

IMPROVING ACCESS TO SPACE WEATHER DATA VIA WORKFLOW  
AND WEB SERVICES

by

ANU SWAPNA SUNDARAVEL  
B.Tech., Anna University, India, 2007

A thesis submitted to the  
Faculty of the Graduate School of the  
University of Colorado in partial fulfillment  
of the requirement for the degree of  
Master of Science  
Department of Computer Science

2010

This thesis entitled:  
Improving Access to Space Weather Data via Workflow and Web Services  
written by Anu Swapna Sundaravel  
has been approved for the Department of Computer Science

---

(Prof. Kenneth Anderson)

---

(Dr. Eric Kihn)

Date\_\_\_\_\_

The final copy of this thesis has been examined by the signatories, and we  
find that both the content and the form meet acceptable presentation standards  
of scholarly work in the above mentioned discipline

Sundaravel, Anu Swapna (M.S., Computer Science, Department of Computer Science)

Improving Access to Space Weather Data via Workflow and Web Services

Thesis directed by Associate Professor Kenneth M. Anderson

The Space Physics Interactive Data Resource (SPIDR) is a web-based interactive tool developed by NOAA's National Geophysical Data Center to provide access to historical space physics datasets. These data sets are widely used by physicists for space weather modeling and predictions. Built on a distributed network of databases and application servers, SPIDR offers services in two ways: via a web page interface and via a web service interface.

SPIDR exposes several SOAP-based web services that client applications implement to connect to a number of data sources for data download and processing. At present, the usage of the web services has been difficult, adding unnecessary complexity to client applications and inconvenience to the scientists who want to use these datasets.

The purpose of this study focuses on improving SPIDR's web interface to better support data access, integration and display. This is accomplished in two ways: (1) examining the needs of scientists to better understand what web services they require to better access and process these datasets and (2) developing a client application to support SPIDR's SOAP-based services using the Kepler scientific workflow system.

To this end, we identified, designed and developed several web services for filtering the existing datasets and created several Kepler workflows to automate routine tasks associated with these datasets. These workflows are a part of the custom NGDC build of the Kepler tool. Scientists are already familiar with Kepler due to its extensive use in this domain. As a result, this approach provides them with tools that are less daunting than raw web services and ultimately more useful and customizable. We evaluated our work by interviewing various scientists who make use of SPIDR and having them use the developed Kepler workflows while recording their feedback and suggestions. Our work has improved SPIDR such that new web services are now available and scientists have access to a desktop-based direct manipulation tool that provides them with improved support for data access and visualization.

## DEDICATION

I would like to dedicate this dissertation to my family, especially to my husband for his motivation and support throughout the course of this thesis work.

## ACKNOWLEDEEMENT

I would like to express my sincere gratitude to my advisor, Prof. Kenneth Anderson, for his encouragement, support and patience from the start of this thesis work. His immense knowledge and enthusiasm for research motivated me through each step of my research study. I also thank him for his guidance in the writing of this work. I could not have imagined having a better advisor for my Master's thesis.

I would like to thank my co-advisor, Dr. Eric Kihn for offering me the opportunity to take up this work as my Master's thesis. His passion for science and research has been an inspiration to me always.

I am indebted to both of my advisors for their motivation and support, without which this thesis would not have been completed.

I also thank Prof. Clarence Ellis for being a member of my thesis committee and for his insightful comments.

I would also like to thank my colleagues, Peter Elespuru and Rob Redmon for evaluating my work and providing me with feedback.

## CONTENTS

## CHAPTER

I.	INTRODUCTION.....	1
II.	BACKGROUND TECHNOLOGIES.....	4
	2.1 Web Services.....	4
	2.2 Kepler – Scientific Workflow system.....	10
	2.3 soapUI Testing tool.....	16
III.	APPROACH.....	17
	3.1 SPIDR web page interface.....	17
	3.2 Existing SOAP-based services.....	18
	3.3 Problem Domain.....	20
	3.4 Proposed solution.....	21
IV.	METHODS.....	24
	4.1 High/Low Threshold Filter Web Service.....	24
	4.2 Kepler-NGDC scientific workflows.....	27
V.	RESULTS.....	35
	5.1 Evaluation.....	35
	5.2 Discussion of results.....	35
VI.	RELATED WORK.....	38
	6.1 THEMIS Data Analysis Software.....	38
	6.2 Kepler/pPod.....	39
VII.	CONCLUSION.....	40
	BIBLIOGRAPHY.....	42
	APPENDIX .....	45

## TABLES

## Table

1. Available Directors in the Kepler component library.....	12
2. Command line parameters for SpidrClient.....	20

## FIGURES

## Figure

1. Web Services Architecture.....	5
2. Web Service Opeartion.....	6
3. Sample Ptolemy model structure.....	11
4. ArrayToElements actor.....	13
5. Example of Composite actor.....	13
6. Relation icon.....	14
7. Sample workflow demonstrating ‘relation’ in a workflow.....	14
8. Sample Kepler workflow.....	15
9. Composite actor.....	15
10. Output of the sample Kepler workflow.....	16
11. HiLoFilterService operation.....	24
12. Interactive workflow for SPIDR data downloads.....	27
13. GetInputData composite actor.....	28
14. Command prompt window.....	28
15. ProcessInputData composite actor.....	29
16. Error message displayed to the user.....	30
17. Interactive workflow for SPIDR data plotting.....	30
18. PreprocessData composite actor.....	31
19. Sequence Plot of SPIDR data.....	31
20. Example inputs for the interactive workflows.....	32
21. HiLoFilter workflow for SPIDR Geomagnetic data.....	33
22. MassageSPIDRIntoKepler composite actor.....	33
23. Plot of SPIDR filtered data.....	34
24. Sample future Kepler workflow.....	40
25. Output of the sample future Kepler workflow.....	41
26. ArrayAccumulator actor.....	45
27. ArrayElement actor.....	45
28. ArrayExtract actor.....	45



29. ArrayPlotter actor.....	46
30. ArrayToSequence actor.....	46
31. BooleanSwitch actor.....	46
32. ConcatenateArrays actor.....	46
33. InteractiveShell actor.....	47
34. LookupTable actor.....	47
35. Ramp actor.....	47
36. SampleDelay actor.....	48
37. SequencePlotter actor.....	48
38. SequenceToArray actor.....	48
39. StringMatches actor.....	48
40. StringReplace actor.....	49
41. StringSplitter actor.....	49
42. StringToFloat actor.....	49
43. URLToLocalFile actor.....	49
44. VariableSetter actor.....	50
45. WebService actor.....	50
46. Edit parameters for WebService actor.....	51

## CHAPTER I

### INTRODUCTION

Technological innovations in computing have dramatically transformed the way research is being carried on in the scientific community. The volume of scientific data is rapidly increasing each year with the precision of scientific instruments and this brings up the need for managing this data efficiently. The Scientific Data Management Center [22] has identified three major requirements for effectively managing and analyzing the scientific data and its related metadata information. The first requirement is to provide efficient access to the data storage system. The system should be able to handle requests of reading and writing large volumes of data without slowing down the simulation and analysis engines. Secondly, the scientists require technologies that provide them the ability to perform quick searches over large volumes of data and the use of data mining techniques that will help them to understand data better. Finally, they need tools to generate data, visualize it and perform further processing on the data to conduct their research. With many data management architectures being introduced to the community, grid [8] is the widely used technology to develop earth science systems that integrate large volume and heterogeneous data sets in a distributed environment. Grid applications provide a platform for managing and accessing the scientific data that are geographically distributed and also data from different institutions thus increasing the accessibility of the data sets to a researcher. This data is used for climate modeling, space weather predication, structural biology and ecological study. These kinds of modeling and study are best supported through scientific workflows. Scientific workflows are automated processes where data is passed from one component to another for processing and the result is a solution to a scientific problem. Grid computing supports scientific workflow technologies [10] to solve complex scientific problems. The advantages of using

scientific workflow systems include the ability to integrate distributed resources, monitoring the execution, sharing the workflows across the community and workflow reuse. Several scientific workflow management systems exist to create, support, manage and execute workflows on computing resources. Some of them include SCIRun [23], Taverna [31], Kepler [12] and Triana [34]. For a taxonomy of the available workflow management systems and their comparison, see [10]. A few of the projects that use workflows to support their research include:

- the Kepler/pPod project developed by a team at UC Davis that makes use of the Kepler workflow system to study phylogenetic analysis;
- the SANParks project which also makes use of Kepler for managing wildlife populations;
- and the European Model for Bioinformatics Research and Community Education (EMBRACE) project which integrates Taverna workflows for the study of bioinformatics.

My thesis work focuses on improving the access to the space weather data sets published by NOAA/NGDC [19] (National Oceanic and Atmospheric Administration/National Geophysical Data Center) via Space Physics Interactive Data Resource (SPIDR) [27; 39] by developing scientific workflows. SPIDR is built to provide online access to a large volume of space weather data sets and to assist in environmental research. It is a distributed network application developed using various open-source tools and currently archives solar, geomagnetic, ionospheric and several other types of data from ground observatories and satellites. The SPIDR components are mirrored at numerous locations that include the USA, Russia, Japan, India, China, Australia, South Africa, France and the Ukraine. This ensures fast data delivery and data visualization irrespective of a scientist's geographic location. The users of this data chiefly include people from academic, U.S government and commercial sectors who perform research

related to space weather effects on Earth and on space. SPIDR provides tools and methods that allows data users to browse, download and visualize these datasets. These services are made available via web applications (accessed via a portal) and via SOAP-based web services. Scientists using SPIDR currently face problems and difficulties while using the SPIDR web interface, as these do not offer any data processing capabilities and lacks ways to integrate data from their research tool. We have identified that the accessibility to the space weather data can be improved via scientific workflows. Hence, we developed workflows that automate the routine tasks of the scientists like data downloading, plotting and filtering.

The thesis document is organized as follows. Chapter 2 gives an overview of the computing technologies used in this thesis work. Chapter 3 and 4 presents the problem domain and the proposed solutions. Results of this work are presented in Chapter 5 and few of the related work are explained in Chapter 6. Conclusion and future work are discussed in Chapter 7.

## CHAPTER II

### BACKGROUND TECHNOLOGIES

This chapter gives an overview of the computer technologies used in this thesis work. The main technologies include SOAP-based web services, the Kepler scientific workflow system and the soapUI tool.

#### 2.1 Web Services

A Web Service [9; 17; 36] is an application-programming interface (API) that provides a set of functions/services that are accessed by other software applications running on different machines, irrespective of their location and implementation. As defined by W3C Web Service Architecture [4], *‘A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL [5; 17; 37]).’* They make use of standard XML [5] messages for their communication via Hypertext Transfer Protocol (HTTP).

Advantages of web services include:

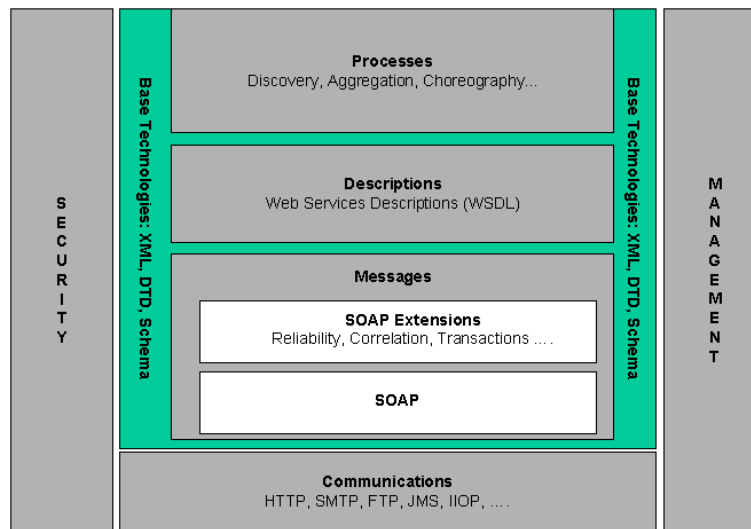
- 1) Interoperability: The use of XML messages makes web services both language and platform independent. For example: an application written in Perl and running under Windows can request a service from another machine where the service is written in Java and running under Linux OS.
- 2) Reusability: Web services allow reuse of other services in other Web Service components.
- 3) Usability: Since web services make use of open standards and protocols, it is easier for developers to understand.
- 4) Easy Integration: Web services allow programs and services from different organizations and locations to be combined easily to provide an integrated service. For example: A custom

application can make use of the services provided by companies like Amazon [2], Google [7], Yahoo [38] and offer an integrated service.

There are two types of web services: SOAP-based [5; 17; 35] and RESTful services. We will focus on SOAP-based services as this is the technology previously used by SPIDR.

### 2.1.1 SOAP-based web services:

These web services make use of Simple Object Access Protocol (SOAP [5; 17; 35]) to exchange messages via HTTP.



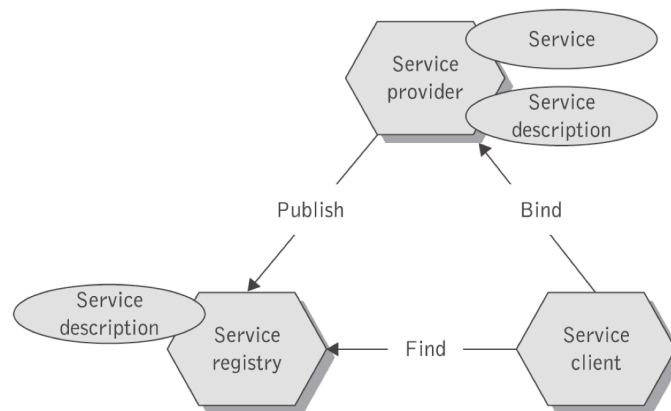
**Figure 1: Web Services architecture**

Figure 1 taken from [4] shows the web service architecture stack that involves many layered and interrelated technologies. This section describes some of the technologies that are involved in making a web service call. In the communications layer, HTTP is used as the transport protocol for the messages. SOAP provides the messaging framework to be used by the applications and the web services to send messages and data across the network. Web Service Description Language (WSDL [5; 17; 37]) contains information about the service operations along with the parameters. This is an XML-based language describing how to interact with the web services and their implementation details. In order to have the service requestor/client know about what web

services are being offered by an organization, the service provider registers their web services on a service registry which acts like a look up directory for the requestor. The requestor can then search and identify the service that would be useful to them. For service registration and discovery, the Universal Description, Discovery and Integration (UDDI [5; 17]) was created.

### 2.1.2 Web Service Invocation:

The figure below taken from [17] demonstrates the web service interactions between the provider and the client.



**Figure 2: Web Service Operation**

First, the service provider publishes its service description (WSDL) with the service registry. The client then looks up for the service in the registry and locates the service it is interested in. It then makes use of the WSDL information obtained from the service registry and invokes the service from the service provider.

### 2.1.3 SOAP:

SOAP is an XML-based technology that allows disparate applications to communicate with each other in a distributed environment. SOAP encapsulates the XML data that is exchanged between applications and web service components. The SOAP message consists of an <Envelope> element, which wraps an optional <Header> element, and a mandatory <Body>

element.

### Structure of a SOAP message:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  <soapenv:Header>
    <!--Application specific information -->
  </soapenv:Header>
  <soapenv:Body>
    <!-- Application specific data -->
  </soapenv:Body>
</soapenv:Envelope>
```

The <Envelope> element is the root element that defines the framework for processing the message. The <Header> element is the immediate child of the <Envelope> element and it contains information about the message, like its origination, destination and also carries information on digital signatures. The <Body> element contains the message or data that is to be sent.

In the example below, GetItemQuantity is the request sent to the server and the GetItemQuantityResponse is the response from the server. The SOAP request has a parameter ItemName and the response is contained in the ItemQuantity parameter. The namespace is defined in <http://www.example.com/Item/>.

### SOAP Request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  <soapenv:Body xmlns:item="http://www.example.com/Item/">
    <item:GetItemQuantity>
      <item:ItemName>Mac Book</item:ItemName>
    </item:GetItemQuantity>
  </soapenv:Body>
</soapenv:Envelope>
```

### SOAP Response:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  <soapenv:Body xmlns:item="http://www.example.com/Item/">
    <item:GetItemQuantityResponse>
```



```

        <item:ItemQuantity>200</item:ItemQuantity>
      </item:GetItemQuantityResponse>
    </soapenv:Body>
  </soapenv:Envelope>

```

There are two communication styles for SOAP: RPC (Remote Procedure call) and Document (or message). In RPC-style web service communication, the client makes a synchronous method call to the service provider with a set of parameters and the service responds with the return value.

While, with Document-style communication, the client sends an entire XML document rather than a set of arguments and the service responds asynchronously with another document.

#### 2.1.4 WSDL:

WSDL is an XML-based service description language that specifies the interface to the web service, the service operations and where it is located. Through WSDL the provider informs the client of what service it provides and how to invoke them.

Sample WSDL document structure:

```

<!-- root element 'definitions' defines a set of related services -->
<definitions targetNamespace="http://www.example.com/sample/wSDL/">

  <!-- 'types' element defines the data types involved in the operations -->
  <types>

    <!-- Namespace declarations -->
    <schema targetNamespace="http://www.example.com/sample/wSDL/">
      <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>

      <!-- Definition of an array of integers -->
      <complexType name="Array_of_xsd_int">
        <complexContent>
          <restriction base="soapenc:Array">
            <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:int[]" />
          </restriction>
        </complexContent>
      </complexType>
    </schema>
  </types>

```

```

<!-- 'message' element defines the data elements of the operations -->
<!-- request 'AddNumbersRequest' is of type 'Array_of_xsd_int' -->
  <message name="AddNumbersRequest">
    <part name="number" type="Array_of_xsd_int"/>
  </message>

<!-- response 'AddNumbersResponse' is of type Integer -->
  <message name="AddNumbersResponse">
    <part name="value" type="xsd:int"/>
  </message>

<!-- 'portType element' states the operations performed and the messages associated with
them -->
  <portType name="AddNumsInterface">
    <operation name="Addition">
      <input message="AddNumbersRequest"/>
      <output message="AddNumbersResponse"/>
    </operation>
  </portType>

<!-- 'binding' element describes the format of the message and the protocol involved in
making the call -->
  <binding type="AddNumsInterface">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="Addition">
      <soap:operation soapAction="http://www.example.com/sample/wsdl"/>
      <input name="AddNumbersRequest"><soap:body use="literal"/></input>
      <output name="AddNumbersResponse"><soap:body use="literal"/></output>
    </operation>
  </binding>
</definitions>

```

The corresponding web service pseudocode for the above WSDL would look like:

```

Function Addition (int[] a)
  initialize b to 0
  for x from 0 to n
    b = b + a[x]
  return b

```

### 2.1.5 UDDI:

UDDI [5; 17] is a directory of web services created for the clients to search and discover the services they are interested in. The service provider by publishing their services on the UDDI, make it easier for the client to look up for the service they want and integrate it with their

application. Through this, the client gets an idea of what web services are currently available. Presently, over 220 companies are members of the UDDI community.

### **2.1.6 SOAP implementations for developers:**

There are many SOAP implementations available for developers, including: Apache Axis for Java [3; 11], SOAP:: Lite for Perl [11; 25] and Microsoft .NET[11; 18].

## **2.2 The Kepler Scientific Workflow System**

Kepler [1; 12; 16] is an open-source software application developed in Java to build scientific workflows that help scientists with analyzing and modeling scientific data. It provides a graphical user interface containing the various components and an editor for creating and executing the workflows. The components can be dragged and dropped on the editor where they can be connected and executed. These workflows can be exchanged as XML and Kepler provides the option of sharing these workflows with other scientists across the world and also connects to data sets from remote location. Kepler is based on the Ptolemy II [21] framework, developed at the University of Berkeley. This is a Java-based open-source framework that enables computational modeling and design. From the Ptolemy II website:

*“The key underlying principle in the [Ptolemy] project is the use of well-defined models of computation that govern the interaction between components.”*

The concept of ‘Actors’ and ‘Directors’ in Kepler are notable features adopted from Ptolemy II.

### 2.2.1 Sample Ptolemy II model structure:

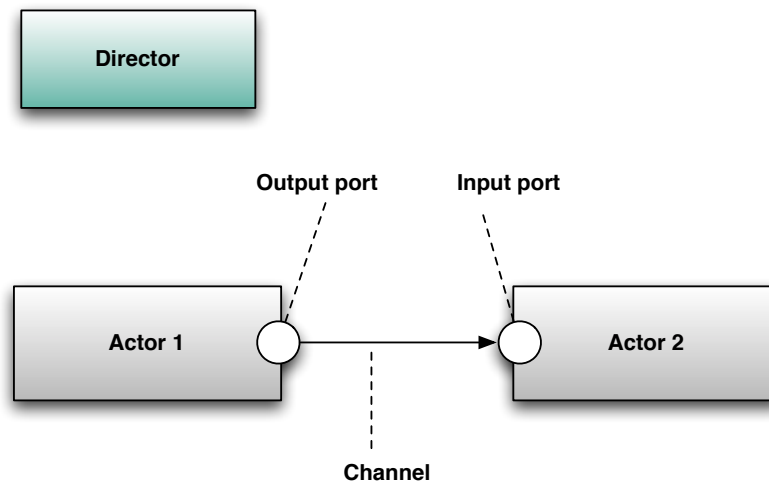


Figure 3: Sample Ptolemy model structure

In Ptolemy II, a model is viewed as a composition of individual, reusable, connected components called ‘Actors’ and the interaction between them. The execution of the model/workflow and the communication between the actors are controlled by another component called the ‘Director’. The director makes sure that the actors are fired in the proper order. A workflow cannot be executed without a director. There are many types of director and each differs by how the workflow is executed. An actor communicates with other actors by means of interfaces called ‘ports’. There can be single/multiple input and output ports. A single port allows only one connection from/to the actor, while a multiple port allows more than one connection from/to the actor. All the actors in the workflow must be connected. The connection that exists between the ports is called a ‘channel’.

### 2.2.2 Kepler Installation:

Kepler is cross-platform software application compatible across the Windows, Mac OS X and Linux operating systems. All three versions of Kepler can be downloaded from the Kepler website [12] and installation instructions can be found in the User Manual [6].






### 2.2.3 Components of Kepler workflow:

The components of a workflow include the directors, actors, parameters and ports.

#### Directors [6; 13]:

The director is an important component in a Kepler workflow. It provides the algorithm that determines when actors in a workflow will perform their computations. Kepler offers five different directors to choose from. These include SDF, PN, DDF, DE and CT.

**Table 1: Available Directors in the Kepler component library; Taken from the Actor-Reference Guide at [13]**

Director	Description
 SDFDirector	The SDF Director is often used to oversee fairly simple, sequential workflows. Types of workflows that run well under an SDF Director include processing and reformatting data, converting one data type to another, and reading and plotting a series of data points.
 PNDirector	The PN Director is often used for managing workflows that require parallel processing on distributed computing systems.
 CTDirector	The CT Director is designed to oversee workflows that predict how systems evolve as a continuous function of time (i.e., "dynamic systems").
 DEDirector	The DE Director is often used for modeling time-oriented systems: queuing systems, communication networks, and occurrence rates or wait times.
 DDF Director	The DDF Director is often used for workflows that use looping, branching or other control structures but do not require parallel processing (in which case a PN Director should be used).

#### Actors [6; 13]:

Actors are the main processing elements in the workflow. Kepler actors can be used to access data sources and web services, perform statistical computations and also provide the ability to create graphs of the data being processed by a Kepler workflow. More than 350 actors are available in Kepler's component library and actors can also be downloaded on-line.

Additional actors are constantly added to increase Kepler's functional capabilities. Kepler also

provides the user the ability to develop their own actors and share it with others through a repository. There are two types of actors: individual and composite actors. Individual actors are independent processing elements, while composite actors are collections of individual actors created to perform one complex operation. In other words, a composite actor represents a nested sub-workflow. An entire workflow can be added to a composite actor and the resulting actor can be included in another workflow.

### Example of an individual actor:

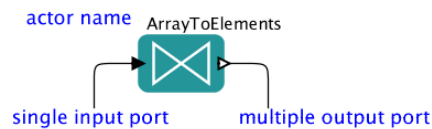


Figure 4: ArrayToElements actor

The ArrayToElements actor accepts an array of any type via its single input port and outputs the disassembled values of the array via its multiple output port. The actor name that appears on the top of the icon denotes the function performed by the actor and can be customized by choosing ‘customize name’ on the right-click menu. A single port is differentiated from a multiple port by a shaded triangle. The right-click menu on the actor provides options to configure the actor and its ports and to view the documentation of the actor.

### Example of a Composite Actor:

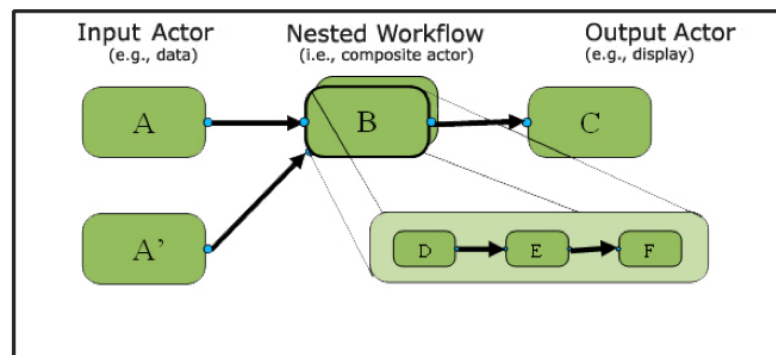


Figure 5: Example of Composite actor; Taken from the Kepler Getting Started Guide [6]

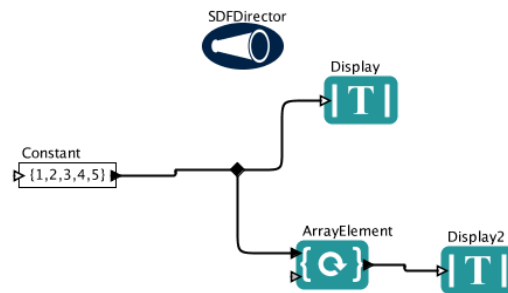
In the above workflow, actor B is the composite actor; it includes another workflow consisting of actors D, E and F. The input to actor B is routed to the input port of actor D and the output from actor B is coming from the output port of actor F. The workflow embedded in B contains a director that controls its execution. The workflow starts executing when actor B is fired on the upper level. There can be many levels of workflows and each level has its own director.

### Relations, ports and parameters [6]:

Relations allow workflows to branch a data flow. The branched data can be sent to multiple places in workflow. For e.g.: an output from an actor can be directed to both a display actor and to another actor for processing.



**Figure 6: Relation icon**



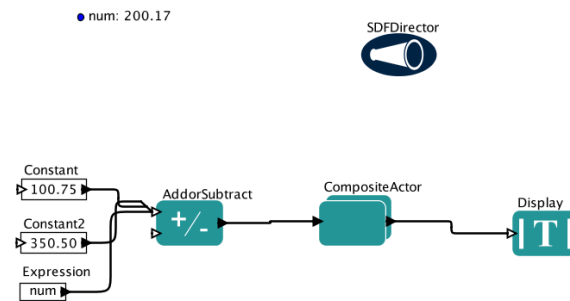
**Figure 7: Sample workflow demonstrating 'relation' in a workflow**

Ports are interfaces through which actors communicate with each other. Ports that are coupled with the actors are called “Actor ports”, while the ports that connect a workflow to another workflow is called ‘External ports’.

Parameters are variables declared in a workflow. Their values can be used in any part of

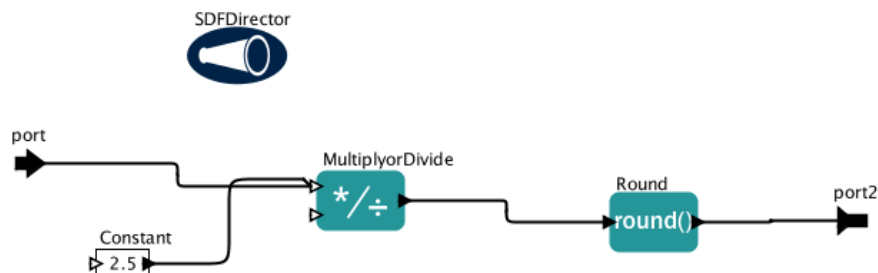
the workflow.

### Sample Kepler Workflow:



**Figure 8: Sample Kepler Workflow**

As you see, workflows developed in Kepler have structure similar to the Ptolemy model. The above is a simple workflow that performs some numerical calculations. It adds three numbers, multiplies the result with another number and the overall result is rounded up before it is displayed in a text window.

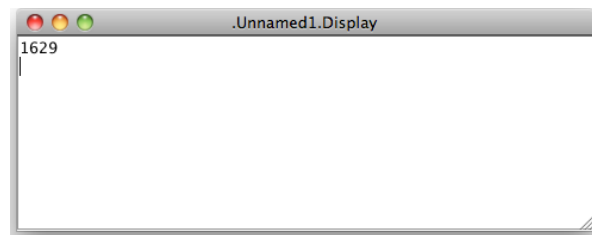


**Figure 9: Composite actor**

This workflow contains a composite actor that performs the multiplication and the rounding calculations. In Figure 9, port and port2 are ‘External ports’ that connects a workflow to another workflow. In this case, it connects the nested sub-workflow in the composite actor with the main workflow. The ‘SDFDirector’ controls each of the workflow execution, as this is a sequential workflow where the order of the actor execution is pre-determined. For adding the numbers, we make use of the ‘AddorSubtract’ actor. As the name indicates, this actor can be used for both the



arithmetic operations: plus and minus. The 'AddorSubtract' actor has two multiple input ports. The first port accepts integers that need to be added and the second port accepts integers that need to be subtracted. Since it is a multiple input port it accepts more than one input. The input is given via three actors. The 'constant' actor outputs the constant value specified in the parameter. The 'Expression' actor evaluates an expression and outputs the value of the evaluated expression. In this case, the 'Expression' actor evaluates the value of the parameter 'num' defined in the workflow and outputs the value to the 'AddorSubtract' actor. The output from the 'AddorSubtract' actor is given 'MultiplyorDivide' actor. This is similar to the 'AddorSubtract' actor but performs the functions of Multiply or Divide. Using the 'Round' actor, the result is rounded up and displayed in a window.



**Figure 10: Output of the sample Kepler workflow**

### **2.3 soapUI Testing Tool:**

For testing of web services we made use of the soapUI tool. soapUI [26] is a open-source testing tool for SOAP-based web services. It is functional testing tool where all the web service operations can be tested against various inputs. The tool takes the WSDL URL as input and generates SOAP requests for all the operations defined in the WSDL. It provides a clean interface to run the tests and to view the SOAP responses.

## CHAPTER III

### APPROACH

This chapter discusses about the SPIDR's web page interface and the SOAP-based services offered to the scientists to access the space weather data, problems faced by them in using the SPIDR interfaces and the possible solutions to the problems. The proposed solution focuses on improving SPIDR's SOAP-based services by adding new functionality and making it easy to use for the scientists with a desktop-based data processing and analysis application, hence providing greater accessibility to the scientists who use SPIDR's data in their day-to-day research.

#### **3.1 SPIDR web page interface**

SPIDR [27; 39] provides an interactive web page interface [30] to access its data sets. It is a distributed network application developed using various open-source tools and currently archives solar, geomagnetic, ionospheric and several other types of data from ground observatories and satellites. This tool allows data users to browse, download and visualize these data sets.

It performs two main functions: data management and data access. The data administrator has the privileges to load data to the SPIDR databases and the data user can download these data sets in various formats, view their metadata and can also download a graph plot of the data. The page displays all the data sets with links to the metadata holdings and the time period for which data is available.

Data access is a three-step process.

- 1) Choosing the time interval the user is interested in
- 2) Picking the data set
- 3) Selecting the parameters and the download option

The user selects the time interval for which he needs the data and the data sampling interval. On selecting this, the data sets categories page displays all the data sets that have data in the time interval selected by the user. Then the user chooses the data set followed by selecting a few related parameters; SPIDR then offers the user with the choice to download data in a variety of formats such as ASCII, XML and MATLAB or plot a graph of the data.

### **3.2. Existing SOAP-based services**

SPIDR offers two separate SOAP-based services to allow applications to access SPIDR as a data source. They are called the FileService and the SpidrService. The FileService web service provides methods to load space weather data into a database and the SpidrService web service provides methods to access these data sets. SPIDR also provides client applications that make use of these web services. They include: FileClient, SpidrClient and the MetadataClient. These applications are wrapped within a spidr.war file and can be obtained from SourceForge [29]. The client applications can be invoked from a command prompt. FileClient can be used for loading data into SPIDR; the MetadataClient is used to retrieve metadata and inventory details; and the SpidrClient is used to exporting data from the SPIDR database.

We will now go into detail on the existing data retrieval web service and its client. The WSDL (Web Service Description Language) for the ‘SpidrService’ web service can be found at:

<http://spidr.ngdc.noaa.gov/spidr/services/SpidrService?wsdl>

#### **3.2.1 Methods:**

Below are some of the methods offered by the SpidrService related to data retrieval.

- public string getTables (string viewGroup)

This method obtains a list of tables for the specified view group name. It returns a string, containing the list of SPIDR tables.

- `public InventoryBean[] getInventory (string table, string station, string element, string dataFrom, string dateTo)`

This method returns the inventory details for a given table, station, element and time interval.

The value returned is a SOAP array of type `InventoryBean`. Each element of type `InventoryBean` includes information on the element name, the number of records for the given time interval, the station name that collected the data, a table name, and the time interval itself.

- `public string getData (string Table, string Stn, string Elem, string dataFromStr, string dateToStr, string format, string samplingStr)`

This method exports the data based on the given table name, station, element, time interval, format and sampling interval and returns the URL link to the exported file.

- `public string getDataStream (string spidrTable, string station, string element, string startDayAndTime, string stopDayAndTime, string format, string samplingRate)`

This method returns raw data in the format specified as input for the given table name, station, element, time interval and sampling rate.

- `public StationBean[] getStations (string table)`

This method returns the list of station belonging to the specified table name. The value returned is an array of type `StationBean`. Each element of type `StationBean` includes information about a particular station including its name, lat/long, station code, etc.

- `public InventoryBean[] getElementPeriods (string table)`

This method returns the time period for which data is available for all the elements of the given table. The value returned is an array of type `InventoryBean`.

- `public InventoryBean[] getStationPeriods (string table)`

This method returns the time period for which data is available for all the stations of the

specified table. The value returned is an array of type InventoryBean.

### 3.2.2 SpidrClient:

This client application can be called from the command prompt using the following command.

```
> java spidr.export.SpidrClient <parameters>
```

It downloads the data specified by the parameters to the local machine and returns the name of the file.

**Table 2: Command line parameters for SpidrClient; taken from SPIDR guide [28]**

Parameters	Description
(-h   --help)	Prints usage help
(-l   --link) <url>	Sets the web-service URL
(-u   --user) <name>	Sets user name
(-p   --passwd) <password>	Sets password
(-t   --table) <table>	Sets SPIDR table name
(-s   --station) <station>	Sets station code
(-e   --element) <elements>	Sets element name
(-d   --day) <day>	Sets a single day for the data request. <day> must be given YYYYMMDD. If you need to obtain data within a time interval, use d1 and d2 parameters.
(-d1   --dayfrom) <day>	Sets upper boundary of the time interval. <day> must be given as YYYYMMDD.
(-d2   --dayto) <day>	Sets lower boundary of the time interval. <day> must be given as YYYYMMDD.
(-f   --format) <format>	Sets export format. <format> may be one of the following: ASCII, XML, Matlab, IIWG (ionospheric data only), WDC (geomagnetic data only)
(-ts   --timestep) <sampling>	Sets data sampling in minutes.

### 3.3 Problem Domain

Scientists were not completely satisfied with the way of accessing SPIDR data via its web interface and the command-line client application. This interface did not satisfy the entire

scientific community needs to access the data. Scientists who work with this data everyday for their research wish to access the SPIDR data directly from their data analysis software/tool. The web page interface does not provide this facility to the scientists and the existing SOAP-based services produce complex SOAP array type outputs that introduces difficulties and inconvenience to the scientists while making calls to these service operations from their data analysis applications. So, their only option was to make use of the command-line client application to download the data.

The web page interface and the SOAP-based services with its client application is a useful tool for data downloading and plotting activities, but does not provide any interactive data processing capabilities and also offers no ways to combine data from other data sources. It retrieves data from its internal databases and displays it to the scientists. The scientists after downloading it to their machine perform various data processing, filtering and analysis techniques on the data using data analysis and research tools.

### **3.4 Proposed solution**

On analyzing the problems, we found that by enhancing the SOAP-based services by adding new functionality and by developing an interactive client application to support them, it would help the scientists to better access SPIDR data. To improve the SOAP-based services to include data processing capabilities, we added new operations to the service. To identify what new services can be added to SPIDR, we got input from the users and scientists by analyzing the ways they access and manipulate the data currently. From that, we found quite a number of processes that are widely being used by researchers. To begin with, we worked upon adding the High/Low Threshold filter service to SPIDR. This filter process is used by scientists to filter the data they need for their research. By adding this operation to the service, the scientists can

directly download the filtered data, rather than downloading the data to their machine and then using external applications to filter out the ones they do not need.

For the SOAP client application, we chose to develop a scientific workflow client using the Kepler workflow system. The reasons for choosing Kepler over other scientific workflow systems is because:

- 1) Kepler offers its users with a wide range of models of computation with the help of its ‘Director’ components.
- 2) The Kepler tool is already widely used in fields such as ecology and geology and hence it is quite popular among geoscientists. Providing the scientists with a tool that they are already familiar with makes them comfortable in using the data.
- 3) Kepler has great support for SOAP-based services and data visualization techniques.

Using Kepler, we designed and developed workflows that automate the routine tasks of the scientists by adapting the existing SOAP-based services. The basic tasks that scientists perform are data downloading and generating a plot of the data. We started off by creating a Kepler workflow for downloading and plotting of data from a single dataset. The parameters were hard-coded in the workflow. Since it is not a good approach to have separate workflows for each datasets and also not so user-friendly, we analyzed that it would be better to develop interactive workflows that get input from their users and can also be applied across multiple datasets. We then developed a separate workflow that interacts with the user to obtain inputs and sets up the parameters required for data downloading and generating a plot of the data. This workflow was added as a composite actor to the first developed workflow, replacing the hard-coded inputs. Thus, we ended up developing two interactive workflows, one that supports downloading of data and the other workflow generates a plot of the data. We also created a

sample Kepler workflow that makes use of the newly added High/Low Threshold Filter web service. These workflows were added to the Kepler system and distributed as a customized Kepler-NGDC [14] workflow application.



## CHAPTER IV

## METHODS

This chapter explains the implementation of the solutions to the problems. It also states the problems faced while implementing the solutions and their associated limitations.

#### 4.1 High/Low Threshold Filter web service

The Filter Service (known as the HiLoFilterService) takes the data as a comma-separated value string and a threshold value as input and outputs the filtered data based on the high/low function selected. The output is either an integer array or a float array. The HiLoFilterService offers 4 methods. The WSDL for this web service can be found at:

<http://spidr.ngdc.noaa.gov/spidr/services/HiLoFilterService?wsdl>

##### 4.1.1 Architecture Diagram:

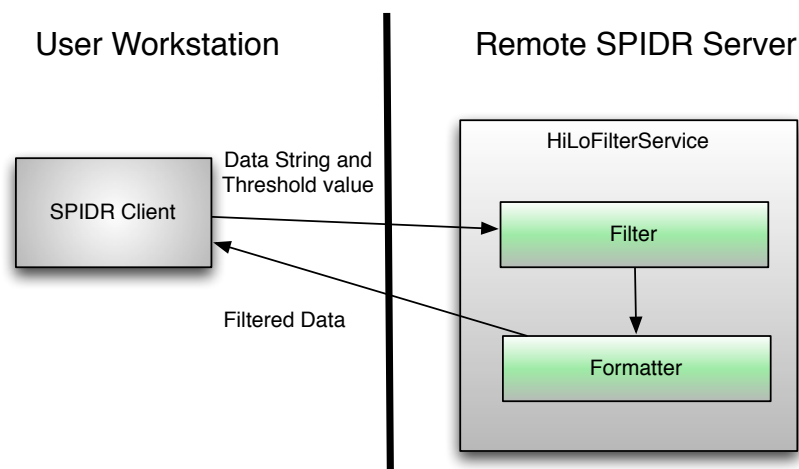


Figure 11: HiLoFilterService operation

The HiLoFilterService resides in a remote SPIDR server. The client makes a method call by passing the data values and the threshold value as input arguments. The service executes the called method that filters the data and formats the filtered data as an array and returns the result to the client.

#### 4.1.2 Methods:

- `public int[] loPassFilterInt (string data, int thresholdValue)`

This method takes the data string and a threshold value as input and outputs the data values that are below the threshold value as an integer array.

Pseudocode of the function is given below:

```
function int[] loPassFilterInt (string data, int thresholdValue)
    var y[0..n]= split input data string by (,)
    var h[0..n]
    initialize i to -1
    for x from 0 to n
        var int a = integer (y[x])
        if (a<= thresholdValue)
            increment i
            h[i]=a
    var r[0..i+1]
    copy array values from h[0..n] to r[0..i+1]
return r[0..i+1]
```

- `public int[] hiPassFilterInt (string data, int thresholdValue)`

This method takes the data string and a threshold value as input and outputs the data values that are above the threshold value as an integer array.

Pseudocode of the function is given below:

```
function hiPassFilterInt (string data, int thresholdValue)
    var y[0..n]= split input data string by (,)
    var h[0..n]
    initialize i to -1
    for x from 0 to n
        var int a = integer (y[x])
        if (a>= thresholdValue)
            increment i
            h[i]=a
    var r[0..i+1]
    copy array values from h[0..n] to r[0..i+1]
return r[0..i+1]
```

- `public float[] loPassFilterFloat (string data, int thresholdValue)`

This method takes the data string and a threshold value as input and outputs the data values that are below the threshold value as a float array.

Pseudocode of the function is given below:

```
function float[] loPassFilterFloat (string data, int thresholdValue)
    var y[0..n]= split input data string by (,)
    var h[0..n]
    initialize i to -1
    for x from 0 to n
        var int a = float (y[x])
        if (a<= thresholdValue)
            increment i
            h[i]=a
    var r[0..i+1]
    copy array values from h[0..n] to r[0..i+1]
return r[0..i+1]
```

- public float[] hiPassFilterFloat (string data, int thresholdValue)

This method takes the data string and a threshold value as input and outputs the data values that are above the threshold value as a float array.

Pseudocode of the function is given below:

```
function float[] hiPassFilterFloat (string data, int thresholdValue)
    var y[0..n]= split input data string by (,)
    var h[0..n]
    initialize i to -1
    for x from 0 to n
        var int a = float (y[x])
        if (a>= thresholdValue)
            increment i
            h[i]=a
    var r[0..i+1]
    copy array values from h[0..n] to r[0..i+1]
return r[0..i+1]
```

#### 4.1.3 Testing:

The developed HiLoFilterService was tested using soapUI - 3.5, a functional Testing tool for Web Services. The goal of the testing was to make sure that all the operations offered by the

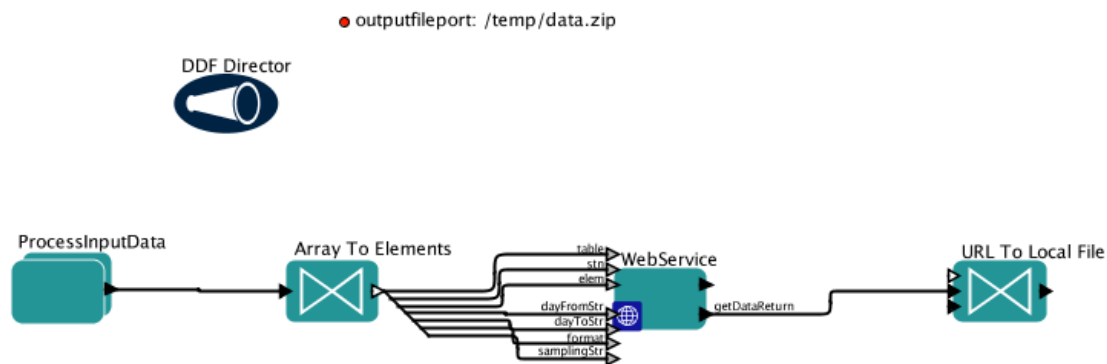
service worked correctly for valid inputs. Different values for data and threshold were given as inputs and the outputs were compared against the actual outputs. After it passed all the test cases, it was deployed on SPIDR's Production server.

## 4.2 Kepler-NGDC scientific workflows

As part of this thesis work, we developed three Kepler workflows that made use of the existing SOAP-based services and the new filter service added. All these workflows are part of the Kepler-NGDC [14] application which can be downloaded from the SPIDR website. The workflows can be accessed from the 'Workflows' tab of the application.

### 4.2.1 Interactive workflow for SPIDR data downloading:

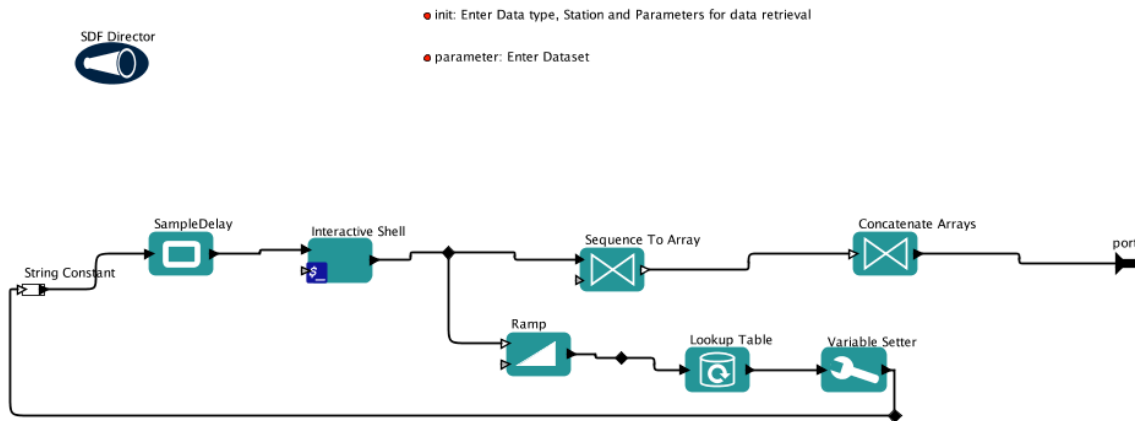
This workflow is named as 'SPIDR\_Multi\_Prompt' in the Kepler-NGDC application. It allows the users to download data from the SPIDR database and save a copy of it to their local machine. It interacts with the users to get the input arguments that are needed to retrieve data from the database and also asks them to enter the path and file name that should be used for the downloaded data file.



**Figure 12: Interactive workflow for SPIDR data downloads**

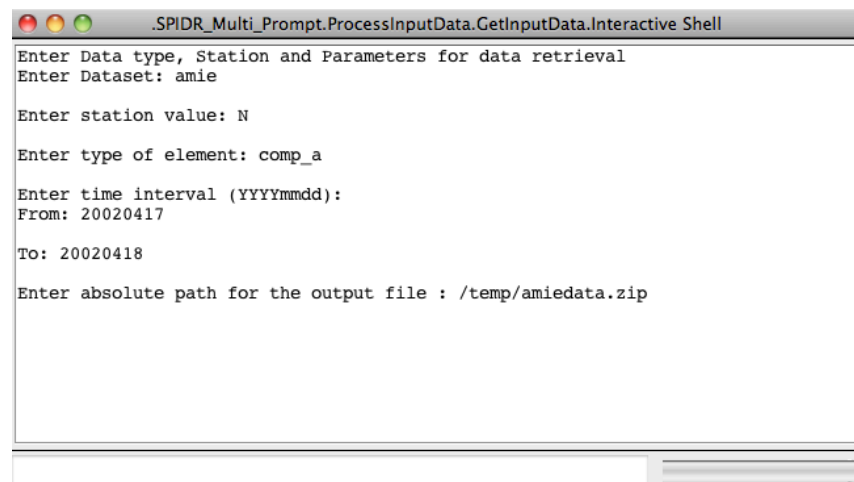
The workflow consists of 4 actors, each with their own role to play in the workflow (see Figure 12). The DDF director dictates the actors' executions. The ProcessInputData actor is

responsible for interacting with the user to get the necessary inputs and prepares the inputs to serve as arguments to query the database. The ProcessInputData is a composite actor. For details on directors, actors, composite actors and connection, please refer to Chapter 2. The ProcessInputData actor encapsulates another composite actor, GetInputData.



**Figure 13: GetInputData composite actor**

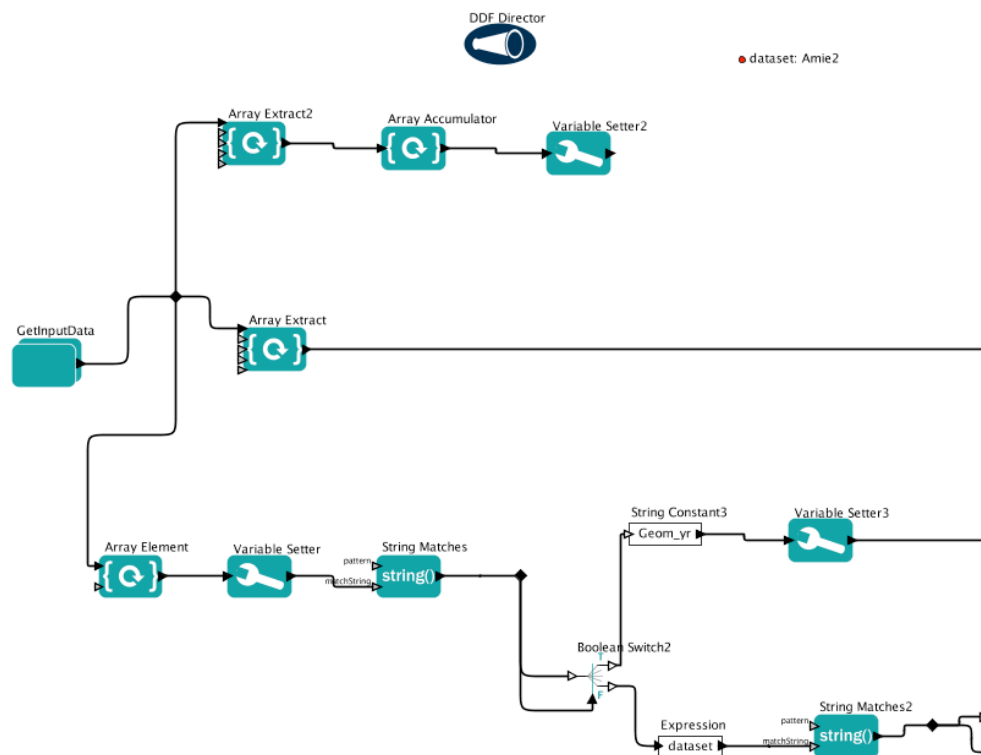
The GetInputData composite actor contains the interactive part of the workflow. We made use of the InteractiveShell actor to display the prompt command window.



**Figure 14: Command prompt window**

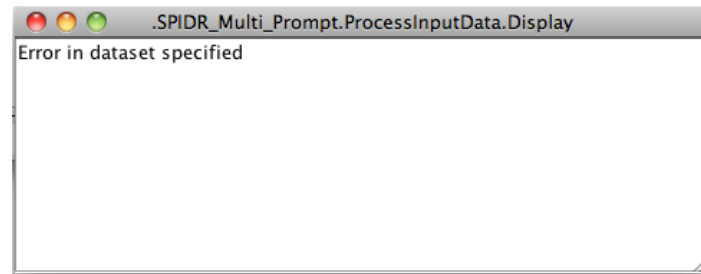
The GetInputData actor displays the text-based form to the user prompting them to enter the data

set name, station name, element name, time interval they are looking for along with the absolute path for the output file (the directory path under which the file is to be saved and the file name to be assigned). The values entered by the user are collected in an array and passed to the ProcessInputData actor. This actor then sets the outputfileport parameter value to the absolute path of the downloaded file and extracts from the array the query parameters to retrieve the data. The actor retrieves the table name from the data set value entered by the user. It matches for a particular keyword in the data set name and sets the table name appropriately. It is not required for the user to know the table name. However, the workflow expects the user to know the station code and element name for that particular dataset. For example: If the user enters ‘Ionospheric data’ for the data set value, the actor looks for the word ‘iono’ and assigns the table name as ‘iono’.



**Figure 15: ProcessInputData composite actor**

The workflow supports for 14 different data sets and if the user enters a value different from the supported data sets, it displays an error that the data set specified is invalid.

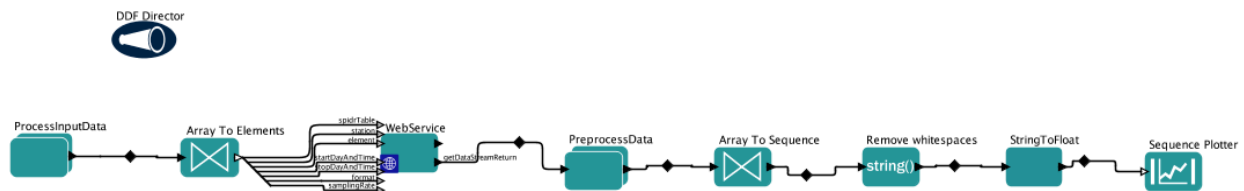


**Figure 16: Error message displayed to the user**

The output from the ProcessInputData actor is an array, which is split into elements using the ArrayToElements actor. The output is then passed to the WebService actor that calls the corresponding remote SpidrService and executes the getData method. The output of the operation is a URL link pointing to the datafile in the remote server. The URLToLocalFile actor reads the URL and saves the file under the path and the filename specified by the user.

#### 4.2.2 Interactive workflow for SPIDR data plotting

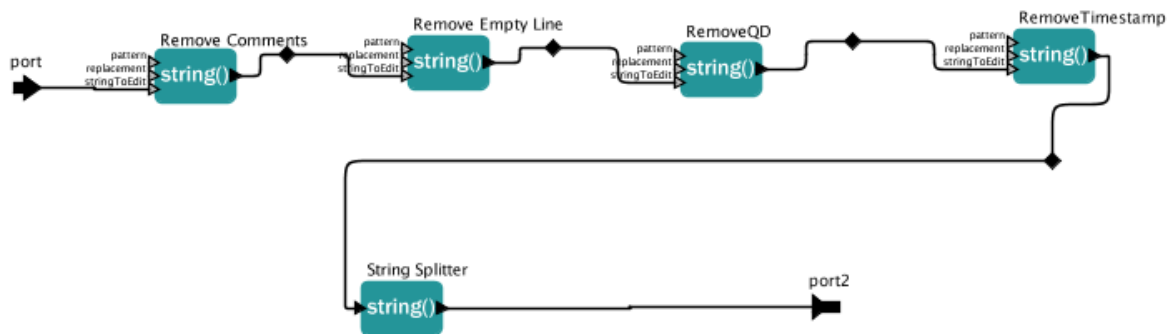
This interactive workflow is named as 'SPIDR\_Interactive\_Plotting' in the Kepler-NGDC application. It is very similar to the previous one, except that instead of downloading the data it generates a sequence plot of the data. The actors involved in this workflow consist of actors from the previous workflow and also includes a few other actors necessary to generate the plot.



**Figure 17: Interactive workflow for SPIDR data plotting**

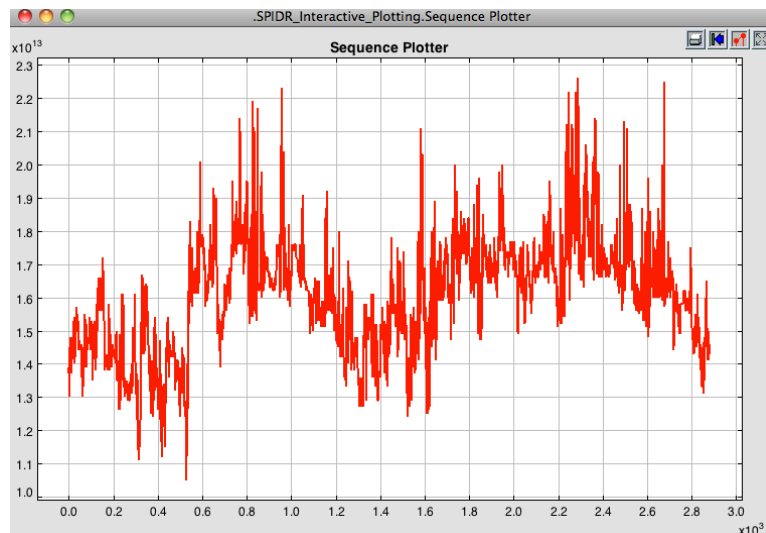
The ProcessInputData actor followed by the WebService actor in the workflow provides the means to get input from the users and to retrieve data from the server. This WebService actor

uses the `getDataStream` method of the `SpidrService` web service and returns data in ASCII format. The output from the web service call is then processed by the `PreprocessData` composite actor to remove comments and other text and to extract only the data values.



**Figure 18: PreprocessData composite actor**

The string data values are then converted to a sequence of float values and then plotted via a `SequencePlotter` actor. Below figure is the plot generated by this workflow.



**Figure 19: Sequence Plot of SPIDR data**

#### 4.2.3 Problem faced:

These Kepler workflows automate the routine tasks of the scientists, which include SPIDR data downloading and graph plotting. It also provides the scientists with interactive data processing and analytical tools through Kepler and using this workflow, the scientists can build



other workflows over them. For example: they can combine data from SPIDR and data from another source and perform further analysis and research on both the data with the tools provided by Kepler. Though this solution solves the problems faced by the scientists, it is a less ideal solution. This is because these workflows allow users to enter the inputs to the workflow. This may cause the possibility for the users to type in wrong station and element names.

Initially we wanted to build a workflow that guides the users through the inputs, by displaying the data sets available, allowing the users to choose the station name from a list of stations and by displaying the time interval for which data is available. To support this kind of design, Kepler provides the SelectionDialog actor that allows users to select a value from a list of values. This actor worked well on the Windows platform, but did not work well on Mac OS X. Due to this incompatibility issue, we had to design the workflow with the text-based form allowing users to enter the inputs. In order to reduce the error caused by this design, we have added a table document to the web site that gives examples of inputs for each of the data set.

Dataset	Dataset pattern	Station	Element	From	To
AMIE derived indices(release 2)	<a href="#">amie</a>	N	comp_a	20020417	20020418
geomagnetic annual means	geomagnetic annual	ABK1	D	20020101	20021231
ionospheric	<a href="#">iono</a>	bc840	h'E	20020408	20020409
<a href="#">hpi dmsp data</a>	<a href="#">hpi dmsp</a>	F11	<a href="#">corr</a>	20000505	20000506
<a href="#">hpi noaa data</a>	<a href="#">hpi noaa</a>	NOAA-12	normFactor	20000505	20000506
cosmic ray data(4096 format)	cosmic ray 4096	athens	<a href="#">criCorr</a>	20020417	20020418
cosmic ray data preliminary	cosmic ray preliminary	<a href="#">kie</a>	<a href="#">cri</a>	20000417	20000418
cosmic ray data hourly	cosmic ray hourly	<a href="#">athn</a>	<a href="#">cri_c</a>	20020417	20020418
cosmic ray data minute	cosmic ray minute	<a href="#">athn</a>	<a href="#">cri_c</a>	20020417	20020418
<a href="#">rstn data</a>	<a href="#">rstn</a>	lear	f1415	20020417	20020418
goes data	goes		10 el	20020417	20020418
<a href="#">imf data</a>	<a href="#">imf</a>	ace	<a href="#">swDensity</a>	20020417	20020418
geomagnetic minute means	geomagnetic minute	<a href="#">bou_min</a>	X	20060805	20060806
geomagnetic hourly means	geomagnetic hourly	<a href="#">ALE_hr</a>	X	20020417	20020418

**Figure 20: Example inputs for the interactive workflows**

However, a few months into the development of Kepler-NGDC tool, the Kepler team solved the incompatibility issue and added a new version of the SelectionDialog actor to the Kepler repository, which is compatible across all OS. We now intend to integrate this dialog into the workflows we developed for the Kepler-NGDC tool.

#### 4.2.4 HiLoFilter workflow for SPIDR Geomagnetic data:

This workflow is named as ‘SPIDR\_Geomagnetic\_Filtered’ in the Kepler-NGDC application. This workflow makes use of the HiLoFilterService to filter geomagnetic data and generates a plot of the filtered data.

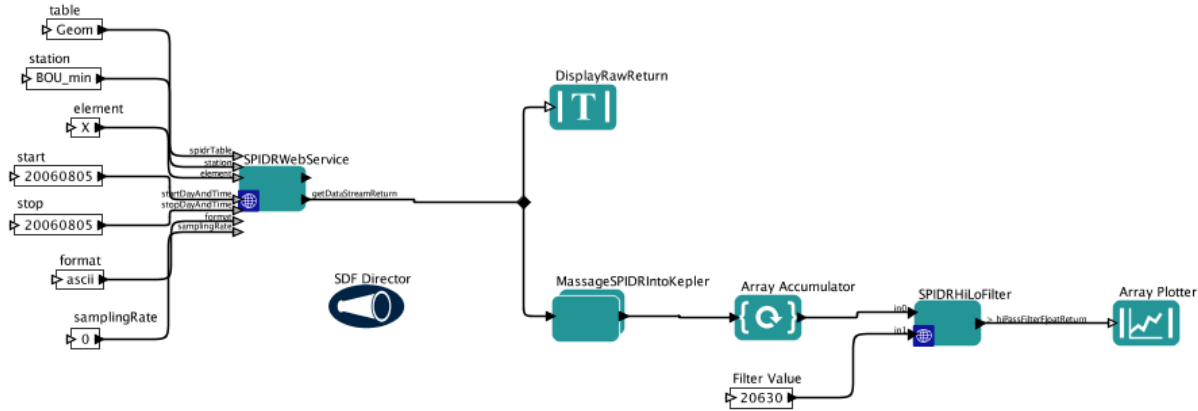


Figure 21: HiLoFilter workflow for SPIDR Geomagnetic data

This is a sample workflow created to make use of the HiLoFilter service developed. Hence, the inputs to the workflow are hard-coded. The WebService actor calls the `getDataStream` method of the `Spidrervice` to retrieve data in ASCII format.

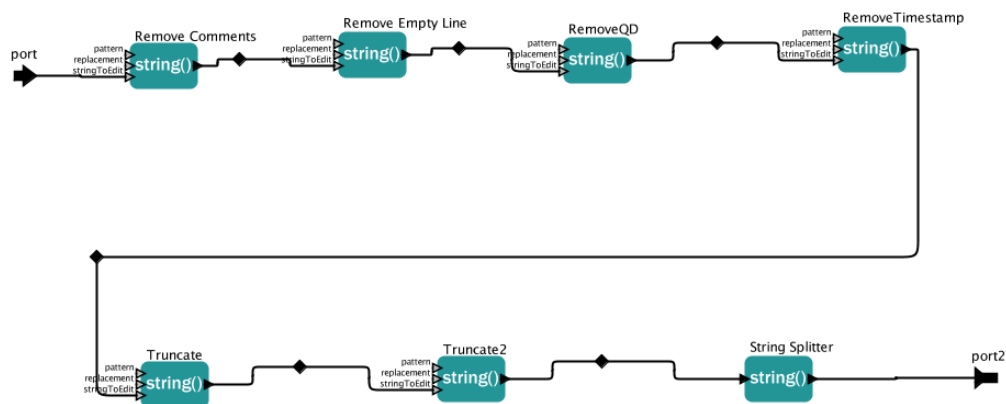
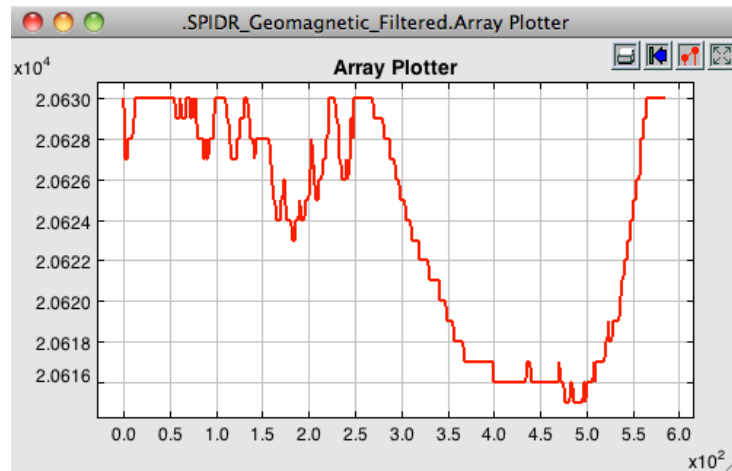


Figure 22: MessageSPIDRIntoKepler composite actor

The data is passed to the `MessageSPIDRIntoKepler` composite actor, which formats and processes the data to remove comments and other text and retrieves only the data values. The

data values are then passed to another WebService actor, which calls the HiLoFilterService and executes the method selected by the user. The threshold value is provided via a Constant actor. The filtered data values are plotted using an Array plotter as the output from the previous actor is in array form.



**Figure 23: Plot of SPIDR filtered data**

#### 4.2.5 Testing:

The interactive workflows were tested with inputs from the 14 data sets and the outputs were verified. The goal of this testing was to make sure that the data were downloaded and plotted for all the supported data sets. Once they passed the testing phase, they were added to the customized NGDC-Kepler system.

## CHAPTER V

### RESULTS

In this chapter we discuss the evaluation of the developed Kepler-NGDC tool by scientists and their suggestions for improving the tool.

#### **5.1 Evaluation**

The developed Kepler-NGDC workflow system was evaluated by having two scientists and a software engineer from the National Geophysical Data Center (NGDC) use the created Kepler workflows and recording their feedback. The scientists make use of SPIDR data to carry out their research and the software engineer has experience in developing clients for SPIDR's RESTful Web services. These users have prior Kepler experience and have a basic understanding of what Kepler does and provides. We requested the scientists to use each of the workflows and to build their own Kepler workflows by re-using the composite actors from these built workflows. Following this, they gave us feedback on the system, suggestions on how each of the individual workflows can be improved and also gave suggestions on what new web services and workflows can be included in the Kepler-NGDC application. Below is a summary of their feedback.

#### **5.2 Discussion of Results**

Having prior Kepler experience, the scientists found the Kepler workflows simple to use and easy to understand the flow of the data through the actors/components. However, they found that the interactive workflows required a small learning curve, but once trained in its use, they found it to be a useful tool for issuing repeated data requests. The created Kepler workflows covered their routine tasks with SPIDR data like downloading and plotting. One of the scientists pointed out that the ability of incorporating web services to the workflows, which provides

complex analytical tools for wide variety of data makes the whole experience more interactive, direct and rapid.

Regarding the question of how the created Kepler workflows can be improved, the scientists suggested some features that will enhance the usability of these workflows with the data. One of them observed that in the workflow, SPIDR\_Multi\_Prompt (the Kepler workflow built for interactive data downloading), the prompt shell requests the user to enter the absolute path for the downloaded file and the name of the file. This appears as two separate questions on the prompt shell. The scientist remarked that these can be combined as a single question, prompting the user to just enter the absolute path for the downloaded file which includes both the path and file name for the downloaded zip file. This change has already been deployed in subsequent versions of the Kepler-NGDC application. They also commented that cataloging and documenting the information within the interactive workflows, like which data sets are available for what time ranges, would make the interactive workflows more usable. Currently, the Kepler-NGDC application offers a table document that lists the data sets that are presently applicable to the workflows and also gives an example of the inputs to the workflows. If instead, the workflow guides the user through the inputs, by displaying the data sets that are applicable and stating the time interval for which the selected data is available, it will make the interface more interactive and usable.

Below are the features the scientist preferred to be added to the developed Kepler workflows.

- 1) Display timestamps on the plots. This would help them in case they come across any suspicious data on the plot, then they can use the timestamp to investigate the data set in more detail.

- 2) Display Quality flags of the data on the plot. This would enable the scientists to understand the data.
- 3) Add a refresh option to the Kepler workflow so that it updates itself every 5 minutes with the most recent data; this will help those scientists working with real-time data.
- 4) Add the Lomb-Scargle method of transformation as a web service. This method would allow scientists to look for periodic harmonics in the data.

The users found the composite actors to be very useful. Indeed, the scientists found immediate use for these actors in their own workflows.

On the whole, the users felt that this tool has improved SPIDR, making it a better resource supported by more interactive data processing capabilities. They concluded that this tool has offered the ability to combine and process data from both SPIDR and other data sources like Space Weather Prediction Center (SWPC) quickly and easily.

## CHAPTER VI

### RELATED WORK

Many scientific data centers provide web page interfaces and software tools to access and perform analysis on their datasets. This chapter will cover two such data analysis tools that are very similar to this work.

#### **6.1 THEMIS Data Analysis Software:**

The THEMIS Data Analysis Software [32] (TDAS) is an IDL-based software (IDL - Interactive Data Language) tool developed to provide the public access to THEMIS (Time History of Events Macroscale Interactions during Substorms) data. NASA's THEMIS [33] mission is to identify what physical processes cause the auroras to occur in the Earth's magnetosphere. THEMIS data includes observations from both ground observatories and the spacecraft. These datasets provide observations of the solar wind, magnetosphere, aurora and the magnetic field perturbations over Northern North America. The THEMIS Data Analysis Software (TDAS) tool is very similar to the work performed in this thesis in that they both provide their associated research communities with a desktop-based direct manipulation tool to access the datasets. This application provides various IDL routines to download, open and view, analyze and plot these datasets. It provides two levels of data. Level 1(L1) data is raw data in CDF format, while Level 2 (L2) data is calibrated in physical quantities. Downloading and plotting routines can be called on both these levels of data. The IDL routines can be invoked from the command-line to execute functions as needed. The software tool in addition provides a Graphical User Interface to access and analyze these datasets. This software tool is compatible across Windows, Linux, and Mac OS X.

## 6.2 Kepler/pPod

There are various projects that make use of Kepler to develop their applications. Kepler/pPod [24] is one of them. It is a customized distribution of the Kepler scientific workflow system, very similar to the Kepler-NGDC tool. It was designed to support researchers involved in phylogenetic data analysis. The goal was to provide the researchers with an easy-to-use desktop application to create, run and share the phylogenetic workflows as well as to manage the provenance of the workflow results. It is one of the core database technologies developed within the NSF-funded processing PhylOData (pPod), which aims at providing a research tool that is helpful to the phylogenetic communities. The Kepler/pPod tool contains reusable components to work with biological sequences and inferring phylogenetic trees and a workflow editor provided by Kepler to create and execute workflows. The tool contains sample workflows for phylogenetic analyses. These workflows can be easily modified by changing the parameters, selecting different input data and by selecting different methods for analysis steps. The tool can be downloaded from [15]. The Kepler/pPod team is working on the tool to add components and workflows that will help the researcher's work with phylogenetic data.

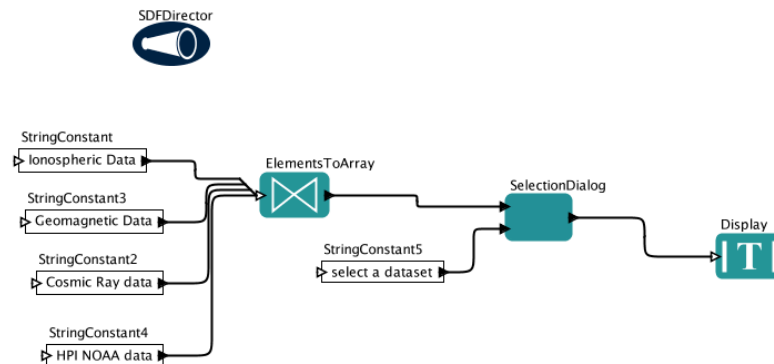


## CHAPTER VII

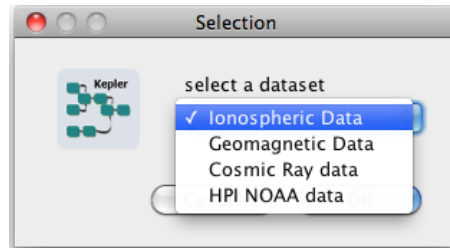
## CONCLUSION AND FUTURE WORK

Through our work we have made a new SOAP-based web service—the HiLoFilterService—available in SPIDR and also provided scientists with an easy-to-use desktop application—the Kepler-NGDC scientific workflow system—to access and visualize the datasets. The Kepler-NGDC tool has built-in workflows to download, filter and plot the SPIDR datasets. These workflows can be customized to the scientists needs and can also be used in other workflows.

In the future, based on community input, new SPIDR SOAP-based web services, such as a service that applies the Lomb-Scargle method of transformation, will be made available alongside additional Kepler workflows that will be added to the Kepler-NGDC tool to enable other types of analysis on SPIDR data. The interactive part of the Kepler workflow will be improved to guide the users through the inputs by making them choose from a list of options. This can be accomplished by using the SelectionDialog actor. Internal documentation to inform users of what data sets are applicable and the time interval for which the selected data is available will also be added to the workflows. Below is an example of the workflow that demonstrates how the interactive part can be improved with the help of a SelectionDialog actor.



**Figure 24: Sample future Kepler workflow**



**Figure 25: Output of the sample future workflow**

With the help of the SelectionDialog actor, the workflow guides the users through the inputs and thus reduces the error caused by entering invalid inputs on the current text-based form. The workflows can be developed using the existing web services and if necessary, new services will also be created to support them. This will make the workflows more usable and user friendly.

## BIBLIOGRAPHY

- [1] Altintas I, Berkley C, Jaeger E, Jones M, Ludäscher B, Mock S. 2004. Kepler: An Extensible System for Design and Execution of Scientific Workflows. Proceedings of the The Future of Grid Data Environments, Global Grid Forum 10
- [2] Amazon web services. <http://aws.amazon.com/>
- [3] Apache Axis for Java: <http://ws.apache.org/axis/index.html>
- [4] David Booth, Hugo Haas, Francis McCabe et al., Web Services Architecture, W3C Working Group Note, 11 February 2004. <http://www.w3.org/TR/ws-arch/>
- [5] E. Newcomer, Understanding Web Services: XML, WSDL, SOAP, and UDDI. Addison-Wesley, 2002.
- [6] Getting started guide - Kepler: <https://code.kepler-project.org/code/kepler-docs/trunk/outreach/documentation/shipping/2.1/getting-started-guide.pdf>
- [7] Google web services: <http://code.google.com/apis/ajaxsearch/>
- [8] I. Foster and C. Kesselman (editors). The Grid: Blueprint for a Future Computing Infrastructure, Morgan Kaufmann Publishers, USA, 1999.
- [9] J.Roy and A. Ramanujan, Understanding Web Services. IEEE IT Professional, November/December, 2001.
- [10] J. Yu and R. Buyya. A Taxonomy of Workflow Management Systems for Grid Computing. Technical Report GRIDS-TR-2005-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, 2005.  
<http://www.cloudbus.org/reports/GridWorkflowTaxonomy.pdf> .
- [11] James Snell, Doug Tidwell, Pavel Kulchenko, Programming web services with SOAP. O'Reilly Media, 2001

- [12] Kepler: <https://kepler-project.org/>
- [13] Kepler Actor Reference Guide: <https://code.kepler-project.org/code/kepler-docs/trunk/outreach/documentation/shipping/2.0/ActorReference.pdf>
- [14] Kepler-NGDC workflow application can be downloaded at  
<http://spidr.ngdc.noaa.gov/spidr/friend.do?hlink=http://spidr.ngdc.noaa.gov/workflow/>
- [15] Kepler/pPod web site: <http://daks.ucdavis.edu/kepler-ppod/>
- [16] Ludäscher B., Altintas I., Berkley C., Higgins D., Jaeger-Frank E., Jones M., Lee E., Tao J., Zhao Y. 2006. Scientific Workflow Management and the Kepler System. Special Issue: Workflow in Grid Systems. Concurrency and Computation: Practice & Experience 18(10): 1039-1065
- [17] Michael P. Papazoglou, Web Services: Principles and Technology. Pearson Prentice Hall, 2008
- [18] Microsoft .NET: <http://msdn.microsoft.com/en-us/netframework/aa663324.aspx>
- [19] NOAA/NGDC <http://www.ngdc.noaa.gov/>
- [20] O'Loughlin, K. F. (1997), SPIDR on the Web: Space Physics Interactive Data Resource on-line analysis tool, Radio Science, 32(5), 2021–2026
- [21] Ptolemy II project and system. Department of EECS, UC Berkeley, 2004.  
<http://ptolemy.eecs.berkeley.edu/ptolemyII/index.htm>
- [22] Scientific Data Management Center website: <https://sdm.lbl.gov/sdmcenter/>
- [23] SciRUN <http://software.sci.utah.edu/scirun.html>
- [24] Shawn Bowers, Timothy Mcphillips, Sean Riddle, Manish Kumar Anand, Bertram Ludäscher, Kepler/pPOD: Scientific Workflow and Provenance Support for Assembling the Tree of Life, Provenance and Annotation of Data and Processes: Second International Provenance and

Annotation Workshop, IPAW 2008, Salt Lake City, UT, USA, June 17-18, 2008. Revised  
Selected Papers, Springer-Verlag, Berlin, Heidelberg, 2008

[25] SOAP::Lite for Perl: <http://www.soaplite.com/>

[26] Soap UI, Web Service Testing tool: <http://www.soapui.org/>

[27] SPIDR: <http://spidr.ngdc.noaa.gov/spidr/>

[28] SPIDR SOAP-based web services user's guide:

<http://spidr.ngdc.noaa.gov/spidr/docs/SPIDR.WSGuide.en.pdf>

[29] SPIDR SourceForge: <http://sourceforge.net/projects/spidr/>

[30] SPIDR web interface user's guide:

<http://spidr.ngdc.noaa.gov/spidr/docs/SPIDR.UserGuide.en.pdf>

[31] Taverna <http://taverna.sourceforge.net>

[32] THEMIS Data Analysis Software (TDAS) web site:

<http://themis.ssl.berkeley.edu/index.shtml>

[33] THEMIS NASA web site: [http://www.nasa.gov/mission\\_pages/themis/main/index.html](http://www.nasa.gov/mission_pages/themis/main/index.html)

[34] Triana <http://www.triana.co.uk>

[35] W3Schools SOAP: [http://www.w3schools.com/soap/soap\\_intro.asp](http://www.w3schools.com/soap/soap_intro.asp)

[36] W3Schools Web Services: [http://www.w3schools.com/webservices/ws\\_intro.asp](http://www.w3schools.com/webservices/ws_intro.asp)

[37] W3Schools WSDL: [http://www.w3schools.com/wSDL/wSDL\\_intro.asp](http://www.w3schools.com/wSDL/wSDL_intro.asp)

[38] Yahoo web services: <http://developer.yahoo.com/>

[39] Zhizhin, M., E. Kihn, R. Redmon, D. Medvedev, D. Mishin, Space Physics Interactive Data  
Resource- SPIDR. Earth Science Informatics, 2, 79-91, 2009.

## APPENDIX

This Appendix describes the actors that have been used to develop the Kepler-NGDC [14] workflows in this thesis.

### **ArrayAccumulator:**



**Figure 26: ArrayAccumulator actor**

The ArrayAccumulator actor takes an array as input and outputs the array elements as a string, with each element separated by the character specified in its parameter.

### **ArrayElement:**



**Figure 27: ArrayElement actor**

The ArrayElement actor takes an array and an index number as inputs and outputs the element present at the given index.

### **ArrayExtract:**



**Figure 28: ArrayExtract actor**

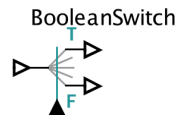
The ArrayExtract actor reads an array and outputs a subarray depending on the specified parameters. The parameters for this actor include sourcePosition, destinationPosition, extractLength and outputArrayLength.

**ArrayPlotter:****Figure 29: ArrayPlotter actor**

The ArrayPlotter actor takes an array of double values via its multiple input port and displays a plot of each array received as a separate dataset.

**ArrayToSequence:****Figure 30: ArrayToSequence actor**

The ArrayToSequence actor reads an array and outputs the values of the array elements as a sequence of tokens.

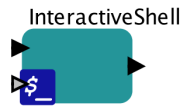
**BooleanSwitch:****Figure 31: BooleanSwitch actor**

The BooleanSwitch actor takes a value of any type and a Boolean token as inputs. Depending on the value of the Boolean token (true/false), either the trueOutput port or the falseOutput port is selected.

**ConcatenateArrays:****Figure 32: ConcatenateArrays actor**

The ConcatenateArrays actor reads any number of arrays via its multiple input port and outputs a single array containing the elements of all the input arrays.

### **InteractiveShell:**



**Figure 33: InteractiveShell actor**

The InteractiveShell actor displays a Unix command shell along with the string it receives as input and waits for the user to enter any commands. The entered commands are sent via its output port. We used this actor to create text-based forms that allow users to enter inputs to a workflow.

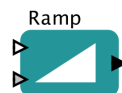
### **LookupTable:**



**Figure 34: LookupTable actor**

The LookupTable actor takes an array as input and looks up for an element specified by a given index number. This is very similar in function to the ArrayElement actor but differs in how they read the input array. In the LookupTable actor, the input array is specified via its parameter and takes the index number via its port, while the ArrayElement actor reads the array via its input port and the index number via its parameter.

### **Ramp:**



**Figure 35: Ramp actor**

The Ramp actor is similar to the ‘for’ loop in a programming language. Its parameters



include an initial value, the amount by which the value has to be incremented or decremented and the upper limit the value can reach. The actor outputs an integer value each time it fires.

### **SampleDelay:**



**Figure 36: SampleDelay actor**

The SampleDelay actor is used to output an initial value when a workflow starts up. This actor can also be used to prevent deadlock loops that occur when an output of an actor is the input to itself. In such cases, adding a SampleDelay actor fixes the deadlock loop issue.

### **SequencePlotter:**



**Figure 37: SequencePlotter actor**

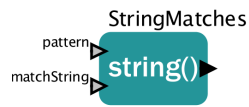
The SequencePlotter actor reads one or more sequences of values via its multiple input port and displays a plot of the data received.

### **SequenceToArray:**



**Figure 38: SequenceToArray actor**

The SequenceToArray actor reads a sequence of elements and wraps them up in an array. The length of the output array can be specified as a parameter.

**StringMatches:****Figure 39: StringMatches actor**

The StringMatches actor takes a string as input and matches it against a given pattern. The actor returns true if it matches, else false.

**StringReplace:****Figure 40: StringReplace actor**

The StringReplace actor replaces a part of the input string with the specified element.

**StringSplitter:****Figure 41: StringSplitter actor**

The StringSplitter actor splits the input string with the delimiter specified via its input parameter.

**StringToFloat:****Figure 42: StringToFloat actor**

The StringToFloat actor reads a string value as input and outputs it as a float value.

**URLToLocalFile:**



**Figure 43: URLToLocalFile actor**

The URLToLocalFile actor reads a URL and copies the file to the local machine. The output value is true when the actor is done saving the file.

### **VariableSetter:**



**Figure 44: VariableSetter actor**

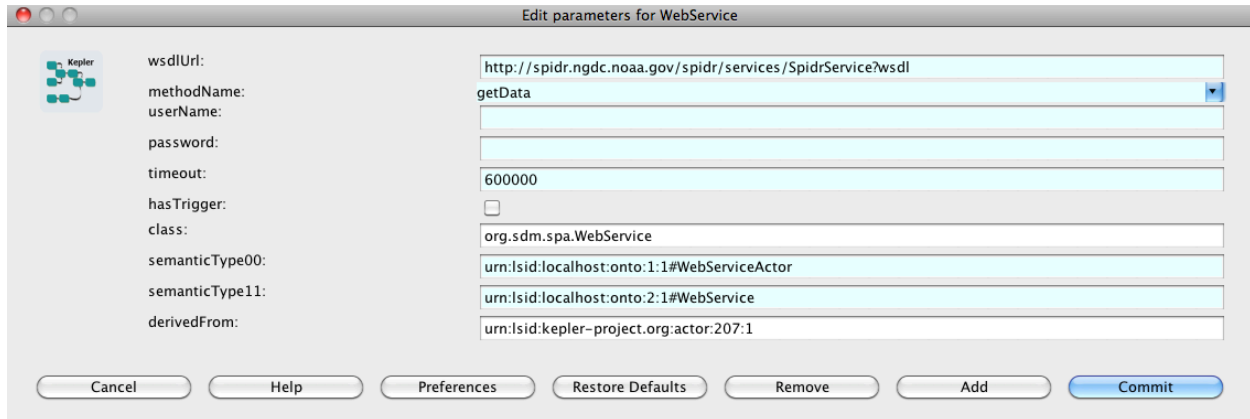
The VariableSetter actor sets the input value to the variable specified as its input parameter.

### **Web Service:**



**Figure 45: Web Service actor**

The Web Service actor accepts the URL of the WSDL file and the name of the operation to be executed via its parameters and executes the web service running on the remote machine and outputs the result. Once the user enters the WSDL URL, the actor automatically configures the input and the output ports.



Kepler

Edit parameters for WebService

wsdlUrl:

methodName:

userName:

password:

timeout:

hasTrigger: ☐

class:

semanticType00:

semanticType11:

derivedFrom:

Cancel Help Preferences Restore Defaults Remove Add Commit

**Figure 46: Edit Parameters for WebService actor**