user is allowed to “Pause” the execution and resume it later.

5.1.2 The SDMSWE Graphical User Interface

As mentioned earlier, since the start of this project, we realized that ease-of-use is one of the most important goals of our tool. The main reasons scientists expressed interest for a new tool was that other existing tools were too complex. The pilot user for this tool, Dr. Matthew Coleman (LLNL), was a part of the design process. It can be said that the design process was indeed User Centered (UCD) [DFAB97, VIR01], since we made changes to the tool iteratively based on Dr. Coleman's feedback and comments.

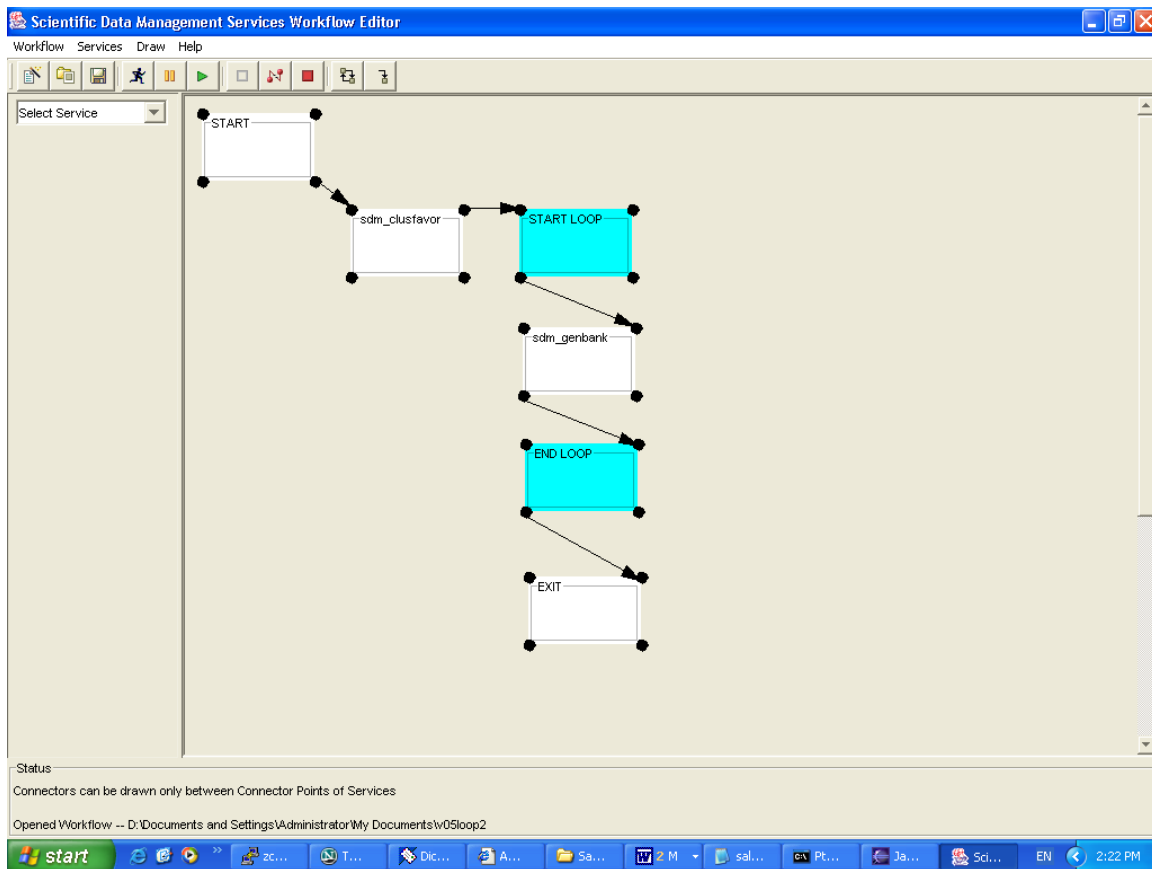


Figure 5.1.2a: SDMSWE Screenshot

As can be seen in the screenshot, the layout of the SDMSWE is very simple yet aesthetic. It closely resembles many other graphical user interface tools. Such a design ensures that the user does not get intimidated by a user interface that looks new and different than what the user may be used to. The colors used are pleasing to the eyes and suggestive of the message being conveyed. For example, green means task has executed successfully; red error messages indicate interruption, etc.

Right from the start, the tool presents helpful tips and comments to help the user, at the status bar. At all times while the user creates workflows, the status bar displays the current status of the system, and helpful comments if needed.

Each “action” in the menus is represented graphically in the toolbar along with helpful tool tips. These toolbar buttons are intuitive and very easy to understand as they closely resemble similar widgets performing similar functions in other GUI’s. The “drag and drop” look and feel of the interface helps the user construct the workflows without looking at too many places for complicated instructions. It also decreases the number of mistakes (and eventually errors upon execution) the user would make had he been creating services (i.e. parameterize them) by himself.

The interface helps the user avoid errors by clearly guiding him through the workflow creation and execution process, but in a manner such that the user does not constantly get annoyed and distracted by the messages if he doesn’t need them. The tool points out basic syntactic errors to the user so that he doesn’t encounter any major errors during execution. Users usually think that

the errors encountered during execution are more difficult to fix, and at times users altogether scrap the current design and create a new one.

The interface has good support for the user as far as help and documentation goes. The help menu points the user to pertinent information about the tool (version information, date, etc.), and links to a help manual online.

When it executes, the tool tracks the progress by changing color of the nodes. This is illustrated in 5.1.2b.

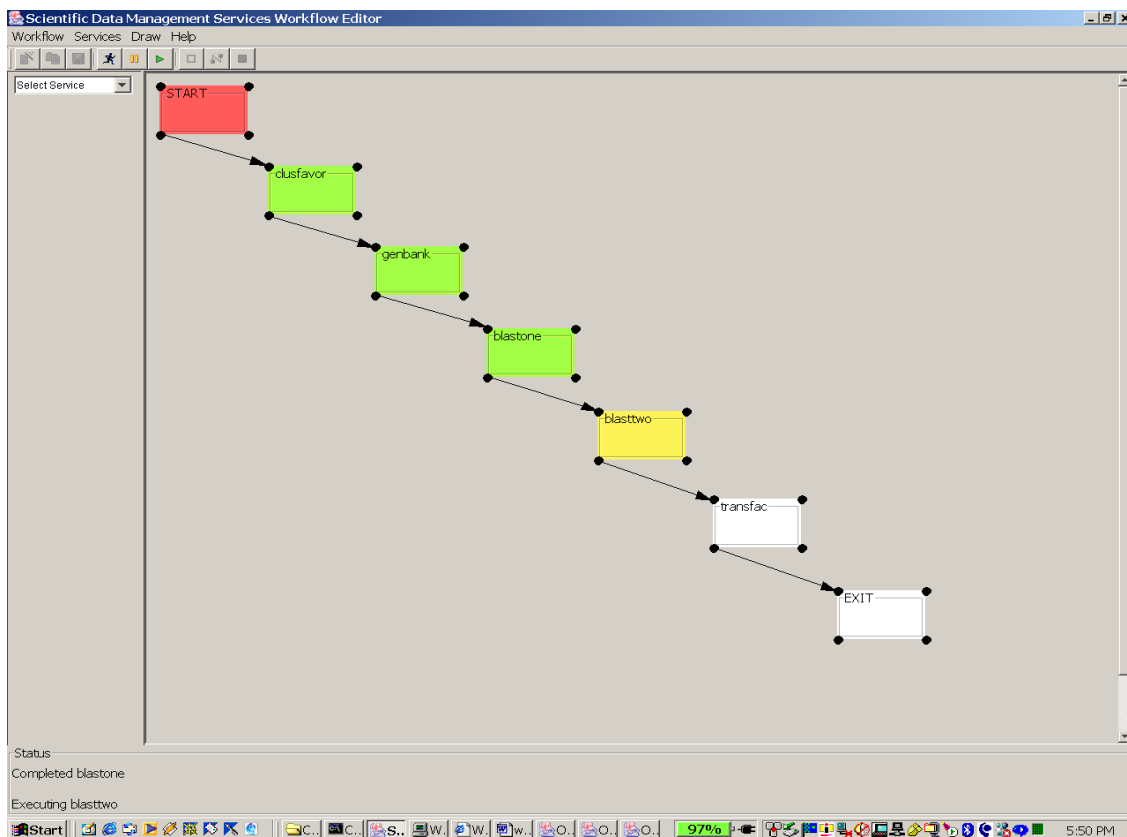


Figure 5.1.2b: Execution in SDMSWE

5.2 Lessons learned from the Pilot

Since SDMSWE was created as a rapid prototype using feedback from our pilot user iteratively, the final version released caters to a very specific kind of workflow as described in [Cha02, ABB+03]. We realized a need to create a workflow system that would be able to support a more generic set of applications. Although the SDMSWE architecture is object-oriented and easily extensible to support a variety of services, a lot of work that would go into extending the system would be redundant since other similar systems have been created with required modules. Our goal, then, is to modify an already existing system (or more than one systems) to develop an architecture as envisioned by the SciDAC initiative, and as explained in Section 3.

Following are the more noteworthy issues, discussed above, that led us to adopt a Ptolemy-based architecture:

- The SDMSWE system does not use a specific industry standard to represent data. All data was represented using a format similar to XML, which although worked for smaller sets of data, was inefficient for more structured and heterogeneous data.
- The SDMSWE has limited capability as far as looping and conditional branching goes. Although not impossible, the task of implementing these functionalities would have been redundant since other similar systems (like Ptolemy) already implement them and are easily extensible.
- The system has limited visualization features, and implementing functionality that allows the user to visualize the data in the format desired will be a non-trivial task. It would be much easier to integrate the system with other visualization tools.
- Since the system does not implement a standard for representing data, it is difficult to integrate the system with third party tools.

- SDMSWE system is not a generic SWFMS. It allows the user to create very specific workflows in the bioinformatics domain. We require a solution that would allow the user to create workflows in different domains.

6 A Comparative Evaluation of Scientific WFMS

As seen in Section 4, a good Scientific Workflow Management system should allow the end-user to create, execute and view the results of an executing workflow. It should provide the user with a repository that the user might use to sequentially chain together tasks or services to form a workflow. The user-interface should implement good Direct Manipulation [Sch86] techniques such that the user can freely modify the workflow to cater to various needs, yet at the same time not be too complex and difficult to learn and use.

In this section, we compare tools (including our SPA system [SPA05]) based on the heuristics defined in Section 4.2. As shown in [NM94], when an evaluator is not using the system as such, it is possible to perform heuristic evaluation of the user interfaces that exist on paper, or in our case, where the system has been implemented but is not available for use due to various reasons (e.g., licensing). Instances such as these will be pointed out wherever applicable.

The WFMS's considered here are chosen out of many commercial and research-based tools available using their relevance to our research goals and area of study, as well as their domain of application [CompList04]. The tools evaluated here represent, in author's opinion, a reasonable sample of leading edge tools that amongst them cover a broad spectrum of scientific workflow automation and problems solving needs as covered in this study. Specifically they:

- Allow users to construct and execute workflows, and are considered Workflow Systems
- Have support for creation of workflows in the life and physical sciences domain
- Are open source

For each tool discussed here, we first describe the overall functionality and appearance of its user-interface. We then point out usability problems that we discovered through numerous evaluation passes, and heuristics that the system does not satisfy.

A good Scientific Workflow Management system should allow the end-user to create, execute and view the results of an executing workflow. It should provide the user with a repository that he might use to sequentially chain together tasks or services to form a workflow. The user-interface should implement good Direct Manipulation [Sch86] techniques such that the user can freely modify the workflow to cater to his needs, yet at the same time not be too complex and difficult to learn and use.

6.1 Scientific Process Automation toolset [SPA05]

The Spa project was introduced in Section 2.5.2. 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 [Pto04], and is a part of a larger project, Kepler [Kep04a]. Kepler's goal is to produce an open-source scientific workflow system that allows scientists to design scientific workflows and execute them efficiently using emerging Grid-based approaches to distributed computation. Kepler's goals and objectives are described in detail in [Kep04b].

The Ptolemy II project [Pto04] 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, including for example analog and digital electronics, hardware and software, and electronics and mechanical devices and systems that mix a variety of different operations, such as signal processing, feedback control, sequential decision-making and user interfaces. 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.

6.1.1 Main features of Ptolemy II

- The components in Ptolemy II are designed to be domain polymorphic, meaning that they can interact with other components within a wide variety of domains. It implements an architecture where components can be easily designed to interact in a number of ways.
- It provides a rich set of interaction mechanisms embodied in the Ptolemy II domains. It supports an abstract formal framework that describes models of computation at a Meta level. This eliminates the need to perform awkward translations to describe one model of computation in terms of another.
- Its workflow exchange language is called MOML. It is an XML-based language, but it is not a standard such as BPEL.

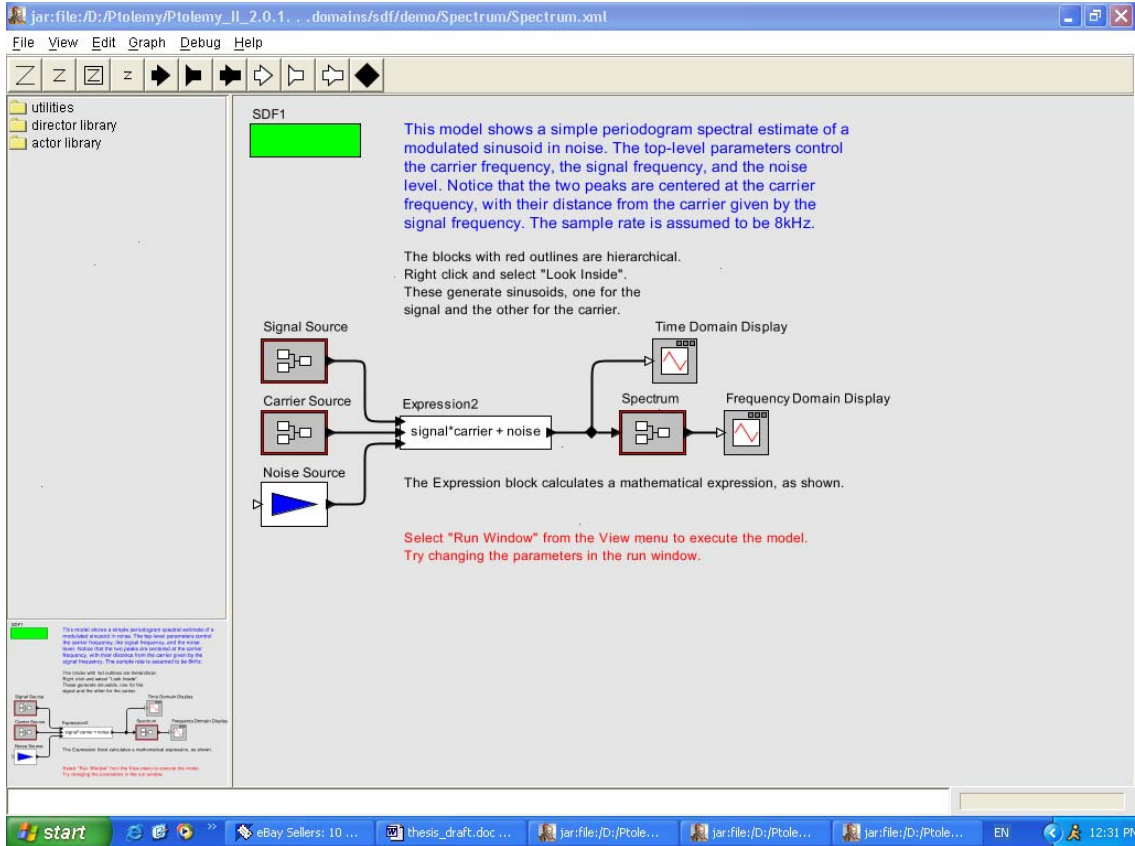


Figure 6.1.1: Screenshot of a Signal Processing Workflow in the Ptolemy II environment

6.1.2 The SPA system

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 and 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 [Kep04b]. We describe an astrophysics

workflow in Appendix II. We describe below the Promoter Identification Workflow (PIW) as described by our pilot user. Details of the PIW are given in Appendix I.

The PIW aims at constructing models of transcription binding sites to identify co-regulated genes with data obtained from conducting microarray analysis. Specifically, the SPA project has developed a set of “Bio-Services” to be used in executing the PIW workflow, wrapped as Actors and made available as a new Actor package. These Bio-Actors are executable forms of standard programs such as NCBI GenBank [GenBank04], BLAST [BLAST04], Transfac [Trans04], Clusfavor [Clus04], etc. These Actors implement the XWRAP [XWRAP00] system developed at Georgia Tech to convert these programs into web-services. The SPA project has also modified the user-interface to allow the user to pause and inspect intermediate results of the workflow, and proceed using only a part of the data as an input to the remainder of the workflow if needed. A generic WSDL (Web-Service Description Language) actor has been created that allows the user to execute a web-service program of his choice. The user enters the URL of the web-service, due to which the Actor binds itself to a specific definition of a web-service and gets customized. This ensures that the user can easily extend the Actor package.

Following are the more prominent characteristics of Ptolemy-based SPA

- **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.

- Retrieving Saved Workflows: In PtolemyII workflows can be saved as XML/MOML descriptions. It can be invoked later and used with different inputs.
- Visualization: PtolemyII has special applications (graph plot, Matlab) and libraries, which provide better visualization of data.
- Scalability: PtolemyII is based on Java technology and scalability is achieved by developing modules or components that interface with the existing interfaces of PtolemyII.

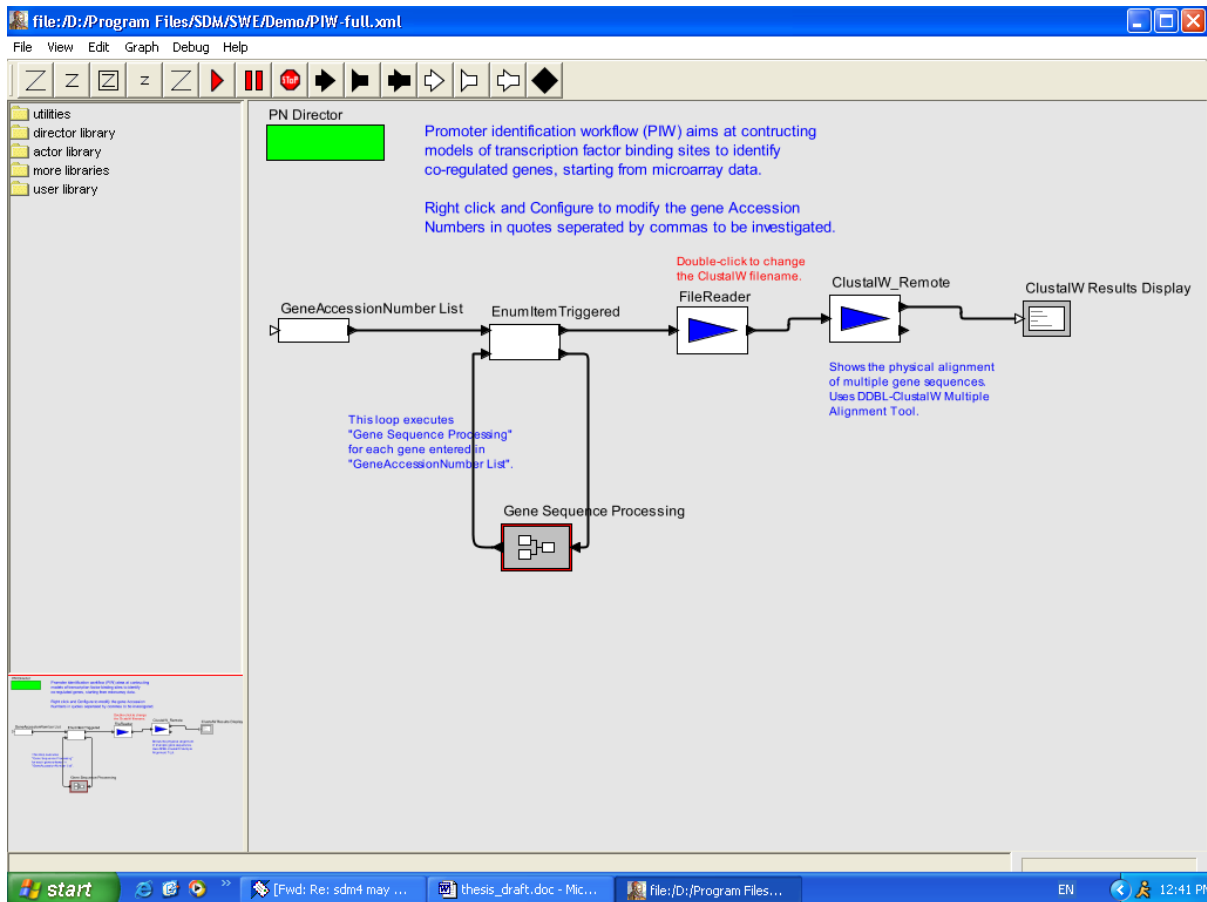


Figure 6.1.2: Screenshot of the high level PIW in the SPA environment

6.1.3 Some Usability Issues

Following are some usability problems discovered in Ptolemy II-based SPA -- They do not focus only on the user interface, but on the end-user experience as a whole.

- The user-interface requires a significant learning curve. Although the “Actor” and “Director” metaphors make it easier to grasp the concept of executable tasks and an entity controlling the execution of the workflow, it is not immediately obvious to the end user how to implement these.
- Actors, Directors and other components listed in the component tree, do not come with tool tips or any other help. Since the status bar is “idle” for most part before execution (i.e. while the user is constructing workflows), the user has no way of determining the function any of these components serve unless he uses the Help menu.
- Service Components: Ptolemy II supports only those applications in workflows that are customized to run in its environment as “Actors”. So for any algorithm or program that the user wants to run in the Ptolemy environment, it would have to be packaged as an “Actor”, which is not a generic job that the end-user can perform. Developers would have to be involved to help achieve this, hence slowing down the workflow process even more.
- Lack of Central Repository: Ptolemy accesses and executes local files in MoML [Pto03] format, and does not implement a central repository for workflows that can be accessed by multiple users within or across the organization. Thus, to share workflows, users would have to share the workflow files (and thus, MoML files) via email or file-sharing programs.

- Since Ptolemy II does not implement a central repository, new services (or Actors) or updated services will have to be shipped frequently, requiring the end user to update his version of the system with a newer patch each time.
- Ptolemy II does not support execution tracking for all Director types. Thus, at the time of execution, it becomes difficult for the end user to see how far the workflow has progressed unless he sets up breakpoints or intermediate result displays. At this point, tracking workflow execution is possible only in the Discrete Time (DT) domain.
- The system does not inspect the workflow before execution for any obvious problems, thus there is lack of verification and validation functions in the system.
- Error notification to the user is not very understandable. If the user encounters any errors, the message is usually not understandable by the end-user, so the user does not get any help to rectify his mistake. Most of the error messages are exception names and stack traces which are understood by developers only.

6.1.4 Other evaluations of SPA

Ptolemy-based SPA toolset has attracted the attention of the scientific community for the promise it shows in revolutionizing the way scientists carry out their experiments. At NC State, we had two groups of computer science students, one graduate [PSC04] and one undergraduate [SSYA04], evaluate different aspects of the solution from the perspective of personnel trained in information technology, but not the workflow domain.

In [PCS04], the authors study the use of Scientific Workflow Management Systems as a way of enabling information exchange between data sources. They study and critique a few existing systems with special emphasis on Kepler. Although SPA itself has good installation features, the authors note that installing the Kepler package in general and supporting software is “confusing”. The authors observe that the user interface is often complicated. They suggest that the problem could possibly be remedied by providing a robust library of sample applications – something that both SPA and Kepler projects are working on. They note that the goal of Kepler, to reduce the cost of creating and maintaining workflows, may be threatened by components such as web services which are outside the control of the user.

Overall, the authors seemed to have some difficulties in installing and using the Kepler suite as a whole. However, they also note that a lot of these problems arise from the lack of standards and from the difficulty of integrating external services, problems inherent to such an architecture, and software systems that try to build on them.

On the other hand, authors of [SSYA04] were able to install and use the SPA system with more ease. In this study, the authors evaluate the SPA system overall by reviewing its installation features, user interface, development and maintainability features. They observe that installation of the SPA suite is fairly easy for the novice user if supplemented by the supporting help documents and manuals. Creating workflows is easy for simpler workflows since most Actor names are conventional and easy to understand. For more specialized workflows, the user has to refer to the descriptions associated with each Actor.

The authors observe that a serious drawback with the SPA system is the lack of a central repository. They suggest the use of UDDI [UDDI04] or StrikeIron [SI05], a web site offering a web service directory similar to UDDI, to solve this problem. Details of their evaluation are given in Section 6.6.

6.2 An Evaluation of SCIRun

SCIRun [SCIRun04] is another, very successful, modular dataflow programming Problem Solving Environment (PSE) [69], which we alternatively refer to as a WFMS. It allows integration of the steps of a scientific workflow as components in a single, unified, extensible, problem-solving environment. Specifically, SCIRun is considered as a PSE framework upon which application specific PSE's are built. Specific PSE's, such as the BioPSE, are provided as packages within SCIRun. Thus, the user has the ability to manage each step in a sequential computing process, and to create batch processes that execute repeated simulations.

A workflow in the SCIRun environment is called a Dataflow Network (and is very reminiscent of AVS data flows), or **Net**. Such a Net is created by connecting **Modules** together, which are components that perform specific functions on a data stream. Each module reads data from its input ports, calculates the data, and sends new data from output ports. Modules are connected by means of **Pipes**, which represent data flowing between modules. A detailed description of Modules and Pipes is given in the following sections.

One of the significant functionality that SCIRun implements which other WFMS's do not, is the interrupt ability - ability to intervene and control execution anywhere in the chain at any time during the execution. This is also referred to as *computational steering* [VS96]. The SCIRun environment also comes with a series of powerful visualization components. The combination of steering, visualization and component integration allows the user to spontaneously explore a problem.

6.2.1 The SCIRun User Interface

On starting the application, The SCIRun **NetworkEditor** window appears on the screen (see figure below). The NetworkEditor screen consists of three frames: the **GlobalView** frame in the upper left corner, the **Message** frame in the upper right corner, and the **NetEdit** frame.



Figure 6.2.1: The SCIRun Window Frame [SCIRun04]

The NetEdit frame is the large area where networks of modules are constructed. Since the NetEdit frame is extensive, the GlobalView frame shows which part of the NetEdit frame is currently being viewed. The Message frame reports errors, warnings, and other such important information. A sample workflow in the SCIRun environment is shown below.

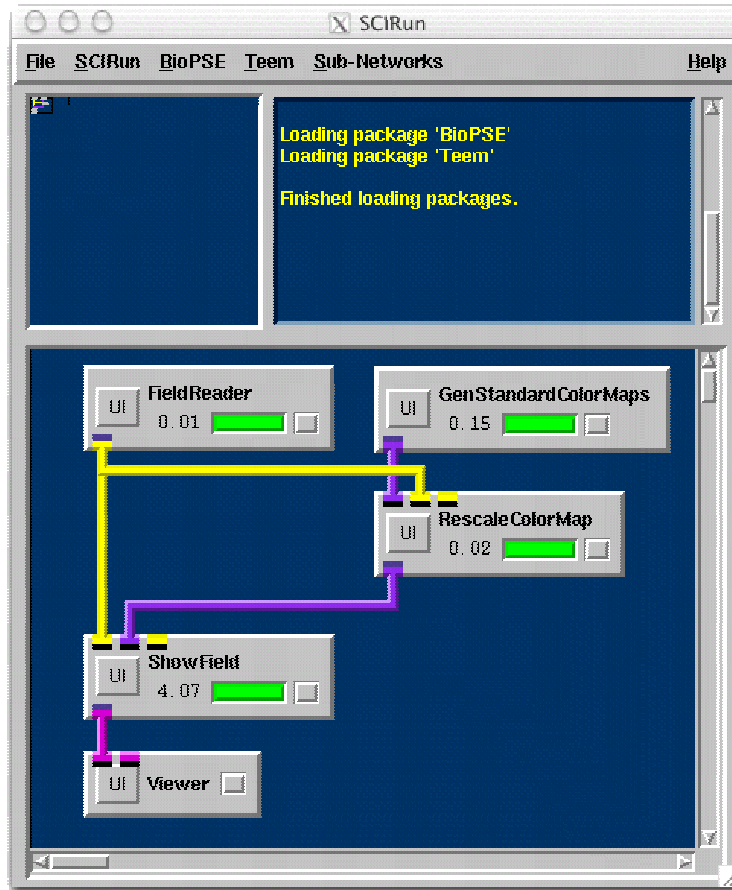


Figure 6.2.2: SCIRun Sample Workflow [SCIRun04]

6.2.2 Modules

A module is a single-purpose unit that functions within a dataflow environment. Modules have at least one input port for receiving data, located at the top of the module, or one output port for sending data, located at the bottom of the module.

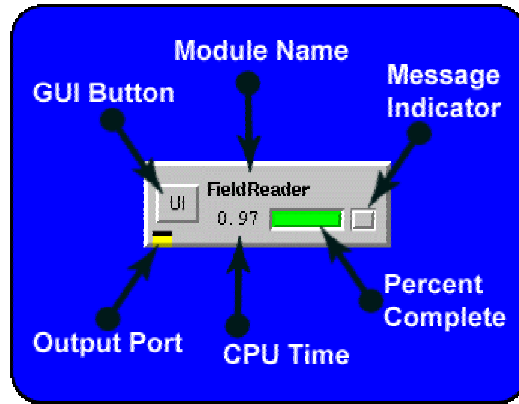


Figure 6.2.3: Anatomy of a SCIRun Module [SCIRun04]

All modules have an indicator that alerts the user to messages that exist in a module's log. Different colors represent different types of messages. Gray means no message, blue represents a Remark, yellow represents a Warning, and red indicates an Error. To read messages, the user clicks the module's indicator button to open the log window.

Most modules are created using predefined module files in the SCIRun environment. These modules have been pre-written using TCL script, and in some cases the user may not be able to create new modules from the scratch.

6.2.3 Pipes

Data is transferred from one module to another using dataflow connections, commonly referred to as **Pipes**. Each dataflow pipe transfers a specific data type in SCIRun, denoted by a unique color. Pipes run from the output **Port** of one module to the input Port(s) of one or more other modules. Ports of the same color correspond to the same data type and can be connected. The pipes in SCIRun have various sophisticated features that allow users to transform data as it goes from one Module to another.

The **Viewer Module**, which is usually the last module to be added to a Net, allows the user to visualize the final output data as per his requirements. SCIRun allows **Random** and **Streamline** visualization, and multiple concurrent visualizations are possible as well. By means of **Widgets** in the Viewer, the user can interactively control features of the display.

6.2.4 Some Usability Issues

- While the SCIRun environment is relatively easy to use, the user may not have full control of the SCIRun environment if the user wishes to create a workflow from scratch. Specifically, SCIRun modules that are used to create a workflow, can be imported from the file system as pre-written scripts (files with a .fld extension). Although SCIRun tries to include as many different module types as possible, when the user wants to create a module from scratch the required effort is considerable..
- SCIRun cannot be run as a background application.
- The error messages displayed are at times confusing and cryptic, and do not necessarily help pin down the problem.
- To create a Net (or a workflow), the user uses predefined modules (files with an extension .fld). These files are downloaded with the SCIRun application to the user's machine. However, since this system essentially lacks the concept of a central repository, and uses the local file system to store and retrieve Nets, it is a challenge to share these Net's, using SCIRun (although an option is to use a file-transfer program).

- The files that represent the Nets and Modules are not in a form that can be edited by the novice user. A good WFMS should allow the user to edit the text form of the workflow to make smaller changes that do not require the resources of a graphical user interface.
- SCIRun does not allow very extensive data mapping or transformation of data as it passes from one module to the other. Data Mapping and Transformation at this stage are allowed using Pipes, but the user does not have full control over the data at this point.
- As mentioned earlier, SCIRun does not implement any standard language or format to represent workflows (such as XML, XPDL, etc.), thus it becomes difficult for the user to edit the workflows or individual modules without the help of a graphical user interface.
- The SCIRun environment does not allow web-based (or browser-based invocation) of the user interface, due to which, the user has to download the whole system on his machine even if he wants to just try out a sample Net.
- SCIRun has been originally written in C/C++, and uses TCL scripting for some part. Since more existing WFMS's are Java-based and use XML for scripting, it is difficult to connect SCIRun to different systems and allow interoperability.

6.3 Evaluation of Enhydra JaWE

Enhydra JaWE (Java Workflow Editor) [JaWE04] is an open source graphical Java workflow system (referred to as a process editor here) that allows the user to create, manage and review process definitions (workflows) using a visual tool. It lets users create workflow process definitions, and store and reuse them at a later time.

JaWE has been developed fully according to WfMC specifications [WfMC04] supporting **XPDL** (XML Process Definition Language, [XPDL02]) as its native file format and LDAP connections. Thus, it can be used to edit/view any XPDL file that conforms to WfMC specifications. The final XPDL file produced from using this tool can be interpreted at runtime by a workflow engine (say, ENHYDRA Shark, [Shark04]). As a consequence of involving the WfMC proposed package concept, JaWE divides itself into two logical parts: **Package Level** and **Process Level**. As will be explained in detail below, the Package level manages entities and attributes within the Package, while Process level manages entities and attributes within Workflow Process Definition.

6.3.1 Package Level

A view in Package level is shown below.

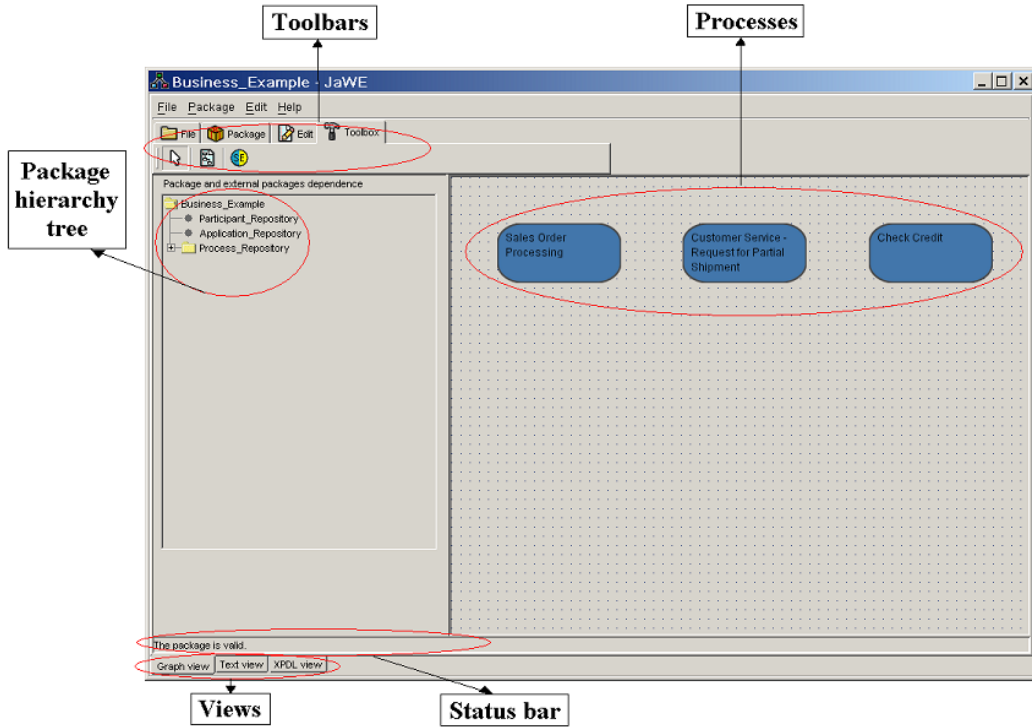


Figure 6.3.1: Package Level in JaWE [JaWE04]

At the package level, the user is allowed to choose among several XPDL views such as graphical, text, and xpdL view which represents the XPDL as it will be saved into file.

The graphical view of package level window shows Package Hierarchy Tree on the left. The root of the tree is main package and branches are imported external packages. Since all packages are assigned unique IDs, JaWE allows circular references, though tree expansion stops on the package that is displayed before.

6.3.2 Process Level

The Process Level view is displayed below.

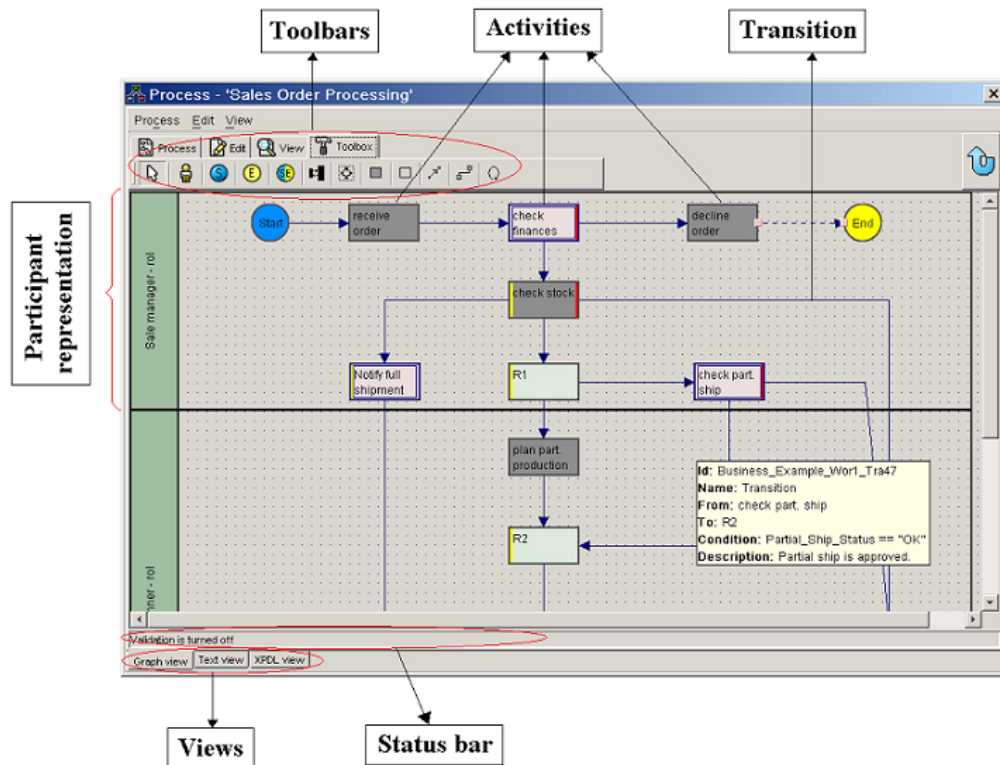


Figure 6.3.2: Process Level view in JaWE [JaWE04]




The Process level view is essentially an editing window that can be used for graphical representation of process definition and for defining attributes of entities on that level. In the visible working area, the user inserts visible objects and adjusts them. The first thing drawn must be a participant, after which the user may insert other elements such as **Activities** and **Transitions**.

There are 4 kinds of Activities defined in JaWE:

1. Generic Activity
2. Sub-Flow Activity
3. Block Activity

4. Route Activity

A Subflow is a type of activity whose implementation is another workflow process, so subflow execution mode and parameters passed to it are important. The Route activity does not perform any transaction, and is used for synchronization and conditional branching.

JaWE has two (graphical) types of transitions - simple and self-routed. Simple transitions act as links between two activities as a straight line, whereas self-routed transition between two activities which is 'broken' in three parts. Transactions can be created using transaction tools using ,  or .

JaWE also allows the user to edit and view the **Text View**, where he can manipulate the XPDL directly. The default file extension for JaWE workflow files is XPDL, however, while saving the file, the user may use either the .xpdl or .xml extensions. When a user opens or saves documents in JaWE, its validation against XPDL schema, activity connection validation, graph conformance validation and logic validation is performed and the user is notified of any validation errors if encountered.

6.3.3 Some Usability Issues

- In JaWE, the user is given full support of the system as far as creating activities and process definition goes. To allow the user to achieve all his goals, the system includes various different elements and widgets on the graphical user interface, hence rendering

the system complex at places. Thus, the novice user finds the system difficult to use and get accustomed to at the start.

- The system does not implement very sophisticated computational steering features, thus it becomes difficult for the user to control the various elements of the workflow at all times during execution.
- The error messages displayed to the user are not very friendly. There are instances when exceptions are thrown and the user does not get any meaningful messages, but just some part of the stack trace and the exception name. Such error messages may end up confusing the user if he does not know what caused the error.
- The JaWE system depends on the underlying workflow system to provide most of the data transformation features, so the user interface itself has limited capability as far as data-mapping and data transformation goes.
- The system does not include sophisticated decision-based execution. Again, a lot depends on the underlying workflow system.
- The JaWE system is targeted more towards business processes than the scientific community. As a result, it lacks visualization tools that would allow a user to visualize the data in the scientific domain.

6.4 Evaluation of ObjectWeb Bonita

Bonita [Bonita04] is a flexible J2EE cooperative workflow system based on WfMC [WfMC04] specification, which allows users to specify, execute, monitor and coordinate workflows. Bonita is compliant with Enterprise JavaBeans 2.0 (EJB), hence providing a more flexible and portable environment for distributed applications. The key concept of the Bonita system is the Bonita Execution Engine that is based on the new activity anticipation model proposed by the ECOO team [ECOO04]. The Bonita system is composed of two main components or sub-systems:

6.4.1 The Bonita Modeling Component

The Workflow Modeling Component (**GraphEditor**) allows the user to represent and visualize workflow processes. This tool allows the user to create and edit workflow processes using the graphical user interface.

Workflow processes are constructed by chaining basic functional units, called **Activities**, which can be defined and edited using the GraphEditor tool. Attributes of these activities, such as activity type, activity description, activity deadline, activity role, etc. can be modified by the user. Activities such as these can be connected using **Edges**.

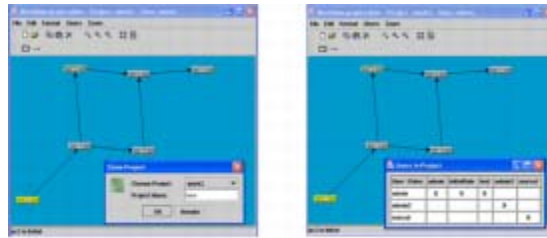


Figure 6.4.1: Bonita GraphEditor tool [Bonita04]

The Bonita modeling component allows the definition and visualization of the process by different users at run time, and it thus involves some coordination and groupware methods for better communication between users.

6.4.2 The Bonita Execution Component

The Bonita Execution environment is based on the principles of flexible data and workflow management which allows activities to exchange data more efficiently. The Bonita system offers a **WorkList** application which allows the user to control process execution. The user WorkList provides different information about the projects of every user. The WorkList is organized in three different lists: the project list, the to do list, and the activity list.

The Bonita Execution Engine allows activities to share intermediate results when executing, thus executing in a non-isolated way. The workflow execution is based on the principle of anticipation [ECOO04], which allows an activity to escape to the traditional start-end synchronization model such that the user can cancel an execution activity or change the activity execution mode at runtime.

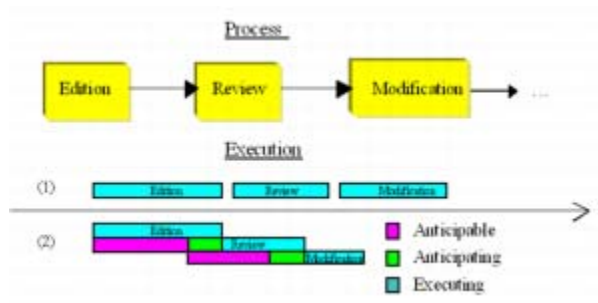


Figure 6.4.2: Bonita Activity Execution [Bonita04]



Figure 6.4.3: Bonita WorkList Execution [Bonita04]

The Bonita Workflow also includes a browser-based environment with Web Services integration (using SOAP and XML) that allows users to define and control the workflow processes by means of a browser.

The basic process implementation in Bonita is the Project CMP Bean that represents the key component of the Bonita Cooperative Workflow Kernel. Every workflow process contains basically data information (process name, user creator, etc.) and associated data in terms of activities, connections between these activities, process data, process properties, and participants.

Following shows the basic Architecture of the Bonita System.

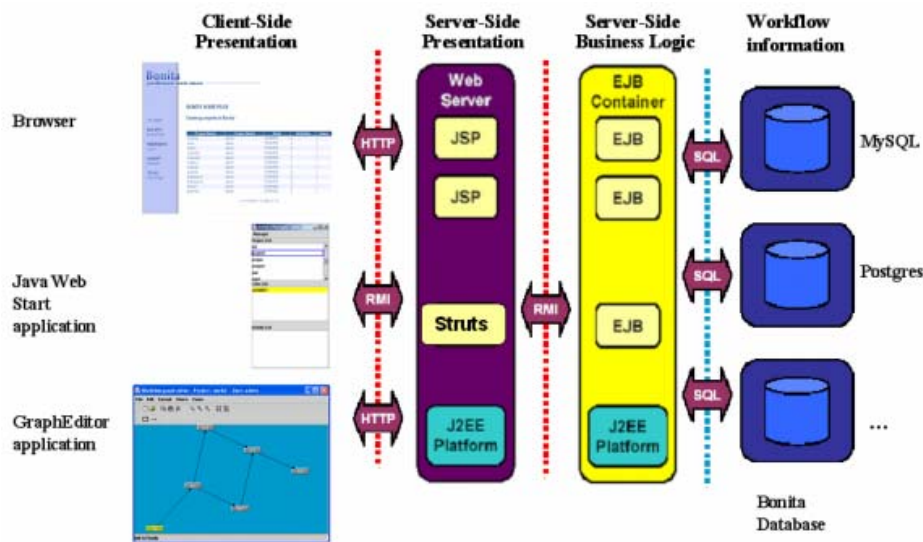


Figure 6.4.4: The Bonita Architecture [Bonita04]

Since the Bonita system is meant to be used in a cooperative environment, it includes some additional features such as the ones described below.

- **The Bonita JMS message service** implementation that notifies the definition and execution changes within a workflow process. Every user interaction is notified to Bonita Kernel and throws a JMS event.
- **Bonita activity deadline service** that uses the Java Management Extensions (JMX) to advice the user if an activity doesn't end at the expected date.
- **Bonita Jabber** service notification that allows the users to receive notifications at real-time and exchange different kinds of messages.

6.4.3 Some Usability Issues

- The Bonita system does not include very sophisticated computational steering facilities, as a result of which, the user does not have complete control over the workflow as it is executing.
- The Bonita system does not include extensive data-mapping and data transformation features. The ones that exist thus far are very basic.
- Since the Bonita system is developed with Business Organizations in mind, it does not handle scientific data very accurately. It also has limited visualization features.
- The Bonita system includes very basic decision-supporting features, and the user has limited ability to define decisions and conditions on his own.
- The Bonita system does not provide extensive verification and validation features while constructing the workflows. If an exception is caught during execution, the system displays error messages on the screen that may not be completely understood by the user.
- The Bonita system does not allow easy interoperability with other systems.
- Workflow Administrators cannot view, modify and update certain project information at this point.
- The workflows described in the Bonita system do not conform to any particular standard, so the user is not able to edit the workflow for minor changes in text format.
- The current version is pre-configured to with the JonAS Application server, and it to change it to work with an application server of the user's choice is not a trivial task.

6.5 An Evaluation of Taverna

Taverna [Taverna04] is an open source workflow system that provides a language [Xscuf104] and the software tools that allow for easy use of workflow and distributed compute technology within the e-Science community. Below is the architecture diagram that depicts the core Taverna software components.

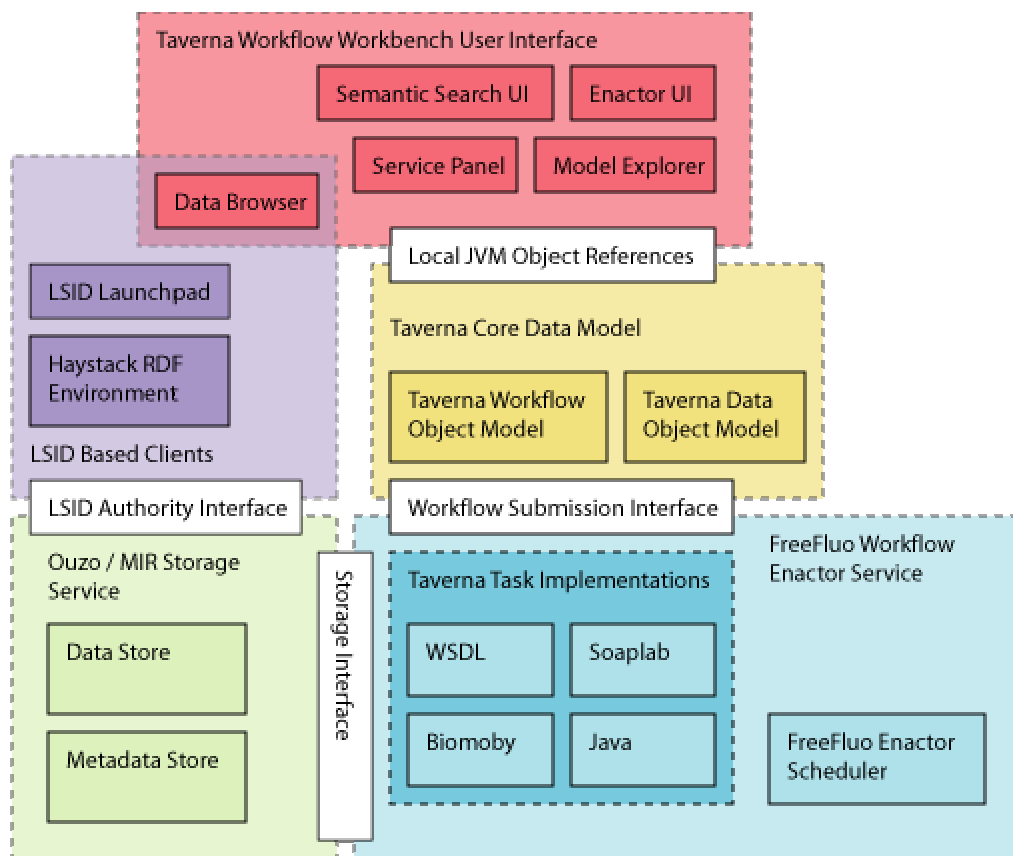


Figure 6.5.1: Taverna Architecture [Taverna04]

The Taverna system can be split into the following three systems:

1. The Core Data Models

The core data models include the object representations of the workflow, all entities within the workflow, and the model for the data values flowing along data links during a workflow enactment.

2. Enactor Task Extensions

In conjunction with an enactment engine, the Taverna task extensions provide concrete implementations of the abstract tasks specified by the Processor (Service) objects within the workflow object model, and perform actions associated with Processor entities. When a workflow and associated objects are submitted through the Workflow Submission Interface, instances of the task extensions are created.

3. Graphical User Interface

The graphical user interface on the client side allows the user to construct workflows, edit workflows, and visualize data as well as manage enactors and data browsing across the results and intermediate, and hence interact with the core data model classes.

6.5.1 The Scufl Workbench

The Scufl Workbench allows the user to create, edit and execute workflows. It acts as a container of windows that provides various views and a controller to manipulate a workflow. On startup, the Scufl Model Explorer, Scufl Diagram view and Service palette are opened by default. A simple workflow is shown below.

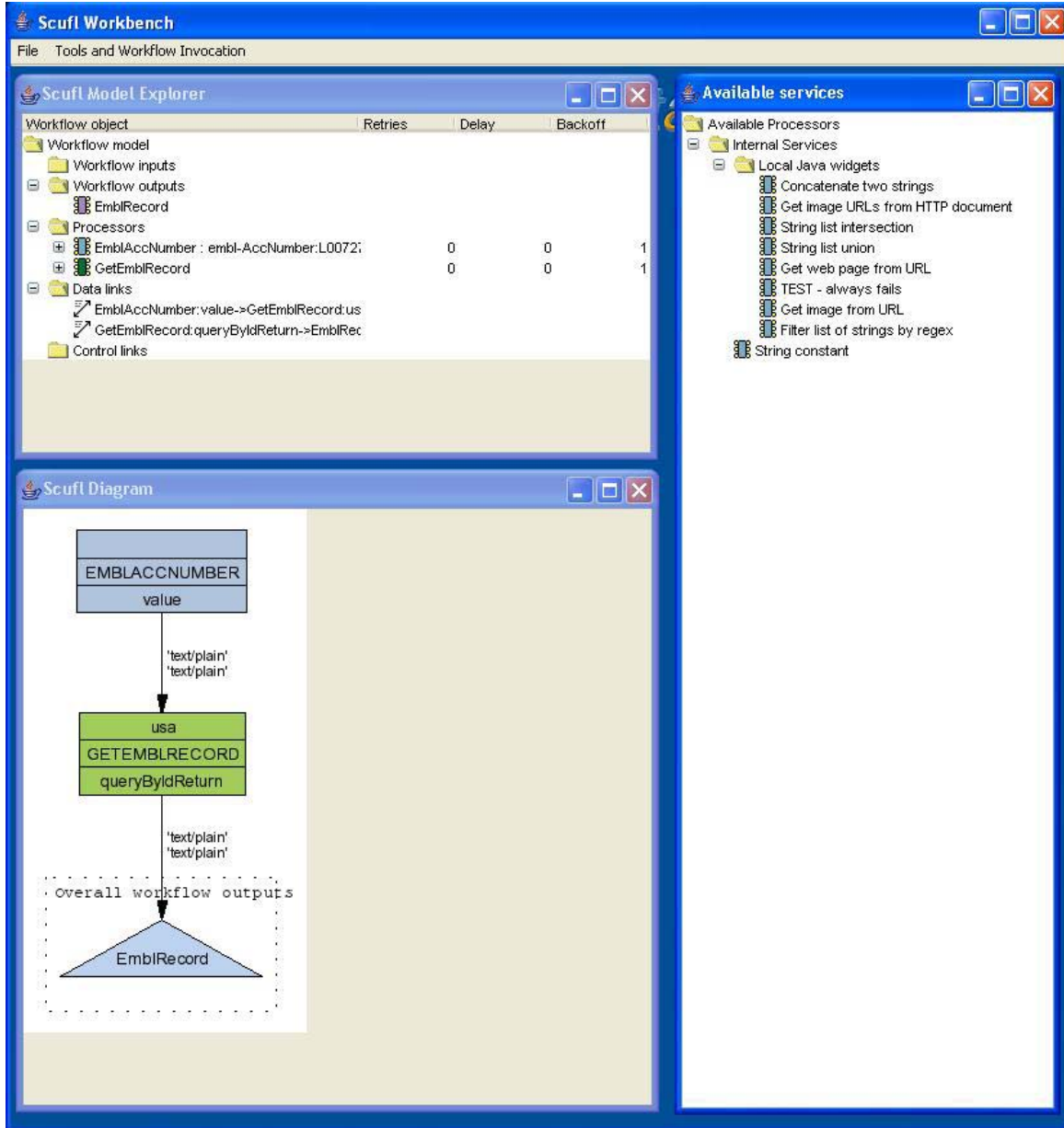


Figure 6.5.2: The Scufi Workbench [Taverna04]

The **Scufi Diagram** provides a graphical view of the current Scufi workflow model, which is a read-only view created from Taverna’s internal model. The Scufi Model Explorer window provides a view of the state of the current model as a tree structure and is also used to edit

workflows in the workbench. The **Available Services** window provides a palette of services and workflows using which the user can create workflows.

The enactor launch panel is used to enact workflows. On clicking “Run Workflow”, the workflow executes and invokes an Enactor Invocation window which consists of the **Status**, **Results** and **Process** report windows. The Status pane provides information about the status of each processor in the workflow, and the Results pane shows the actual results of the workflow. The Process report pane shows the provenance information.

The latest version of Taverna includes fault tolerance features that allow users to create more robust workflows. Specifically, it allows

- Retry processor instances in case events fail and configurable back off delays between retries and,
- Specification by the users of alternate processors when retries have been exceeded and are not applicable anymore.

6.5.2 Some Usability Issues in Taverna

- The ScufI workbench at this time does not support conditional execution of a workflow or a subworkflow based on any predicates. Since the workbench does not support AND, OR, XOR, etc., the user is limited to constructing and executing linear workflows.
- The ScufI workbench does not allow the user to create or edit a workflow graphically. To create a new workflow or edit an already existing workflow, the user has to use the ScufI

Model explorer. Thus, the user has to essentially click one of the many choices or “type-in” the workflow and its components.

- When the workflow executes, various output windows show intermediate results and current status of the workflow which help the user track the flow of data in the workflow. However, graphically, the workflow “diagram” doesn’t change to show the user which service is executing currently.
- Because the workflow diagram, as mentioned above, is not interactive with the user, the user has no way to set breakpoints in the workflow, or pause and resume the workflow.
- The error messages displayed when enactors or workflows fail are cryptic and do not help the user understand the problems in the system.
- While the Scufi Workbench allows the user to view the output data in different formats such as tables, text, etc., the user does not have any way of visualizing data in more scientific or domain-specific formats.

6.6 A Comparison

In this section, we compare the SWMS tools described above (including the SDM SPA system) based on the heuristics defined in Section 4.2. Each row considers a specific heuristic given in Section 4.2, whereas each column identifies a given tool. For each cell, “Y” indicates a “Yes”, meaning the tool includes the feature satisfactorily, a “N” indicates a “No” which means the tool does not include that feature at all, and an “S” indicates “Somewhat” which means that the tool includes some aspects of that feature. A blank means that the particular feature has either not been investigated yet or information available is either insufficient or ambiguous for evaluation purposes.

Although evaluations of these systems against the heuristics have been performed by the author of this document herself, these conclusions are supplemented by [PCS04] and [SSYA04] evaluations as discussed in Section 6.1.4.

Table 6.6.1: Comparison Table (Y = yes, S = Some, N = No)

Heuristic Number/Tool	Pilot system	Ptolemy II based SPA	SCIRun	Enhydra JaWE	ObjectWeb Bonita	Taverna
1 Construction and Execution	Y	Y	Y	Y	Y	S
2 Match between system and Real World	Y	Y	Y	Y	Y	Y
3 Ease of Use	Y	Y	S	S	Y	S
4 Flexibility and efficiency of use	Y	Y	Y	S	S	S
5 Consistency and Standards	S	Y	S	Y	Y	Y
6 Recognition rather than recall	Y	Y	Y	Y	Y	Y
7 Aesthetic and minimalistic Design	Y	Y	S	Y	S	Y
8 Employing dialog to resolve key uncertainties	Y	Y	S	Y	S	Y
9 Interaction with the end-user and visibility of system status	Y	S	S	S	Y	Y
10 Allowing efficient direct	Y	Y	Y	Y	Y	Y

invocation and termination						
11 Interaction with a Central Repository	Y	Y	S		Y	Y
12 Data-Mapping	S	Y	Y	Y		Y
13 Decision-based execution	S	S	N	Y	N	Y
14 Visualization of output data	S	Y	Y	S	S	Y
15 Reusability of Workflows	Y	Y	Y	Y	Y	Y
16 Verification, Validation and Fault-Tolerance	S	S	S	Y	S	Y
17 Minimizing the cost of poor guesses about action and timing	Y	Y	Y	Y	Y	Y
18 Help and Documentation	Y	Y	Y	Y	Y	Y
19 Security and Protection	S	Y		N	N	
20 Web-based GUI	N	Y	N	N	Y	N
21 Interoperability	S	S	N	S	S	N

6.6.1 Discussion

All the systems considered here allow the end-users to create, execute, save, and retrieve workflows satisfactorily. These systems provide the user with sufficient help and documentation.

Although simple design and ease of use are primary goals of these systems, SCIRun, JaWE and Taverna appear to be a little more difficult to learn and get accustomed to than the other systems we considered. Both the SDMSWE Pilot system and SCIRun do not use standard languages or formats to represent workflows and data, and thus may have limited extensibility. Other systems like SPA, JaWE and Bonita conform to XML-based formats and are easier to interconnect with other systems.

SPA, SCIRun and JaWE do not appear to communicate with the end-user efficiently during the execution of a workflow. This increases the possibility of confusing a novice user. Although all systems except our pilot system do not allow the end user to interact with a service registry (like UDDI), they all allow the users to access stored workflows and services in some way. All these systems also allow the user to transform data as it passes through the various steps of the workflow, but the transformations are very basic. All these systems need to add functionality that allows transformation of complex and domain-specific data.

The pilot system, JaWE and Bonita do not have visualization support specific to the life sciences' domain. JaWE and Taverna allow the user to create conditional and decision-based workflows, however the user does not have a lot of flexibility while defining conditions. SPA and Bonita allow web-based invocations of the software.

We observe that Verification, Validation and Fault-Tolerance remains in issue in the investigated systems. Although JaWE has better Verification and Validation features as compared to the other systems, Fault-tolerance is still an issue. If a service fails while the execution of a workflow in any of these systems, no backup services are available to continue the execution of the workflow and the workflow execution terminates. Taverna has some fault-tolerance implemented, but verification and validation is an issue. A similar argument can be made about Interoperability. It is easier to make systems like JaWE and Bonita interoperate with other systems since they conform to industry standards for representing data. On the other hand, systems like SCIRun and

Taverna may have more interoperability as they employ formats that are either not recognized by the industry, or are not popularly used.

7 Conclusion

A workflow management system (WFMS) is a system that allows a user to create executable workflows. Resources may be network-based. A Scientific WFMS is a system of this kind that caters to the scientific research domain.

This work proposes an architecture for a SWFMS. We discuss tools and technologies that can be leveraged to achieve our goal of creating a user-friendly workflow system operating in a collaborative environment. This thesis studies user interfaces and design of the UI for such systems. We explain why we think the careful design and review of user interfaces is important, and discuss the user interfaces of a few Scientific WFMS's. We point out usability problems with such systems to help the reader understand commonly made mistakes by user interface designers and shortcomings of the visual components of extremely complex systems.

7.1 Summary

Scientific research and experiments are conducted in a trial and error manner. Typically, a scientist creates a simulation sequence using a number of data analysis and transformation steps. If not satisfied with the results, the scientist usually makes numerous changes to such steps of his experiments and resubmits the simulation. Such a simulation may execute in a local or remote environment. Thus, a requirement is created for a solution that allows the scientist to create such experimental flows (called workflows) using network-based data and tools.

Existing solutions to such a problem focus on business workflows. Workflows in the life science domain are much different than workflows in the business world as they are more dynamic and evolving in nature, and involve the use and manipulation of highly structured and heterogeneous data. Unlike business workflows, the science workflows are used by end-users (scientists) who are not experts at building the workflow system using software and network resources.

In this study, we proposed an architecture that enables the scientist to create such semi-automated workflows using reusable components across a network. We call these components Services. Such a system is referred to as a Scientific Workflow Management System (SWFMS).

At the start of this study, we explained in detail a Scientific Workflow Management system, its functionality and impact. We introduced an architecture for such a system, and the basic requirements it should satisfy in order to be considered usable. We defined components that form an integral part of such an architecture.

We observe that the visual component of a Scientific WFMS is of utmost importance, since based on its ease-of-use and intuitivity, scientists will be able to achieve their goals via experiments. By definition, such systems are highly complex in nature as they deal with heterogeneous data and tools, and their interactions. As complexity of the underlying architecture increases, the number of widgets in the user interface increases exponentially, hence making the system difficult to use.

In our study we introduced the basic concepts of Human Computer Interaction and related fields of study. We discussed how HCI and other disciplines interrelate, and the impact of such an interrelation on the design of User Interfaces. We explained in detail the significance of the Visual Component of a complex system, and we introduced the user to various practices and approaches towards design of interactive user interfaces. We elaborated on the choice of User-Centered Design [11] as our design methodology and explained how it helped towards an iterative design process to maximize usability of a WFMS.

We next introduced the user to the concept of Heuristic Evaluation and its significance as an evaluation methodology while creating and designing user interfaces in an iterative manner. In Section 4.2 we defined more than 20 heuristics based on which user-interfaces for Scientific WFMS's can be evaluated, and can also be used as a guide to modify or create a new interface. These heuristics were adapted to the generic heuristics for user interfaces defined by Nielsen in [Nie92].

We introduced the reader to our prototype system, the SDMSWE, and explained how it closely manifested both the developer and the end user's view of an ideal workflow system. Although the SDMSWE system was developed in a manner such that it would incrementally lead to our final system after several iterations, we decided to build a new system based on the Ptolemy II architecture for reasons mentioned in Section 5. Thus, the SDMSWE Pilot system was used as a "throw-away" prototype. It helped us understand the requirements of such a system, and a feasible approach towards building it.

We then discussed a few existing workflow systems that could be used in the life sciences domain. Amongst others, we discussed SDM's Ptolemy II based SPA system along with its shortcomings at the time of publication of this work. For other systems considered as well, we discussed usability problems that the systems had.

We did a comparative analysis of all these systems studied, and discussed problems with the usability of the systems. We proposed areas that needed more work in order to make these systems more usable. We realized that complexity of the user interface was an issue, and needed special emphasis. We also observed that features such as decision-based execution, fault-tolerance, visualization and interoperability posed significant challenges to the acceptance of these systems.

We hope that such an evaluative discussion of existing systems helps the reader understand the difficulty a developer faces while designing user interfaces for complex systems, and how shortcomings in the design of visual components of such a system may impact the effectiveness of the system as a whole as perceived by the end user. Alternatively, such a discussion also helps the User Interface designers avoid commonly made mistakes while developing such a system, and create a more usable system.

7.1.1 Contribution

We discussed several approaches towards design of user interfaces of scientific workflow management systems, which could also be applied towards design of other similar complex and interactive systems. We recommended an evaluation techniques that can help assess scientific

workflow systems in a manner that is easy, efficient and cheap. To this end we defined a set of heuristic metrics. These heuristics define a set of necessary requirements that an end-user interface of a scientific workflow system should meet. Metrics have been defined taking into account HCI research, discussions with researchers and workflow experts over the past couple of years, and feedback from end-users.

7.2 Future Work

Our study explains in great detail the impact of the design of the User Interface on the usability of the system as a whole. As seen from our evaluative discussions in previous sections, much work still needs to be done as far as creating User Interfaces to be simple goes. Most interfaces get extremely complex as more and more functionality is added to the underlying layers.

We observe that interoperability with third party tools is a challenge faced by many developers. Although possible up to an extent, interoperability remains a topic of research since most end users find it extremely difficult to get their existing systems interoperate with other tools due to shortcomings in the User Interfaces, and the absence of industry-accepted standards. Visualization of data is also a challenging aspect of interface design since most visualization tools and techniques are domain specific, whereas data used and produced in workflow systems and the systems themselves are intended to be generic in nature. Verification, Validation and Fault-tolerance also continue to be topic of interests in the context of user interface design as they also increase the complexity of the system, and decrease the end user's ability to use such features while creating workflows with ease. Interactivity with the user is of utmost importance for a system, and more tools should seek to include such features up to a greater extent.

We hope that future extensions of this study will focus on the above-mentioned challenging issues in the design of User Interfaces for Scientific Workflow systems. We hope to define in the near future the exact set of primitives and widgets that implement complex features in the system, yet at the same time keep the overall design simple and minimalistic. Since we strongly believe in the concept of teaching by example, we hope to implement many such features in our SPA system effectively and explain in detail our approach and recommendations.

References

- [ABB+03] 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>
- [ABHK00] W.M.P VAN DER Aalst, A. P. Barros, A.H.M. ter Hofstede, B. Kiepuszweski, “Advance Workflow Patterns”, 7th International Conference on Cooperative Information Systems, volume 1901 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2000, p.18.
<http://tmitwww.tm.tue.nl/research/patterns/download/coopis/pdf>
- [All01] Rob Allen, “Workflow: An Introduction”, Open Image Systems Inc., 2001.
http://www.wfmc.org/information/Workflow-An_Introduction.pdf
- [Bak02] Kevin Baker, “Heuristic Evaluation of Groupware”, MSc thesis, Uniiversity of Calgary, 2002.
- [BGG02] Kevin Baker, Saul Greenberg, Carl Gutwin “Empirical development of a heuristic evaluation methodology for shared workspace groupware”, Proceedings of the ACM Conference on Computer Supported Cooperative Work, ACM Press, 2002, p 96-105.
- [BIRN04] The Biomedical Informatics Research Network, 2004.
<http://nbirn.net/>
- [BLAST04] NCBI BLAST, 2004.
<http://www.ncbi.nlm.nih.gov/Education/BLASTinfo/information3.html>
- [Boe94] Peter Boersma, “Experimental Research into Usability and Organizational Impact of Workflow Software”, Master’s Thesis, University of Twente, 1994.
- [Bonita04] The ObjectWeb Bonita Website, 2004.
<http://bonita.objectweb.org/>
- [BR99] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, “Modern Information Retrieval”, Addison Wesley Longman Publishing, 1999.
<http://www.sims.berkeley.edu/~hearst/irbook/index.html#contents>
- [Bro03] Lecture Notes for HCI. <http://wwwcms.brookes.ac.uk/modules/p08794/lectures.htm>, Oxford Brookes University, 2003.
- [Cer02]Ethan Cerami, “Web Services for Bioinformatics”, 2002.

<http://webservices.xml.com/pub/a/ws/2002/05/14/biows.html>

[Cha02] Sandeep Chandra, "Service-Based Support for Scientific Workflows", M.S. Thesis, NC State University, 2002.

[Clus04] Clusfavor HomePage, 2004.
<http://mber.bcm.tmc.edu/genepi/clusfavor.html>

[CompList04] List and Comparison of Open Source Workflow Engines, 2004.
<http://wiki.apache.org/cocoon/WorkflowImplementationComparison>

[DADS05] Dictionary of Algorithms and Data Structures, 2005.
<http://www.nist.gov/dads/>

[Dav91] David J. Michael, "The Optimal Representation of Activity Networks as Directed Acyclic Graphs", PhD. Thesis, NC State University, May 1991.

[Delphi] The Delphi Method.
<http://www.iit.edu/~it/delphi.html>

[DFAB97] Alan Dix, Janet Finlay, Gregory Abowd, Russell Beale, "Human-Computer Interaction", Prentice Hall, 1997.

[DOE04] The Office of Science Data-Management Challenge: Report from the DOE Office of Science Data-Management Workshops, March–May 2004.

[ECOO04] ECOO: Environment for Cooperation (Project Team Website), 2004.
<http://www.inria.fr/recherche/equipes/ecoo.en.html>

[Elm64] S.E. Elmaghraby, "An algebra for the analysis of generalized activity networks," Management Sci. 10, 494-514, 1964.

[Elm66] S.E. Elmaghraby, "On generalized activity networks," J. Ind. Eng., Vol. 17, 621-631, 1966.

[Elm95] Elmaghraby S.E., Baxter E.I., and Vouk M.A., "An Approach to the Modeling and Analysis of Software Production Processes," Intl. Trans. Operational Res., Vol. 2 (1), pp. 117-135, 1995.

[Free04] The Freefluo Website, 2004.
<http://freefluo.sourceforge.net/>

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

- [GHR94] Stratis Gallopoulos, Elias Houstis and John Rice, "Computer as Thinker/Doer: Problem-Solving Environments for Computational Science", IEEE Computational Science Engineering Magazine, 1(2):11-23, 1994.
- [Gre98] Irene Greif, "Computer-Supported Cooperative Work: A Book of Readings", Morgan Kaufmann, 1998.
- [Gru89] Jonathan Grudin, "Why CSCW applications fail: Problems in the design and evaluation of organization of organizational interfaces", Office: Technology and People, 4(3):245-264, 1989.
- [GWS+02] Mark Greenwood, Chris Wroe, Robert Stevens, Carole Goble, Matthew Addis, "Are bioinformatics doing e-Business?", The Victoria University of Manchester, 2002.
http://www.cs.man.ac.uk/~markg/mygrid/eWIC_greenwood2.htm
- [Har01] Mark Van Harmelen, "Object Modeling and User Interface Design: Designing Interactive Systems", Addison Wesley Professional, 2001.
- [HM93] William E. Hefley, Dianne Murray, "Intelligent User Interfaces", Proceedings of 1993 International Workshop on Intelligent User Interfaces, 1993, p 3-10, Orlando, FL.
- [Hor99] Eric Horvitz, "Principles of Mixed-Initiative User Interfaces", Proceedings of CHI '99, ACM SIGCHI Conference on Human Factors in Computing Systems, May 1999, p. 159-166, ACM Press, Pittsburgh, PA.
- [JaWE04] The Enhydra JaWE Website, 2004.
<http://jawe.objectweb.org/>
- [Joh96] Brad Johns, "Experiences Implementing Workflow Based Systems in Manufacturing Enterprise Product Development and Change Processes", IBM Corp., 1996.
<http://lsdis.cs.uga.edu/activities/NSF-workflow/bjohns.html>
- [Keo00] David Keolle, "Intelligent User Interfaces, An Independent Study Project", 2000.
<http://www.cs.wpi.edu/Research/airg/IntInt/intint-outline.html>
- [Kep04a] The Kepler Project website, 2004.
<http://kepler.ecoinformatics.org/>
- [Kep04b] The Kepler Project Workflow Examples, 2004.
<http://kepler-project.org/Wiki.jsp?page=WorkflowExamples>
- [Kie02] Bartosz Kiepuszweski, "Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows", PhD. Thesis, 2002.

- [LAGM03] Bertram Ludaescher, Ilkay Altintas, Amarnath Gupta, Reagan Moore, “SDMC Report to the U.S. Department of Energy – Mediation Technologies for Scientific Workflows”, A Report to the SciDAC Program, 2003.
<http://kbi.sdsc.edu/SciDAC-SDM/doe-report-2003.pdf>
- [Link04] The Link-up Project, 2004.
<http://www.mygrid.org.uk/linkup/>
- [McE02] Gregor McEwan, “Heuristic Evaluation of Groupware”, 2002.
<http://pages.cpsc.ucalgary.ca/~mcewan/HEG.html>
- [Nie05] Jacob Nielsen’s Heuristic Evaluation Website, 2005.
<http://www.useit.com/papers/heuristic/>
- [Nie92] Jacob Nielsen, “Finding Usability Problems through Heuristic Evaluation”, Proceedings of the ACM Conference on Human Factors in Computer Science (CHI '92), 1992, 373-380.
- [NM94] Jacob Nielsen, Robert L. Mack, “Usability Inspection Methods”, John Wiley and Sons, 1994.
- [Nor] Sameer Merchant, “Norman's Philosophy of Design for Everyday Interaction”, Georgia Tech.
http://www.cc.gatech.edu/classes/cs6751_97_fall/projects/ms-squared/ubicomp/sam_essay.html
- [Nor90] Donald Norman, “The Design of Everyday Things”, Currency, Reissue Edition, 1990.
- [OfBiz04] The Open for Business Project.
<http://www.ofbiz.org/>
- [Oin04] Tom Oinn, “Comparing Workflow in eScience and eBusiness”, 2004.
<http://twiki.mygrid.org.uk/twiki/bin/view/Mygrid/WorkFlowDifferences>
- [ONG96] Yean Wei ONG, “Visual Complexity in Graphical User Interfaces”, MSc Thesis, The University of Western Australia, 1996.
- [PCS04] Pankaj Chopra, Charles Loftis, Spencer Proffit, “Internal Project Report: Workflows in Bioinformatics”, NC State University, Spring 2004.
- [Pto04] Ptolemy II Website, 2004.
<http://ptolemy.eecs.berkeley.edu/ptolemyII/>
- [QPR04b] Scientific Process Automation Data Integration quarterly report, The SDM Project, 2nd quarter, 2004.
- [Ram99] Magnus Ramage, “Evaluation of Cooperative Systems Project”, PhD. Thesis, Lancaster University, 1999.

[Rivet04] Visualizing Complex Systems, The RIVET Project, 2004.
<http://graphics.stanford.edu/projects/rivet/>

[Sch86] Ben Schneiderman, “Designing the User Interface – Strategies for Effective Human-Computer Interaction”, Addison-Wesley, 1986.

[Sci04] The SciDAC Program Website, 2004.
<http://www.scidac.org/>

[SCIRun04] The SCIRun Website, 2004.
<http://software.sci.utah.edu/scirun.html>

[SEEK04] The Science Environment for Ecological Knowledge Project, 2004.
<http://seek.ecoinformatics.org/>

[Shark04] The Enhydra Shark Website, 2004.
<http://shark.objectweb.org/>

[SI05] The StrikeIron Project, 2005.
<http://strikeiron.com/>

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

[SSYA04] Kevin Snow, Walter Scheper, Eric Yarborough, Joel Allen, “Scientific Process Automation: Workflows and Web Services”, Internal Report, NC State University, Fall 2004.

[SV96] 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.

[Talin98] Talin, “A Summary of Principles for User-Interface Design”, 1998.
http://www.sylvantech.com/~talin/projects/ui_design.html

[Taverna04] The Taverna Website, 2004.
<http://taverna.sourceforge.net/>

[Trans04] The Transfac Program, 2004.
<http://transfac.gbf.de/>

[Triana04] The Triana Website, 2004.
<http://www.triana.co.uk/>

[UDDI04] The Universal Description, Discovery and Integration Website, 2004.
<http://www.uddi.org/>

[UIPatterns04] The UI Patterns and Techniques Website, 2004.
<http://time-tripper.com/uipatterns/index.php>

[UNI04] The UNICOREPro Website, 2004.
<http://www.pallas.com/e/products/unicore-pro/index.htm>

[Usa04] The Usability First Website, 2004.
<http://www.usabilityfirst.com/>

[VIR01] Karel Vredenburg, Scott Isensee, Carol Righi, “User-Centered Design: An Integrated Approach”, Prentice Hall, 2001.

[VisPortal04] VisPortal: Increasing Scientific Productivity by Simplifying Access to and Use of Remote Computational Resources, 2004.
<http://www-vis.lbl.gov/Publications/2004/LBNL-PUB-893-Visportal.pdf>

[VS96] J. Vetter and K. Schwan, “Models for computational steering”, Proceedings, Third International Conference on Configurable Distributed Systems, p. 100-7, Annapolis, MD, USA, May 1996.

[WASA] The WASA Website.
<http://dbms.uni-muenster.de/menu.php3?item=projects&page='wasa/index.php3?id=1'>

[WfMC] The Workflow Management Coalition.
<http://www.wfmc.org/>

[Wino97] Terry Winograd, “From Computing Machinery to Interaction Design” Beyond Calculation: The Next Fifty Years of Computing, Springer-Verlag, 149-162, 1997,
<http://pcd.stanford.edu/winograd/acm97.html>

[WP04] The Workflow Patterns website, 2004.
<http://tmitwww.tm.tue.nl/research/patterns/>

[WSAD04] The WebSphere WSAD Website, 2004.
<http://www-306.ibm.com/software/awdtools/studioappdev/>

[WWV] Jacques Wainer, Mathias Weske, Gottfried Vossen, Claudia M Bauzer Medeiros, “Scientific Workflow Systems”.
<http://lsdis.cs.uga.edu/activities/NSF-workflow/wainer.html>

[XPDL02] The WfMC Workflow Standard Specification: XPDL, 2002.
http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf

[Xscufl04] The Xscufl Language Reference, 2004.
<http://taverna.sourceforge.net/api/org/embl/ebi/escience/scufl/XScufl.html>

[XWRAP00] The XWRAP System, 2000.
<http://www.cc.gatech.edu/projects/disl/XWRAPelite/>

[Zad99] MohanKumar C. Zade, “User Interface”, 1999.
<http://www.cse.iitk.ac.in/research/mtech1997/9711126/node7.html>

[ZLR96] Wayne Zachary, Jean-Christophe Le Mentec, Joan Ryder, “Human Interaction with Complex Systems”, C.Ntuen & E.Park (Eds.), Conceptual Principles and Design Practice, Kluwer Academic Publishers, p 35-52, 1996.
http://downloads.chiinc.com/PDFs/Complex_Systems_Paper.pdf

[ZOO] Yannis Ioannidis and Miron Livny, “Desktop Scientific Experiment Management and the ZOO Project”.
<http://deslab.mit.edu/DesignLab/dicpm/position/ioannidis.html>

Appendix

I Promoter Identification Workflow

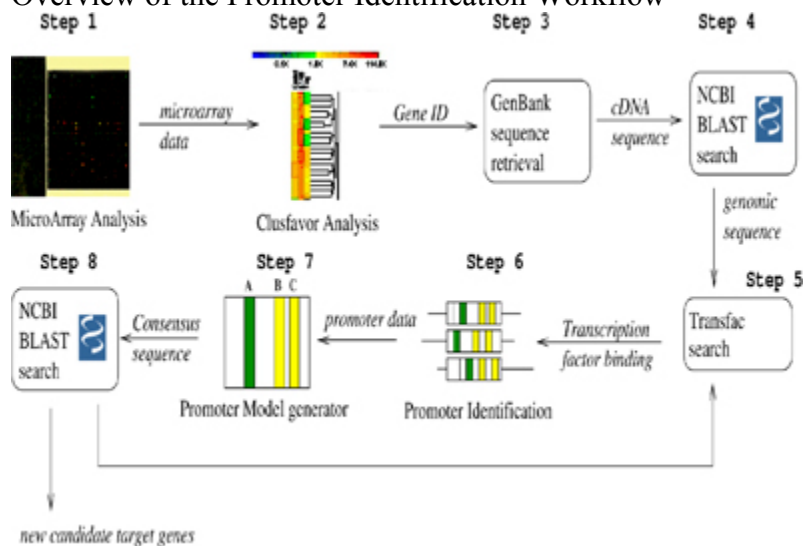
(Composed by Xiaowen Xin, Lawrence Livermore National Labs: <https://www-casc.llnl.gov/sdm/documentation/casestudy-piw.php>)

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 fluorescence 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.

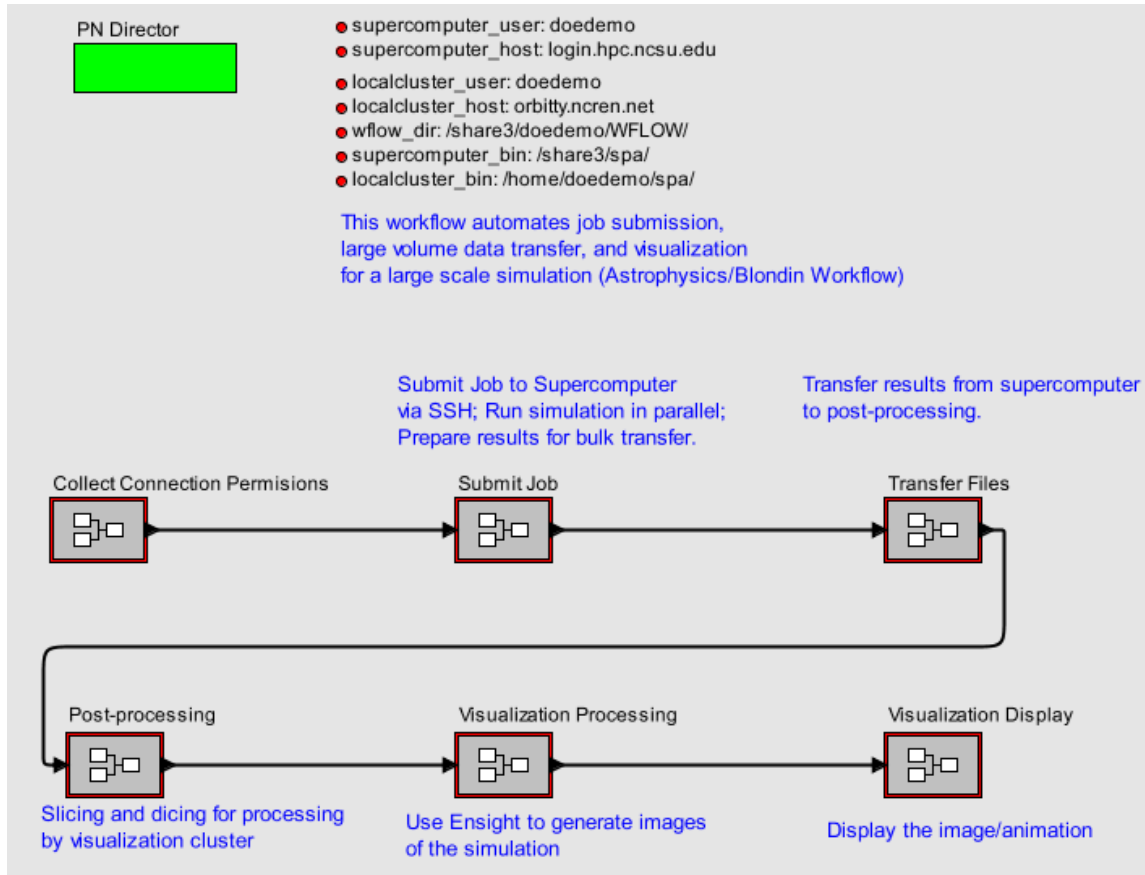
Finding New Genes

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.

II TSI Workflow

(Composed by Zhengang Cheng, NC State University)



The TSI Workflow is for Dr. Blondin (astrophysicist at NC State) to automate his daily execution in Supernova Simulation. The simulation will need to compute on a Supercomputer at ORNL, the result files then will be transferred to a local cluster at NC State, after post-processing, the visualization tool Enight software will generate the video simulation of Supernova explosion.

The Above Workflow works in the following steps.

Collect Connection Permissions: This step is designed to collection user authentication information at the start of workflow execution, so the user does not need to wait during execution.

Submit Job: It will kick start the simulation on supercomputer at ORNL using the information collected in step 1.

Transfer File: The results from supercomputer are transferred back to local clusters at NC State. We can use transfer tools like SABUL or BSCP to do the transfer.

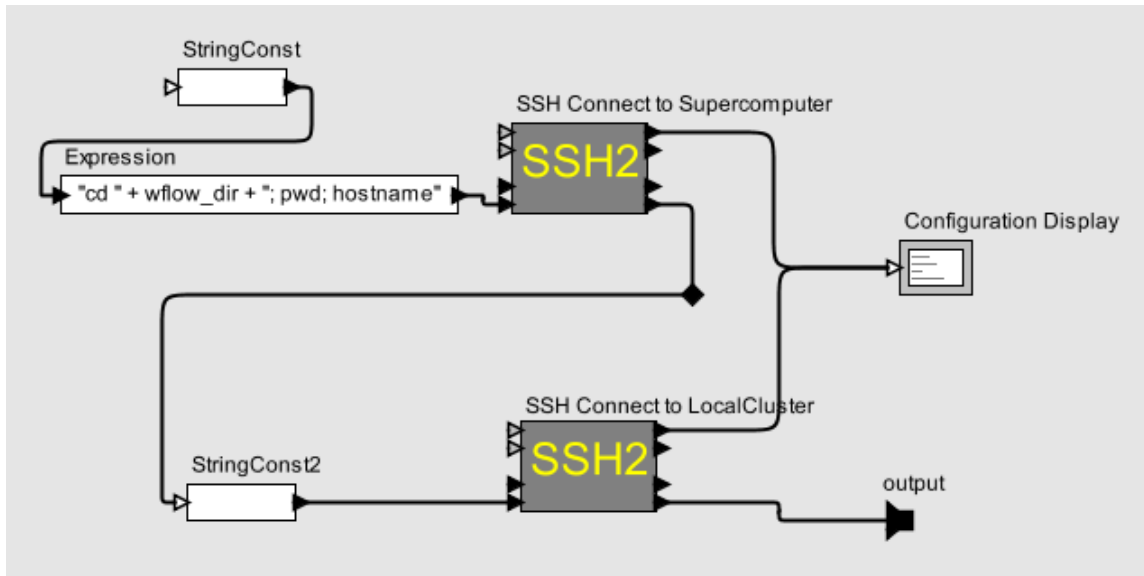
Post Processing: The transferred files will need post-processing like slicing and dicing and then be distributed to 21 nodes in the local cluster.

Visualization Processing: The scientist will start the Ensignt software to do the video processing. The result will be either an image or a video.

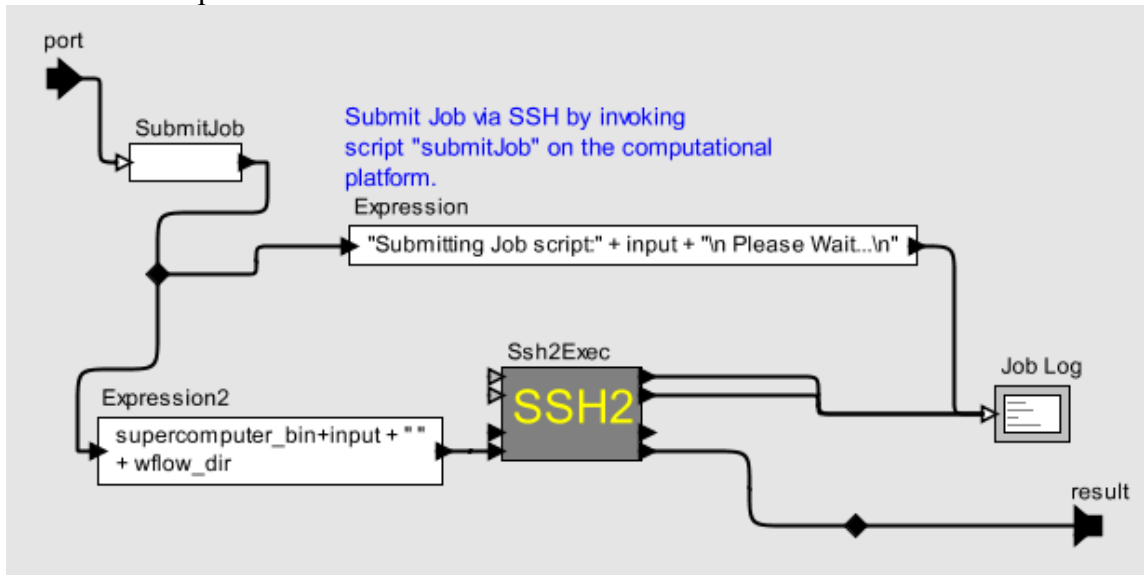
Visualization Display: This step will show the video/image result in Web browser.

Each steps in above is a sub-workflow by itself. The internals of the each step is as following.

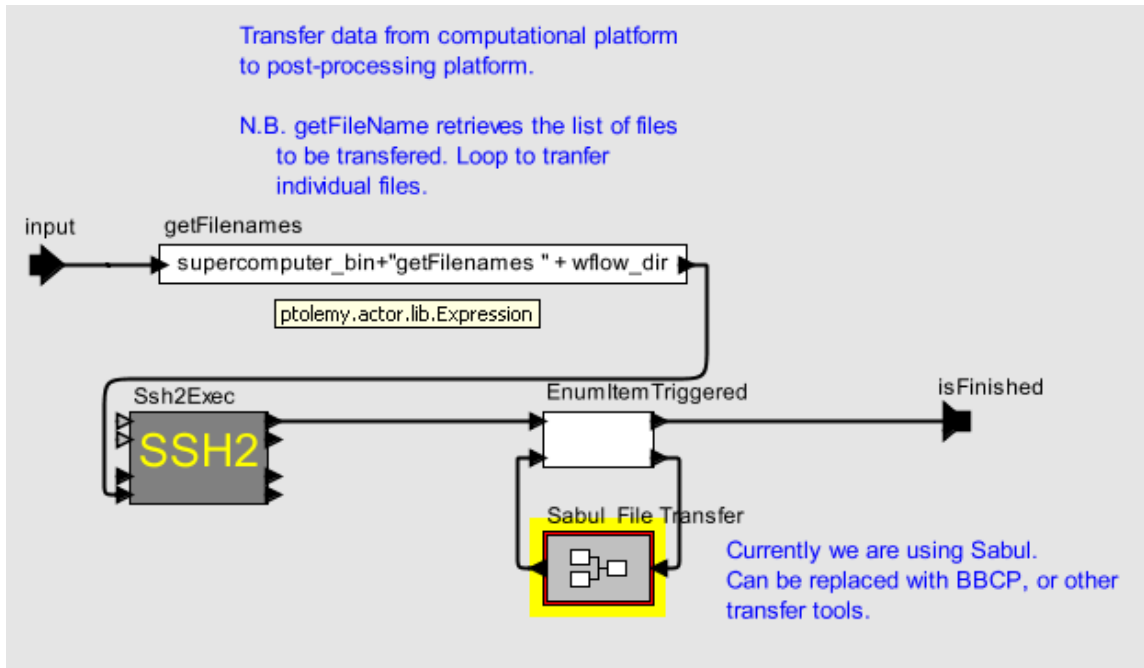
Collect Connection Permissions: This step is designed to collect user authentication information at the start of workflow execution, so the user does not need to wait during execution.



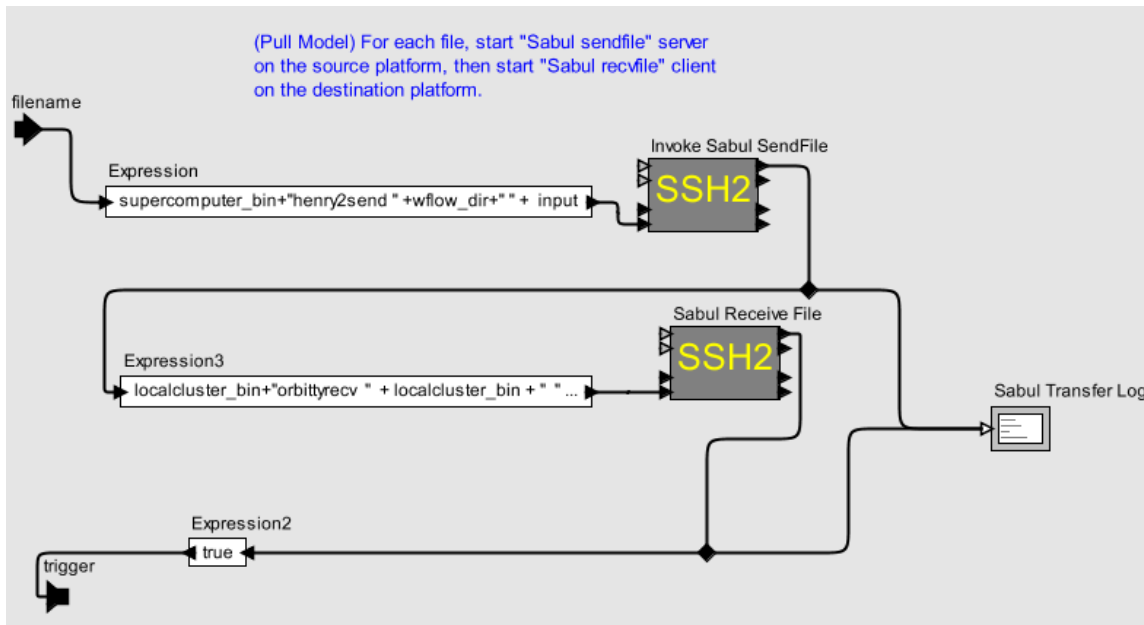
Submit Job: This step will kick start the simulation on supercomputer at ORNL using the info collected in step 1.



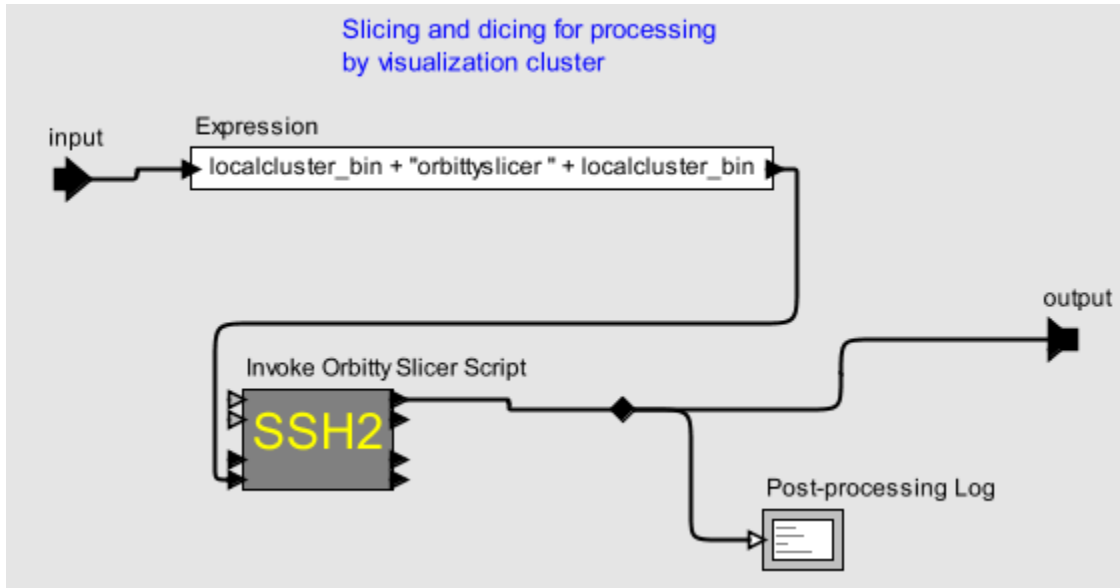
Transfer File: The results from supercomputer will be transferred back to local clusters at NC State. We can use transfer tools like Sabul or BSCP to do the transfer.



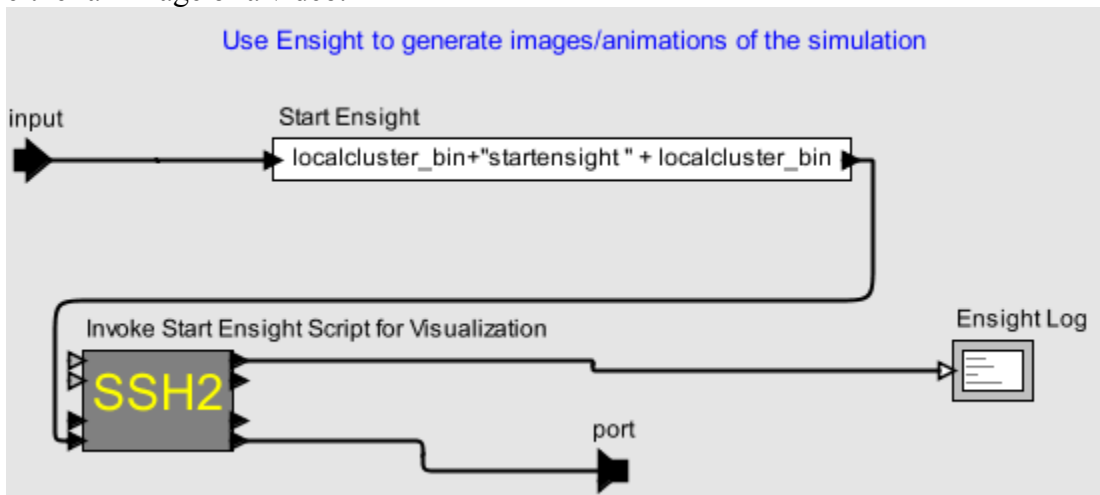
Sabul File Transfer:



Post Processing: The transferred files will need post-processing like slicing and dicing and will then be distributed to 21 nodes in the local cluster.



Visualization Processing: Start of the Ensignt software for video processing. The result will be either an image or a video.



Visualization Display: To show the video/image result in Web browser.

