A Knowledge-Based Approach to Scientific Workflow Composition

Russell McIver

A thesis submitted in partial fulfilment
of the requirements for the degree of
Doctor of Philosophy
in
Computer Science

Cardiff University
School of Computer Science & Informatics

December 2014

Declaration

This work has not been submitted in substance for any other degree or award at this or any other university or place of learning, nor is being submitted concurrently in candidature for any degree or other award.

Signed ……(candidate) Date ……

Statement 1

This thesis is being submitted in partial fulfilment of the requirements for the degree of PhD.

Signed ……(candidate) Date ……

Statement 2

This thesis is the result of my own independent work/investigation, except where otherwise stated. Other sources are acknowledged by explicit references. The views expressed are my own.

Signed ……(candidate) Date ……

Statement 3

I hereby give consent for my thesis, if accepted, to be available online in the University's Open Access repository and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ……(candidate) Date ……

Abstract

Scientific Workflow Systems have been developed as a means to enable scientists to carry out complex analysis operations on local and remote data sources in order to achieve their research goals. Systems typically provide a large number of components and facilities to enable such analysis to be performed and have matured to a point where they offer many complex capabilities. This complexity makes it difficult for scientists working with these systems to readily achieve their goals. In this thesis we describe the increasing burden of knowledge required of these scientists in order for them to specify the outcomes they wish to achieve within the workflow systems. We consider ways in which the challenges presented by these systems can be reduced, focusing on the following questions: *How can metadata describing the resources available assist users in composing workflows? Can automated assistance be provided to guide users through the composition process? Can such an approach be implemented so as to work with the resources provided by existing Scientific Workflow Systems? W*e have developed a new approach to workflow composition which makes use of a number of features: an ontology for recording metadata relating to workflow components, a set of algorithms for analyzing the state of a workflow composition and providing suggestions for how to progress based on this metadata, an API to enable both the algorithms and metadata to utilise the resources provided by existing Scientific Workflow Systems, and a prototype user interface to demonstrate how our proposed approach to workflow composition can work in practice. We evaluate the system to show the approach is valid and capable of reducing some of the difficulties presented by existing systems, but that limitations exist regarding the complexity of workflows which can be composed, and also regarding the challenge of initially populating the metadata ontology.

Acknowledgements

This work would not have been possible without the continued help and support of my family and in particular I would to thank my wife and children who allowed me the time to complete this thesis. Without the constant love and support of those closest to me I would not have been able to achieve as much as I have and I will be forever grateful to those who have helped me to where I am today.

I would like to thank my friends and colleagues at Cardiff University for their help and encouragement. My supervisors Dr. Andrew Jones and Dr. Richard White were integral to the completion of this work and gave me invaluable help and advice throughout the process, I would like to thank them both for their continued support during the research and the writing up period.

The work undertaken in this thesis was supported by a Microsoft Research Europe Studentship and I extend my gratitude to Microsoft and Dr. Rich Williams for their support in making all of this possible.

Contents

List of Tables

List of Figures

List of Abbreviations

API	Application Programming Interface
CAT	Composition Analysis Tool
CSSL	Composite Service Specification Language
DCI	Distributed Computing Infrastructure
GARP	Genetic Algorithm for Rule Set Production
GIF	Graphics Interchange Format
GPS	Global Positioning System
GUI	Graphical User Interface
HCI	Human Computer Interaction
HTN	Hierarchical Task Network
IWIR	Interoperable Workflow Intermediate Representation
JMS	Java Messaging Service
JPEG	Joint Photographic Experts Group
LSC	Live Sequence Charts
OWL	Web Ontology Language
PSE	Problem Solving Environment
RDF	Resource Description Framework
SCA	Service Component Architecture
SHIWA	Sharing Interoperable Workflows for large-scales scientific simulations in Available DCIs
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SPARQL	SPARQL Protocol and RDF Query Language
SSP	Structural Synthesis of Programs
SWS	Scientific Workflow System
UI	User Interface
URL	Uniform Resource Locator
WHIP	Workflows Hosted In Portals
WSDL	Web Services Description Language
WSML	Web Services Modelling Language
WSMO	Web Service Modelling Ontology
XML	eXtensible Markup Language

1 Introduction

Experiments and procedures involved in modern scientific research often involve the use of a combination of a number of tools and data resources in order to achieve the scientist's desired goals. In order to investigate their problem area scientists may make use of software which enables them to bring these resources together to perform analyses and simulations. A variety of challenges are presented when attempting to identify these resources successfully and orchestrate their use into a suitable sequence, including inconsistency in the manner in which such resources are described and accessed, the increasing scale and complexity of both the data and tools required, complications presented by the distributed nature of resource provision, and the considerable overheads involved in identifying the tools required and moving data between them. [1, 2, 5, 70, 71].

In performing their work scientists frequently need to complete complex sequences of operations ranging from accessing data sources to performing calculations and analyses. Successfully completing these tasks presents a number of difficulties the scientist must overcome. They must be capable of identifying and describing the overall goals and requirements of their experiments; translating these high level goals into a selection of appropriate tools; retrieving relevant data to provide input for these tools; and finally they must accurately determine the sequencing of these tools and data inputs to create a process that will satisfy their requirements.

The last decade has seen efforts to develop software that can provide an environment in which scientists can complete the task of joining up resources to create useful computational sequences [3, 5, 8, 56, 61, 66, 67, 68, 69]. These software tools, known as Scientific Workflow Systems (SWSs), provide the user with the capability to select and organise heterogeneous resources from multiple distributed sources, creating sequences of resources and operations that will achieve a user's desired goals. Altintas *et al.* [6] describe workflows as ideal to capture the typical processes undertaken by domain scientists when performing experiments. By providing a single user interface which allows scientists to combine data and analysis resources, they see SWSs as providing a benefit over the previous approach which would have required the use of multiple tools with the user manually transferring data between them. The SWS

orchestrates the execution of the tools and the necessary data transfers, allowing users to concentrate on the higher-level design of their analytical procedures.

Although current systems now support sophisticated tasks that use a wide variety of tools and data, a remaining concern lies with the problem of how to provide users with a means to identify the high level tasks required for their experiments, and subsequently how to support the user in developing this conceptual workflow into a complete implementation that will satisfy those requirements. This thesis focusses on the deficiencies of existing systems which could be alleviated by providing support for users to create workflows more easily, and on the ways in which knowledge relating to those users and the resources available for composition can be used to assist in this process.

At present the process of interaction with workflow systems is primarily driven by the user. The system gives access to a selection of available resources, and provides the functionality to allow users to create sequences of components by specifying the manner in which they are to be connected. At all times the requirement is for the user to select which resources to include and to specify the sequencing of those resources from the possibilities available within the system; limited support is provided to assist the user in identifying which combination of resources will achieve the result they desire. Some systems attempt to remove some complexity by limiting the choices available to user (e.g. Galaxy [97, 98]), however the requirement is still for the user to make complicated decisions about steps to take based on limited information. Currently this process of discovering which resources should be used within a workflow composition, and the manner in which those resources should be connected, has to be achieved by the user manually inspecting the resource metadata and descriptions that the system provides. [46, 70, 72].

A difficulty is that currently resource metadata provided by workflow systems tends to be limited to free-text descriptions of the function a resource performs and basic definitions of the type of data a resource produces or consumes. This metadata is of some benefit in assisting users with the discovery of individual resources which may be of benefit to their workflow; however it is not so helpful in determining how the resource might be used as an element of a solution to a larger problem. Furthermore, the metadata can be too “technical” to be accessible by the typical end user. Additionally

with the onus on the user to inspect this metadata themselves, current workflow systems provide limited support in situations where a user is unaware of the resources their experiment requires.

In this thesis we introduce a framework for maintaining more structured and extensive resource definitions, incorporating additional machine readable resource metadata beyond what is maintained within current workflow systems. We present an approach to workflow composition which utilises our resource definition framework to provide users with assistance during the workflow composition process as a means to address difficulties presented by current systems.

The approach presented consists of several key mechanisms through which relevant metadata can be stored; an ontology is described which defines a representation of workflow components, components from SWSs are represented as instances within this ontology and each is defined by a standard set of information. Additionally a simple data storage mechanism is utilised to retain metadata about a user's interaction with these components, recording the frequency with which connections are made between them. These mechanisms for storing metadata represent a knowledge base which can be utilised to assist users during workflow composition. To this end this thesis describes a set of algorithms which can inspect this knowledge base and the current state of a workflow composition to provide suggestions for how to proceed. Further to this we present an intermediate API which allows the suggestions which are drawn from the knowledge base to facilitate composition of workflows with resources from a number of SWSs, and a prototype user interface which demonstrates the feasibility of each of these elements.

1.1 Statement of Hypothesis

This thesis examines the hypothesis that

> *Composing new workflows with existing scientific workflow systems presents the user with challenges relating to the translation of high level goals into a concrete workflow, identifying required resources, and accurately specifying the sequencing and connection of components. These problems could be reduced if a structured repository of resource metadata were maintained from*

which suggestions could be provided to the user to assist in relating workflow resources to the tasks they perform, to discover appropriate components and to determine the workflow structure.

1.2 Aims and Objectives

In order to investigate the hypothesis that knowledge based assistance can provide benefits over the existing approaches to workflow composition, several aims were set for the research reported within this thesis.

Aims

- To investigate how resource metadata can be used to generate suggestions to assist users in creating workflows.
- To explore how a user interface could be developed to present this assistance to users.
- To determine how such assistance could be provided across multiple existing SWSs.

The approach to workflow composition presented in this thesis expands the role of resource metadata to be used not just as a reference point for users to discover information regarding workflow components, but also to act as a knowledge base which can be used to provide users with effective suggestions for completing their workflow composition. To this end a framework for the structured storage of resource metadata is presented, defining the manner in which relevant knowledge about each resource is maintained within the system.

The algorithms which generate suggestions for workflow composition are described, illustrating how the elements of resource metadata are utilised in order to provide useful, pertinent suggestions. The approach described also makes use of knowledge relating to a user's history of interaction with the system, and this is used to illustrate how incorporating information relating to previous compositions a user has created can assist in providing helpful suggestions during subsequent workflow composition sessions.

In order for this approach to be system independent, and for assistance to be utilised in the composition of workflows using the resources of current SWSs, the approach is designed as a software extension to existing scientific workflow systems. To achieve this, an abstraction layer is defined between the existing systems and the software extension, enabling the user to be provided with assistance to complete a workflow using the resources and structural capabilities provided by the underlying system.

Current scientific workflow systems are capable of composing workflows featuring complex structural elements such as conditional branching and provide detailed control over the execution of the workflows. The approach investigated in this thesis will focus on assisting the composition of relatively simple workflows, although consideration for extending the approach to include these more sophisticated features is given in the further work section.

Accordingly, the objectives below enumerate the main activities which were identified and have been undertaken in order to achieve the above aims:

Objectives

1. Develop a framework for representing knowledge about available resources.
2. Populate the framework with knowledge relating to selected resources to demonstrate how such information can be of benefit when composing workflows.
3. Create algorithms to generate workflow composition suggestions from metadata
4. Develop an API layer to enable the suggestion engine to sit on top of multiple existing workflow systems.
5. Provide a user interface to enable users to utilise assistance during workflow composition
6. Evaluate the proposed framework with respect to the hypothesis and in relation to other published work

1.3 Evaluation Approach

To evaluate whether the approach to knowledge based workflow composition assistance investigated in this thesis is able to overcome some of the difficulties presented by existing workflow composition approaches, several aspects are evaluated.

It should first be noted that while one of the aims listed in the previous section is to provide a user interface to present the user with assistance during workflow composition, this exists primarily as a means to demonstrate the other aspects of the approach presented. The usability of the user interface itself is therefore not a focus of the evaluation presented. Limited feedback on the user interface from a single user is presented; however the interface was not subjected to extensive user trials.

Turning, then, to what the thesis *does* seek to demonstrate, the evaluation approach presented in this thesis is designed to demonstrate that the proposed framework for maintaining knowledge relating to workflow resources is suitable for the task, and that the algorithms defined for generating workflow composition suggestions based upon this knowledge are capable of providing appropriate assistance to the user. In order to achieve this the evaluation focusses on identifying a selection of realistic workflow scenarios and, by walking through the steps involved in composing these workflows, demonstrating how the approach described is capable of supporting their composition. In addition these workflow scenarios are used to illustrate the difficulties which a user is presented with during composition when using the manual composition approach necessitated by an existing scientific workflow system; this is then used to highlight where the approach described in this thesis is able to provide assistance to overcome these difficulties.

To further assess how effectively resource knowledge can be utilised in the provision of helpful suggestions during composition, a measure of "suggestion quality" is defined and used to assess the quality of suggestions provided during the composition walkthroughs. This quality measure is also utilised to investigate the benefit which maintaining a history of user interaction with the system has on the suggestions provided.

1.4 Overview of Thesis

The remainder of this thesis is structured as follows:

- Chapter 2 presents a review of existing Scientific Workflow Systems, discussing a variety of different systems and approaches which have been developed to date. Additionally this chapter discusses other related work in the field of web

service composition, and gives an overview of the three main approaches to resource composition: manual, assisted and fully-automated composition.

- Chapter 3 gives an overview of the areas that have been explored in order to investigate the aims and objectives stated in Section 1.2. These aims are broken down into a set of formal requirements and the features of the system which has been developed to meet these requirements are introduced before being expanded upon in the remaining chapters.
- Chapter 4 describes the metadata framework and ontology that is utilised to maintain knowledge relating to available workflow components and to the user's history of interaction with those components. The rationale behind the choice of metadata elements is discussed and the approach utilised to populate the framework from the available knowledge relating to workflow components is described.
- Chapter 5 introduces the approach that is utilised to provide assistance to users during workflow composition. The manner in which metadata is used to produce useful composition suggestions is described.
- Chapter 6 introduces the API utilised to enable the assisted composition approach to be applied across a number of existing scientific workflow systems. This chapter describes how the API communicates with the underlying system to expose the relevant functionality to the extension.
- Chapter 7 provides an overview of the user interface which has been produced in order to illustrate the manner in which workflow composition through the means of suggestion based assistance can function.
- Chapter 8 presents the evaluation of the assisted approach to workflow composition discussed in the previous chapters, illustrating how it can successfully compose workflow scenarios, how it overcomes a variety of deficiencies with existing approaches, the extent to which component metadata is effective as knowledge for use in providing composition suggestions, and the benefit that a history of a user's interaction with the system can have on future interactions.
- Chapter 9 builds on the previous chapter to provide a broader discussion of the benefits and limitations of the approach to scientific workflow composition proposed in this thesis. The outcomes of the walk-throughs presented in Chapter 8 are discussed in greater detail, highlighting areas where the approach

improves on existing scientific workflow systems, as well as where the user may still encounter difficulty. A comparison with similar techniques in the field of web service composition, a discussion of the scalability of the approach and an overview of user feedback are also presented.

- Chapter 10 presents a summary of the conclusions and discusses potential directions for future research.

2 Background

This chapter establishes the context in which this research has been undertaken, providing an overview of a number of key areas of relevance; establishing the history and current state of the art with respect to scientific workflow systems and related fields such as web service composition, service component architecture, and program synthesis. Whilst a significant period of time has passed since the practical work and experimentation reported in this thesis was undertaken, this chapter includes references to recent literature which reinforces how the problems addressed in this work are still relevant today.

As outlined in Chapter 1 this research primarily aims to address identified concerns with the currently available workflow composition approaches. As such this chapter aims to introduce the field of workflow composition systems, discusses the problems which have been identified within the research community, and how these are attempting to be addressed at present, and the problems which remain unsolved. In addition a discussion of a number of related fields is provided to identify where developments from other fields could be of benefit to workflow composition and how such approaches have informed the research undertaken for this thesis. The related fields considered here are Web Service Composition, which focusses on providing users with mechanisms through which distributed web services can be interconnected to achieve more complex goals; Service Component Architecture, a model for developing Service Oriented Architecture applications which decomposes those applications into constituent components and defines the interactions between those components; and Program Synthesis, a field which aims to develop approaches through which the functionality of an application can be generated from a set of high level requirements.

2.1 Scientific Workflow Systems

Scientific Workflow Systems (SWSs) are a form of Problem Solving Environment (PSE) [42] and have emerged as a principal technology for enabling scientists to perform large scale tasks that involve the integration and coordination of resources [7]. These systems provide scientists with the capabilities to locate their required resources, ranging from simple data sets to complex analysis tools, and an environment in which

to compose these resources in such a way as to achieve their goals. One of the primary aims of PSEs is to provide a mapping between the abstract goals and objectives which users hold, the available workflow resources which can achieve those goals, and the underlying, concrete manner in which they are implemented [6, 13, 65]. This distinction enables users to concentrate on identifying and solving their domain problems without having to understand the complex computing tasks required to achieve this. Churches *et al.* [8] regard workflow systems and PSEs as being tools that enable the interaction between discrete components, and can provide the means through which those component interactions can be represented and reproduced. They further characterise PSEs as mechanisms for representing dependencies between services, either temporal or data driven dependencies; controlling constructs, such as conditional branching or loops; and scheduling and execution of completed workflows.

The origins of workflow systems are widespread and there have been a number of groups and institutions working on developing and providing workflow systems. As a means of composing and executing resources there is a wide range of situations where workflows can be applied, including the composition of Grid resources [8, 43, 46] and as a means to assist in the creation of composite web services [21, 45]. More specifically there has been a variety of domains and applications to which workflows have been applied: to support researchers in the life sciences [4], to support phylogenetic research [6], to enable the modelling and simulation of real-time systems [9], as software to assist signal processing [8], to support work in the field of chemical informatics [56], to assist the use of environmental sensor networks in the fields of terrestrial ecology and oceanography [57], to assist work in the field of neuroinformatics [61], and to support the development of aircraft design [58].

Despite being developed to support a variety of different application domains there are clear similarities between the motivations for developing each of the SWSs currently available, notably the desire to provide tools that enable users to perform their tasks without the need for in-depth, low level, computational knowledge of how these are to be performed. Oinn *et al.* [5] identify the need for systems that coordinate data and tools that are both complicated in their nature and widely distributed. This is especially so in fields such as Biodiversity Informatics where data handling raises a number of challenges based on the scale, complexity and heterogeneity of the relevant data [1] as

well as in locating and accessing data sources which are often highly distributed [2]. Goble *et al.* [10] describe the success of the Taverna system in enabling day to day activities of bioinformaticians to be performed more easily and quickly, reducing the amount of time they have to spend pulling together data and analysis tools and enabling them to achieve more.

Bisby [2] suggests a need for software that can locate and bring together this distributed data and present it in an environment that enables researchers from many institutions to make use of it. Altintas *et al.* [6] describe the typical process undertaken by domain scientists, performing analyses and experiments in many systems and manually transferring data between each system. They identify workflows as a means to capture this process and present scientific workflow systems as an essential tool to support the creation of those workflows. Deelman *et al.* [70] share this view, describing how modern scientific research involves an increasing level of distributed computational activity with scientists repeatedly moving data between local and remote tools to perform analysis or simulation. They describe how SWSs can assist in this process, enabling scientists to focus on the research goals they wish to achieve rather than on the management of the computational tools required to achieve those goals. Gaaloul *et al.* [11] suggest that the step-by-step methodology of the typical scientific process lends itself naturally to the idea of a workflow. Kim and Gil [12] further identify that users need to be able to specify their goals from a high level of abstraction - leaving the system to configure details, and that partial workflows containing descriptions of their services can help users navigate through the space of available workflows.

As they currently exist there are several main elements that make up a scientific workflow system. These are the resource repository, workflow composition environment, and the execution engine. The resource repository is primarily concerned with providing users with access to the tools and data that are available to be composed into a workflow; typically there are mechanisms provided which attempt to help the user to identify which components are required for a particular purpose. The workflow composition environment is the means through which the user creates their workflow by sequencing and providing inputs and specifying dependencies between the available components. This is usually performed in a visual environment where users physically drag and drop components and connections. Finally the execution engine (or

enactment engine) manages the running of completed workflows, providing feedback on this process such as displaying intermediate results returned by individual components or reporting problems encountered in accessing remote resources.

Workflow systems as described here have been available for a number of years. While most have initially been developed to support specific goals or application domains there has been a shift toward providing a generic environment in which users from many domains can access or import the components necessary to solve their individual problems. The Taverna system, for example, has continued to grow and has been successfully utilised in a wide range of fields including gene/protein annotation, proteomics, phylogeny, medical image analysis, statistical analysis, and cancer research [10, 73, 74, 75, 121]. This aim of supporting a wide array of users and tasks within systems has involved a large amount of research into the functional aspects of these systems, with investigation taking place into the languages used to describe resources and workflow compositions [45, 32], the manner in which more complicated workflow structures and means of execution can be supported [8, 3], and how workflow systems can be utilised to support interactions on the Grid [43, 44].

Despite this progress to extend the functionality of workflow systems and increase the distribution of domains in which they can be used, there has been ongoing identification of problems which exist with regards to their complexity and improving the process a user performs in order to compose workflows, Howe *et al.* [71] conclude that "workflow systems are very flexible, but even skilled programmers have trouble operating them effectively". When the research reported in this thesis commenced a number of key challenges were being highlighted; these were summarised in a 2006 workshop on the "Challenges of Scientific Workflows" by Gil *et al.* [7]:

- The potential gap that exists between a user's knowledge of the operation they wish to achieve within their workflow composition, and the knowledge required to create that workflow within a chosen SWS.
- The need to provide a means of interacting with the SWS which hides unnecessary complexity and allows the user to inform the system of their composition goals at a more abstract level.

- The need for further detail and semantic information to be included in the descriptions of available resources.
- The need to support composition not just of common, routine analyses or operations, but also to enable users to investigate more individual or abstract operations that may only be relevant to that individual.

These issues were also identified by Berkley *et al.* [13]. They argue that existing systems and approaches provide a barrier to enabling users of a less computer literate nature to successfully compose workflows. They further argue that whilst scientists are able to describe their intended tasks at a high level, outlining the data they wish to work with and the steps they would perform in converting this into their required results, current SWSs require them to perform many ancillary steps, the technical nature of which is often beyond the capability of such users. Berkley *et al.* also discuss the difficulty presented by the limited information made available to the user in order to identify which components are required to complete their workflow, proposing a need for greater semantic and contextual data to aid the user in making their selections.

Whilst the field of workflow systems has continued to mature, these problems remain largely unsolved today. McPhillips *et al.* [59] also identify this challenge presented by existing systems, describing how although scientists "have a very good idea of the analysis methods they wish to assemble", they lack the necessary computing skills to achieve these goals through the mechanisms which are currently available.

More recently, Bowers [77] discusses the continuing, considerable challenge presented by current systems arising from the lack of consistency in the data formats which are utilised by the components, and in the services which can be used within a workflow composition. This incompatibility has led to the need to provide intermediate components which can translate the output from one component into a suitable format to satisfy the input of another, adding an additional layer of complexity to the composition process and resulting in users of SWSs needing to spend additional time focusing on how their workflow composition will operate from a low level, rather than what it will achieve from a more abstract view. Bowers also describes how further work toward systems which provides better descriptions and definitions of the inputs and outputs of each component could allow users to more readily identify the means

through which two desirable components could be connected, or even allow the system to automate this process.

McPhillips *et al.* [72] also discuss similar issues presented by existing SWSs. In particular they consider the need to provide a means through which users can define their workflow composition from a more abstract level, and state that the wide variety of data formats and types which are present within existing SWSs presents a challenge to users wishing to correctly connect components with differing data types.

Whilst these difficulties have been readily identified within existing literature, research is continuing into the means to deliver effective solutions to these problems. As a result the field currently provides a selection of systems which, whilst technically advanced, are lacking in their ability to be effectively used by their target users. McPhillips *et al.* [72] conclude that in order for SWSs to become widely and effectively adopted they will need to be made “not only useful to scientists, but also directly usable by them”. This same view is shared by Záková *et al.* [105] who describe how, within fields such as bioinformatics, the array of data available and the algorithms required to convert that data into useful output have resulted in a need for tools which support real users, not just computer scientists.

This discussion has highlighted how the field of SWSs still presents a number of key challenges which reduce the ability of these systems to be used effectively by domain scientists. These challenges include the difficulty in translating the high level view a user has of the task they wish to perform into a set of steps that can be achieved by the system, the increasing volume and complexity of resources, data types and data formats with which users are required to work, and the need to hide the complexity of the underlying system in a manner which enables users to still achieve what they require. In Chapter 1 a number of aims and objectives were outlined, focusing on investigating a means through which resource metadata can be recorded and utilised to provide assistance to users during the workflow composition process. By achieving these aims this work attempts to address some of these challenges which remain within the SWS community.

2.1.1 Existing Scientific Workflow Systems

Kepler

The Kepler project [14] is an effort to provide a visual environment in which users can compose distinct elements known as actors with one another to enact complex tasks. Extending the previous Ptolemy II system [15], Kepler has strengths in its ability to provide a variety of execution models, achieved through the inclusion of specific workflow elements known as *directors*. The configuration of these directors enables the user to specify how the workflow will behave at run-time, including controlling circumstances under which a workflow will terminate and the number and nature of iterations to be performed. Kepler also enables the use of distributed components within workflows, where identified web services can be plugged into a workflow and handled in the same way as a local resource, as well as providing specific components that enable the submission of workflows as Grid jobs and for the querying of Grid databases [112]. Development of the Kepler SWS is continuing with the latest version (2.4) being released in April 2013. Recent improvements to Kepler have included the redevelopment of the software to function in a modular fashion, enabling the development of additional modules to extend the functionality of the system. A prominent example of such an extension would be the bioKepler module [91] which provides a selection of specialised workflow components and directors to facilitate the execution of bioinformatics tools.

Triana

The Triana project [16] started as a system to support analysis of gravitational wave detection [17] but has since developed to provide a robust set of tools to support a variety of scenarios. Both Kepler and Triana make use of a visual environment in which desired workflow components can be physically arranged and connected in order to achieve a user's goals. Triana supports the integration of web services into workflows and is similarly capable of accessing and composing Grid resources.

More recently the Triana SWS has been utilised within the SHIWA project [78] to demonstrate the capability of SWS to interoperate when supported by a suitable language to translate the representation of a workflow from one system to that of another. Further details on the SHIWA project are provided in Section 2.1.2.

Taverna

The Taverna Workbench [18], part of the myGrid [19] initiative to provide middleware for experiments in molecular biology, differs from Kepler and Triana in that whilst both of these systems provide a wide array of components which are executed locally on the user's machine, the resources which Taverna provides are primarily remote web services and therefore the integration of distributed components is a central aspect of Taverna's approach. Orchestrating the use of components that are primarily distributed highlights a key challenge of workflow systems, namely coping with a changing situation where resources may not be described exactly as expected or be available when required. Predefined workflows for both the Kepler and Taverna SWSs are available through the myExperiment project which aims to enable greater collaboration between scientists. [10, 76]

As with both Kepler and Triana, development for the Taverna SWS is ongoing, with the latest version (3.0) set to be released in 2014. This version represents a significant re-engineering of the software to make use of OSGi [113] - a platform to implement a component model into Java. Whilst this is a significant change to Taverna in terms of the manner in which the software is implemented, the mechanisms through which workflow composition and execution will be achieved within the system will remain largely unchanged, as demonstrated through the recent beta release of the software featuring a nearly identical UI to the current release. Further developments which are being explored by the Taverna developers include a system to provide an online service through which existing workflows can be executed, using either input data provided by the original developer or uploaded by the current user [102], and a mechanism to encapsulate sub-workflows into re-usable objects called "workflow components" with the intention that these be described in a manner which exposes the task they perform without requiring a user to be concerned with how that task is achieved [103].

Taverna has also recently been utilised as part of the BioVel project [122]. BioVel (Biodiversity Virtual e-Laboratory) is a biodiversity research project involving a wide range of partners which seeks to support the work of scientists by providing a suite of well defined, reliable web services which can be used to perform the research necessary to support decision making around ecological problems such as ecosystem

alteration, changes to the distribution of species, and ultimately species extinction. The project aims to provide a comprehensive e-Laboratory to enable biodiversity scientists from many countries and projects to contribute to common goals. The project has targeted the use of an existing workflow tool, Taverna, as it is easy to introduce new components to the system in the form of web services; these services can then be curated centrally so that all can benefit from their use and ongoing development. Their re-use is also promoted by this arrangement, so scientists can achieve consistency in results of repeated experiments, taking advantage of the mature nature of the product. To this end the project has developed a biodiversity catalogue [123] where users can register their own web services for use by others, discover new web services, annotate and improve the descriptions of existing services, and monitor the development and availability of services.

Despite the ongoing development which occurred with these systems the primary functionality of each SWS remains the same, with the user interface and processes used to create workflows having evolved little from the point at which the work described in this thesis was begun.

Additional SWSs

The work undertaken for this thesis focusses primarily on improving the workflow composition process associated with the three systems described above: Kepler, Triana and Taverna. However, it is pursued in a manner designed to ensure that the resulting benefits will be potentially applicable to additional workflow systems available.

Additional workflow systems include Galaxy [97, 98], a web based system primarily focused on genomic research. Galaxy provides a “wizard-like” interface for specifying analyses as well as a more traditional graphical interface for manipulating workflows. Similar to the use of myExperiment within the Taverna project, Galaxy provides a mechanism through which users can share and publish completed workflows, but in Galaxy this is supported by the “public pages” section of the system. The recent Tavaxy project has developed a system to enable the integration of Taverna and Galaxy workflows [99].

Another workflow system of note is VisTrails [100]. As with other systems VisTrails allows the creation and execution of workflows in a similar fashion; however VisTrails has a particular focus on provenance [101], with data recorded about aspects such as the steps taken during the creation of a workflow. This makes it easy for users to revert to a previous version of a workflow, or to perform comparisons between two versions of a workflow.

The WINGS/Pegasus system [116, 117] adopts an approach which enables users to define *abstract* workflows which describe the activities which they would like to perform, independent of the specific resources which will implement those activities. The system then attempts to translate this abstract description into a *concrete* workflow at runtime using AI planning techniques to identify the resources to use.

These additional SWSs demonstrate that the community is aware of the difficulties that are presented by traditional manual composition systems, and that work is ongoing to attempt to address these difficulties. Approaches such as the "wizard-like" approach offered by Galaxy have benefits in reducing the perceived complexity of the underlying system and presenting the user with a more simplistic and guided mechanism through which to define their goals; however, difficulties regarding the knowledge that a user must have of the task they wish to perform and resources which are available remain. The WINGS approach of enabling a user to define *abstract* workflows and have the system translate this into a *concrete,* executable workflow, offers an approach to overcoming the knowledge gap, however the approach is limited in terms of the interaction a user has over the translation process and the system's ability to work in concert with the user to guide the composition toward the desired outcome. These elements however do provide insight into useful starting points from which to attempt to resolve the larger remaining usability problems that we investigate in this thesis.

2.1.2 Related Work

Whilst the development of many SWSs is still ongoing, and research is still being performed into their usage and functionality, recent work has also focussed on solving the related problems of how researchers can best record information about the computations and analyses which they have performed using workflow systems, how

such information can be shared with others in order to be either reproduced or further built upon, and how approaches can be provided to enable the composition of workflows to be more collaborative.

Workflows Hosted In Portals (WHIP) [35] provides an environment to enable researchers to collaborate on and share workflows via web portals. Harrison *et al.* argue that the current means through which users interact with workflow systems and the way those systems have been designed and implemented pose problems that prevent them from being successfully utilised on the Grid. WHIP provides plug-ins to enable interaction between workflow systems and web portals that will allow information to be exchanged. An additional aim is to provide further semantic information for a workflow being shared that will enable its purpose and functionality to be maintained.

Similar to the WHIP project is myExperiment [10]. Here the developers of the Taverna system desire to see workflows as more than just "one-shot" experiments; rather a workflow should be something that is shared, re-purposed and generally used for aiding others. De Roure *et al.* [60] position myExperiment as a facility to support the wider lifecycle of scientific workflow design and use, claiming that such support is a necessity if widespread adoption of scientific workflows is going to be achieved. The myExperiment project provides an environment in which workflow users and creators can communicate with one another and where workflows can be distributed, shared and worked with collaboratively. The approach also intends to enable links with other related tools so that tasks such as the archiving of results in repositories and the remote execution of workflows can be achieved.

Continuing this trend of establishing workflows as tools which can be re-used and re-purposed for future use, the SHIWA project [78] is aimed at supporting interoperability between existing workflow systems. The project has developed a number of key elements in order to achieve its goals - a repository through which workflows can be stored and shared with others, an environment to enable the execution of those workflows across a range of Distributed Computing Environments (DCEs), and mechanisms to support the combination of workflows from multiple SWSs to perform larger operations.

The SHIWA Workflow Repository is similar to myExperiment in that it provides a repository through which existing workflow compositions can be uploaded and shared amongst research communities. Users can search the repository to locate workflows which they may wish to use. Workflows located through the SHIWA repository can be imported into the SHIWA Simulation Platform; this acts as an environment through which users can execute workflows or where existing workflows can be combined to create meta-workflows. Repositories such as those provided by SHIWA and myExperiment attempt to solve similar problems to some of those addressed in this work, the difficulty users face in creating their own workflows. However, where SHIWA and myExperiment aim to reduce the impact of this by encouraging re-use and evolution of common workflows which others have already created, the approach described in this work focusses on improving the ease with which such workflows can be developed in the first instance.

In addition the SHIWA project has also developed the IWIR (Interoperable Workflow Intermediate Representation) language [79, 80] as a means to enable workflows developed in one SWS to be edited using another, facilitating cross-SWS collaboration during workflow composition from users who may otherwise be unable to assist each other, or at least would need to learn to use a common SWS in order to do so. IWIR breaks workflows down in to two representations, abstract and concrete, where the abstract representation maintains information about the general structure of the workflow and the concrete representation extracts the details required to perform each of the operations defined within the workflow. Through this approach the details of a workflow composed in one SWS can be transferred to another, suitably modified SWS which will present the same workflow within its own workflow format for further editing. The concept of abstract and concrete workflow representations is utilised within the assisted workflow composition approach presented within this work; however here rather than being a means to support interoperability of SWSs this representation is used to bridge the gap identified between a users conceptual view of the tasks they wish to achieve and the concrete implementation required to achieve that within a given SWS. The concept of a workflow interoperability language such as that presented by SHIWA could potentially be utilised to enable the assisted workflow composition approach presented in this thesis to be used in conjuction with multiple SWSs

simultaneously and is a potential route through which this research could be developed further.

In addition to providing mechanisms through which users can publish and share their workflow compositions, those developing workflow tools have identified that in order for others to make best use of the workflows which are being shared a mechanism is required to record a more complete set of information relating to workflows and the manner in which they have been developed and used. De Roure *et al.* [83, 84] describe how the myExperiment project recognised the need for users to be able to associate the workflow which they were sharing with ancillary information such as example input data or papers discussing the results obtained from executing the workflows. The myExperiment platform therefore developed additional capabilities to allow users to upload collections of files in the form of "packs".

Whilst the ability for users to bundle additional files and information with the workflows they share is of benefit in assisting others in understanding the nature of those workflows, or in reproducing previous results, Bechhofer *et al.* [85] argue that in order to enable more complex forms of reuse additional information is required in the form of metadata describing the relationships between each of these additional files; such representations of the information relating to scientific research have been titled "Research Objects". As introduced by Bechhofer *et al.* [82], Research Objects are a mechanism to assist in the sharing and publication of scientific research in order to improve the ability of others to reproduce results or build upon existing work. Bechhofer *et al.* describe how a number of issues with the way research has been traditionally published have reduced the benefit that can be gained from the results of that research, particularly in an era where research is becoming both increasingly collaborative and predominantly computerised.

The Workflow4Ever project [81] exists as a continuation of the ideas explored within myExperiment and aims to utilise Research Objects to provide a mechanism through which the workflows developed during scientific experiments can be stored and annotated in such a way as to enable others to successfully repeat their execution at a later date. A key element explored by the Workflow4Ever project is the definition of workflow-centric Research Objects, a model which seeks to define how various

elements of relevant information relating to workflows can be maintained. As described by Belhajjame *et al.* [81] this information includes the workflows themselves - the computational tools used within them, the versions and authors of those tools, information relating to the provenance of results obtained from executing the workflow, the data which has been used during execution of the workflow etc. Belhajjame *et al.* argue that providing a standard approach to recording such information will enable workflows to be more effectively shared and reused within the scientific community and will enable the results of experiments to be more reliably reproduced.

Whilst each of the systems described in this section provide a useful addition to the current SWS ecosystem, they are focussed on either providing facilities through which existing workflows can be shared with others, or mechanisms to enable users to collaborate on the composition of workflows irrespective of their preferred SWS. Whilst these approaches do not directly address the problems identified in Section 2.1 regarding the challenges facing users when performing the initial composition of workflows using existing SWSs, it should be borne in mind that for many users all that they require may be achieved by an existing workflow and therefore repositories of existing workflows remove the need for such users to directly tackle composition. However it is the situation where an existing workflow does *not* solve a user's problem, or provide a useful starting point from which to work, which the approach taken in this thesis aims to resolve.

2.2 Web Service Composition

The field of Web Service Composition is closely related to that of Workflow Composition. Both focus on enabling users to achieve complex goals through the identification and orchestration of a selection of discrete data and processing resources. A primary difference with web service composition is that there is more variety of implementation approach in the field of workflow composition, with each SWS offering a unique approach to defining the activities performed by a resource, the input and output requirements it exposes, and the manner in which they are to be connected. In the field of web services this information is either provided in a uniform manner through the WSDL specification [124] for SOAP [125] based web services, or using a fully-defined API definition for RESTful services [126, 127]. Similarly web services are designed as

“platform agnostic” software tools that can be utilised by any application that makes a valid request. In contrast, the resources provided by a SWS tend to be tightly coupled to that system: if a user wishes to make use of a resource from the Kepler workflow system then, unless the provider of that resource has provided an equivalent, independent application or web service, they must use the Kepler SWS to access its functionality. In this situation a user will have confidence that the chosen resource will operate as expected each time it is used, as it is implemented and executed locally. By contrast, in the field of web services the resources are remote and potentially constantly changing, resulting in greater concerns around resource availability and the ability to re-use previously constructed compositions in an environment where services may be removed or altered. Of course it should be noted that it is entirely possible to construct workflows within SWSs such as Kepler using entirely web services and therefore this situation would be replicated.

However, these differences are primarily related to the implementation of these respective fields; from a high level their concepts are closely aligned. Like workflow creation, web service composition requires the identification of necessary components, the understanding of how those components can be effectively connected, and the suitable sequencing and configuration of the components in order to satisfy a user's goals. As such, techniques used to assist or automate the composition process in the field of web service composition are worthy of consideration for the potential benefit they could bring to workflow composition.

A web service is “a software system designed to support interoperable machine-to-machine interaction over a network" [20] and represents a shift away from the user driven view of the web, since services can provide functionality that can be accessed and initiated by other programs without requiring the interactions of a human user. The current approach to utilising these services involves the publishing of services with a suitable Broker which will then provide requesting machines with the required information for how to access and interact with the service.

Beyond providing services as stand-alone entities to be used in isolation, over time the field has developed to introduce the concept of Complex Services, an enhancement to the existing approach to enable services to be composable. This means that they can

be combined with other services to form a composite service which can achieve more complex goals without the need to define a dedicated service to perform these [22]. Casati *et al.* [21] describe the way that web services have moved from being simply a mechanism that can provide a "one-shot" service to an end user and instead now aim to provide value-added composite services by composing existing web services. However similarly to the problem of workflow composition there are challenges presented by attempting useful interoperation between web services. To achieve a user's goals it can still be necessary to create manual, *ad-hoc* compositions of web services, a process that takes both time and a large amount of low level programming to achieve success [23]. The increasing number of web services, and the volatile nature of their description and availability, means that any manual composition approach is severely limited in its capability to create the best possible solution to a user's problem. Medjahed *et al.* [24] describe how composing web services is often a frustrating task, requiring much low level programming and a trial and error approach.

In order to make the task of web service composition easier, an important development has been the development of languages which can be used to more completely describe the services themselves. For example, Bartalos and Bieliková [111] identify how improved approaches to web service composition must rely on more than purely syntactic descriptions of the available services.

One such approach to extending the description of services is the OWL-S language, designed to enable the semantic description of web services [25]. The desired outcome of this semantic description is to enable the automatic discovery, invocation and composition of web services. OWL-S intends to provide answers to three fundamental questions that users, or machines, may have regarding web services: What does the service provide? How can it be used? and How does one interact with it? The language describes three main elements of services to answer these questions - the Profile, Model, and Grounding. The idea of adding a layer of semantic description to the definition of services is one which could also be applied in the field of SWSs. Enriching the description of available workflow components with greater semantic information could enable the creation of approaches to automating the composition of those components. In addition, since many scientific workflow systems already allow for the

inclusion of web services within a workflow, these service descriptions could provide immediate benefit to users during workflow composition.

In addition to OWL-S a number of alternative approaches have been proposed for associating semantic information with existing web services. The Web Service Modelling Ontology (WSMO), for example, has been proposed by Lara *et al.* as an approach to overcome a number of limitations identified within OWL-S, such as the inability to effectively describe the relationship between a web service's input and output [114]. WSMO provides four main elements to enable the semantic description of web services; Ontologies, Goals, Web Service Descriptions and Mediators. These are described by Lara *et al.* as follows:

- **Ontologies.** They provide the terminology and formal semantics for describing the other elements in WSMO.
- **Goals**. These elements provide the means to specify the requester-side objectives when consulting a Web Service, describing at a high-level a concrete task to be achieved.
- **Web Services**. They provide a semantic description of Web Services, including their functional and non-functional properties, as well as other aspects relevant for interoperating with them.
- **Mediators**. These modelling elements are connectors that resolve heterogeneity problems in order to enable interoperation between heterogeneous parties.

OWL-WS [26] is an extension to the OWL-S language from the NextGRID project. NextGRID aims to provide a workflow-centric model of execution on the Grid that can adapt to handle different workflow policies. The OWL-WS language has been developed so workflows can be described from both an abstract and concrete level and to enable the handling of semantic information necessary to enable a concrete workflow to be derived from an abstract representation during run time. Abstract workflows are defined without specifying bindings to specific services so that these can be bound at run time through the use of semantic task descriptions which can be utilised for identifying a service that provides desired functionality. OWL-S is used as a base

language as it provides the capability to describe the control and data flows required to model workflows and it has become an accepted standard for describing services.
Whilst this section has shown that there are key differences that exist between the fields of web service composition and workflow composition, there are several aspects which are of interest. By providing a strong focus on the definition of services, the tasks they perform and the manner in which they are to be interacted with, the field of web service composition has made several advances; developing descriptions that enable systems to automatically compose services, as well as promoting the interoperability of services from difference providers. These developments provide a useful starting point for the present research in addressing the remaining concerns regarding scientific workflow composition.

2.2.1 Web Service Composition Tools

The ability for web services to be connected together in order to achieve more complex goals has driven the development of a category of tools which are designed to assist users with the process of creating these composite web services. In this section we discuss a number of these tools and suggest how their capabilities can influence approaches to the design of tools for composition of scientific workflows.
eFlow [21, 27] is an example that has been developed to enable users to compose and enact composite services. This approach represents composite processes as a graph made up of service, decision, and event nodes. Whereas a service node represents the invocation of a simple or composite service, decision nodes describe alternatives that could be used as well as the rules that control the flow of execution through the graph, and finally event nodes describe the messages and events that are sent and received by services. Interesting aspects of the eFlow system are that it enables nodes to remain "un-bound" until runtime when parameters provided by the user and a broker service is used to select a compatible service, ensuring the most suitable services available at that time are utilised. eFlow provides consistency rules and migration semantics to ensure that alterations to a previously defined process do not result in errors and to ensure that the user is aware of the impact that these changes may cause.

The idea that the binding of individual services to a process should be performed at runtime in order to ensure the best possible selection of services is further discussed in

[28]. Zeng *et al.* propose an approach whereby users define a composite service consisting of:

- Specific "elementary" web services - concrete implemented services with no dependencies on other services
- Composite web services - a "service" implemented by connecting multiple existing services
- Web service communities - a collection of services which achieve the same goal but offer differing non-functional properties (QoS parameters, reputation etc.)

If the user has constructed a composite service including web service communities then the specific web services which are executed for that step are decided upon at execution time. Zeng *et al.* propose a selection of criteria which, combined with constraints provided by the user, can identify the most effective services at the particular point in time when the composition is executed. The authors describe the challenge in selecting the right services to use at design-time, the need for mechanisms to cope with this, and the continually changing landscape of available services. Zeng et al. argue that previous approaches have not produced suitable, in-depth criteria with which to assess the choice of services to bind to a process model and have failed to take into account "global constraints" that a user may wish to impose upon the whole process model. They present a model of service quality that characterises the non-functional aspects of the available services, including: execution cost, execution duration, reputation, reliability, and availability. These aspects are then used to drive the selection of services at runtime.

The idea that an approach to web service composition should be QoS aware has been identified by a number of groups. Alrifai *et al.* [118] propose a technique which takes into account a variety of QoS factors such as the response time or reputation of services, indicating that the source of information for such factors comes from a variety of sources: provided by the service itself, based on past experience, or even sourced from the community of users of services. The approach presented breaks down the problem of identifying the optimal set of services to achieve a user’s goals into smaller sub-problems, on the assumption that solving each independent problem in isolation is more manageable than solving the global problem in one step. A similar approach is

presented by Jiang *et al.* [119], QSynth, which also focusses on achieving users' quality goals through decomposing the overall optimisation problem into sub-problems which can be solved more effectively.

Whilst QoS aspects such as these are not a primary concern of the workflow composition approach which has been developed for this research, as our approach is primarily concerned with ensuring the workflow composition which is created is complete, executable, and achieves the user's objectives. It is nevertheless relevant: our approach is designed to guide users towards a completed workflow composition which achieves their goals, as such it is pertinent to consider the quality of the composition which the user has been guided toward, for example is it the fastest or most efficient solution, and has the user constructed a suitably modular and extendable composition that could benefit others.

The idea of associating available resources with various quality criteria is applicable to both scientific workflow composition as well as web service composition, the idea that storing more non-functional information regarding components available in workflow systems could be used to assist in the selection of appropriate components during the composition process. The nature of such information stored would need to be altered to reflect the different requirements which a user has when selecting components for inclusion in a workflow when compared to those of identifying web services. For example whilst information about the reliability and availability of a resource is essential in the field of web services, where those services are distributed, often changing, and potentially available from multiple sources, this can be less applicable to the scientific workflow composition scenario where many compositions will be constructed of locally accessed resources. However quality metrics would still be beneficial for SWS components, relating them to details such as the number of significant figures to which a component produces output or the duration which a component takes to complete a calculation. Information such as this regarding scientific workflow components would centre more on the capability of that component to perform the operation which the user requires and enable that user to determine if both their functional and quality requirements were met. However, as stated previously, workflow compositions within SWSs can and regularly do make use of web services, so any improvement in the

description and definition of web services is something which could be of additional benefit to SWSs.

2.3 Service Component Architecture

Service Component Architecture (SCA) represents an approach to application development that provides a method for creating the components which make up an application, a means to determine how those components interact with one another to achieve the application's overall goals, and represents an approach to development where applications are abstracted from any specific target platform and can be deployed on multiple architectures [62]. In this sense SCA is similar to both Web Service Composition and Scientific Workflow Composition: each has the primary aim of making it possible for a user to orchestrate a sequence of components, each performing an individual activity, into a larger construct that achieves some overall goal. These similarities make SCA an area of interest when considering means through which scientific workflow composition could be improved, techniques and developments within SCA around areas such as problem decomposition, abstraction of goals from implementation, and the interaction between components could potentially be of benefit within the field of scientific workflow composition.

At its most basic level SCA is a framework that defines how applications can be developed based on the Service Oriented Architecture (SOA) [104]. SOA itself is an approach to application development which focuses on breaking software down so that each area of functionality is encapsulated as a single service. It is these services which can then be combined to achieve the overall functionality of a system. This approach has various benefits. In particular, by reducing software to a number of services that represent the individual activities performed it is possible that those services can be readily re-used within many other applications. The provision of associated metadata that describes both the function of a service and the data that it operates on enables services to readily exchange data with one another. Additionally by providing well decomposed services with clearly defined interfaces the underlying complexity of an activity can be abstracted away from the user, allowing them to focus on their required outcome rather than the implementation of individual services.

The main aspects of SCA can be broken down into four areas, with each area defined within its own specification document:

- The assembly specification [63] - this defines how individual components are connected and packaged as services at a level that is abstract from their implementation.
- The component implementation specification - this describes the manner in which components are implemented within a specific programming language, different specifications exist for how this is achieved in a number of languages [92, 93]
- The binding specification - this describes how the service(s) that a component provides can be accessed. Again a number of different specifications exist to define how this is achieved across a number of technologies such as SOAP for web services [94] or using the Java Messaging Service JMS [95].
- The policy framework specification [96] - describes how non-functional requirements can be associated with services, defining factors such as what form of authentication is to be used for communication between a service provider and requester.

These specifications define how SCA components are defined, implemented and connected to form an SOA application. There are a number of elements which make up a SCA component, and these are similar to the representations of components within SWSs.

Each SCA component provides a number of ports which represent either the services which that component provides or the dependencies which it has upon other services, described as the services and references of the component. The services provided by a component are logically similar to the input and output ports of a SWS component. They define the operations which that component can perform, the input that is required to achieve that operation, and the output which will be provided on completion. The references of a component allow that component to describe the services which it requires from other components in order to achieve its own goals. For example a component may represent a Banking service and provide a number of operations such as the ability to login, check account balances, etc. In order to achieve the login goals

the component can define references to a database service which performs the login details lookup.

By providing a rich model to describe the interface between components, the services which a component provides, and those which it requires, the SCA approach provides an environment where the discovery of services which can interact with one another is readily identifiable. This is something which current SWS lack to a certain extent as the information relating to component interfaces is less detailed. However whilst the definition of interfaces and contracts between components is of benefit in identifying compatible services, and in achieving the goals of SCA such as de-coupling the definition of a service from its implementation and promoting the easy re-use and replacement of services, the process of constructing the overall application is still a task that requires the user to be aware of the goals they wish to achieve, the breakdown of services that can fulfil those goals, and the means through which those services are sequenced.

2.4 Program Synthesis

An additional area of research which has overlapping goals with that of workflow composition is the field of program synthesis. A long term aim for some researchers and developers has been to develop systems which allow for the automatic creation of program code from a defined set of specifications or requirements; indeed, approaches such as those by Green [86] and Manna and Waldinger [87] have been proposed from as far back as the 1960's. If successfully implemented, program synthesis would have benefits in reducing the effort involved in the manual development and testing of software. Users would only need to provide fully defined requirements from which the system could automatically generate "correct" code. Similar to the work presented in this thesis, an aim of program synthesis is to support users who can express their requirements at a high level, but lack the technical skill, time or inclination required to formally implement those requirements, generating complete programs from these high-level requirements.

A number of approaches to synthesise software have been explored since the early works of Green and of Manna and Waldinger. One key technique, and one which was the focus of much of the early work on program synthesis, is to follow a mathematical or

logic-based approach, where the specification of the desired outcome is expressed by some form of logic [106, 107, 108]. Whilst these techniques are able to synthesise logically complete, executable programs from the input provided, they are often limited in the complexity of the programs which they can generate, limiting the extent to which more complex program structures can be used, and are often targeted at the synthesis of programs within a restricted application domain.

An alternative approach to program synthesis is based on generating programs from examples [88, 109, 110]. Here the user specifies the output that is desired from a given input; using this information the system explores the space of possible operations which could translate from the input to the output. Gulwani [88] acknowledges a potential weakness in the use of examples as a means to specify a user's desired outcome in that often this results in an "under-specified" or ambiguous goal. Gulwani proposes a number of interaction models which can help resolve this issue, including enabling the user to "test" the program provided by the synthesis - if this program generates output that the user does not desire then they can submit this new combination of input and output back to the system in order to generate a refined program. A second model proposed is that where the system is able to identify more than one program which can perform the transformation between the given input and output it will provide the user with a *distinguishing input.* This is an input which results in different outputs when transformed by each of the generated programs. By allowing the user to specify what the desired output from this distinguishing input is, the system can then refine the selection of programs it has generated. By repeating this process several times the system will eventually produce a single program which generates the required output for each of the inputs tested. The relative complexity of scientific workflow components and their configuration means that adopting such an approach for workflow composition could prove problematic but using this mechanism to synthesise discrete sections of a larger workflow is a possibility. In addition the notion of a dialogue being presented between the system and the user to work toward developing a program that achieves their desired goals is an aspect which will be explored in relation to SWS in this thesis.

Whilst the primary aim of the work explored in this thesis, an improved approach to the composition of scientific workflows, is not directly related to the automatic synthesis of programs from a given specification, there are parallels to be drawn between the fields

of program synthesis and workflow composition. Both aim to reduce the complexity involved in generating a solution which achieves the user's goals, seeking to develop a mechanism through which users can specify their goals from a level which is abstracted away from the low level implementation which will actually those goals. The purpose of automatic program synthesis is to generate single program which achieves a user's specified goals. Similarly, the approach explored in this thesis seeks to guide users toward a solution based on knowledge of both their requirements and the decisions they have made thus far. Similar knowledge of users, requirements, and the capabilities available to achieve those requirements are necessary in both of these cases.

2.5 Approaches to Scientific Workflow Composition

Currently there are a number of approaches available for the process of locating and composing a workflow from a set of resources, varying from the simplistic – allowing the user access to a list of available tools or services and providing means for them to manually connect or sequence them– to the more sophisticated approaches that aim to either fully automate the process or to provide guidance and assistance as the user creates their sequence of components.

2.5.1 Manual Composition

Existing workflow systems such as those described in Section 2.1.1 have developed out of a desire to support a task for which dedicated software did not previously exist; that is the composition of local and distributed resources. As with any initial iteration of a new approach the aim was to provide something that could support most of the basic needs of the users.

Additionally these initial approaches had limited users and composition scenarios in mind – potentially only dealing with a relatively small number of resources, and assuming that the users already knew what they wanted to do with them. To this end the requirements of an approach to workflow composition were simply to enable users to directly select and connect the resources they had already been passing information between manually.

Manual workflow composition systems, such as Kepler, Triana and Taverna, traditionally function by simply providing users with access to a selection of components which they can insert into a workflow (e.g. by placing them on a "canvas"), and require them to manually sequence and configure each of those components. As the demonstration system described later in this thesis builds upon the mechanisms through which these systems work, further detail of the approach to manual composition within each of these systems is given in Chapter 6.

With the growth in the number of available resources, their increasing reliance on diverse and incompatible data formats, and the desire to provide workflow systems that are more generic and less focussed on an individual group of users, the ways in which workflows are composed using current SWSs have been identified as a hindrance to non-specialist users in adopting scientific workflow systems [7, 59, 71, 72, 77]. Similarly the field of web services has seen a continual and rapid increase in the number of available services which users may desire to interconnect to solve larger problems and here too the need for a more sophisticated approach to composition has been identified [12,111,114].

2.5.2 Automated Workflow Composition

The goal of automated workflow composition is to remove the need for the user to directly specify which resources they desire to incorporate into their workflow and the manner in which they are to interact, in order to provide a system where the user is concerned with the “what” aspects of their workflow instead of the “how”.

To completely eliminate any specification of implementation details by the user during the composition process is a challenging aim, and requires a greater level of sophistication in the definitions of available resources, the information that controls how those resources can interact, and the implications and outcomes of utilising any individual resource. Additionally a completely automated approach would require an entirely new interaction method between the user and the underlying system.

There are two main challenges to the fully automated approach – how to offer a means for the user to provide a suitably rich, high level description of the task they want to

perform, in a format that can be easily interpreted by the system, and how to provide greater clarity and depth in the definitions of available resources in order to enable the system to identify those required to carry out the user's specified task. For example current workflow systems provide resource definitions which will be of benefit to a user, such as textual discussion of a tool's function and the motivation for its development. However, to be of benefit to an automated system these would need to be translated into simple, parseable statements to inform the system of a tool's overall function or requirements for usage.

A common approach that has been taken to solve the problem of automating service composition is to make use of the existing field of planning systems research [115]. Using this approach a user's specification is translated into a planning problem and then a suitably modified planning system is utilised to generate a solution to the problem as an orchestration of concrete services. Similarly, Blythe *et al.* [29] identify the need for automated systems for the discovery and composition of grid resources, rather than relying on an individual's knowledge and time to solve the problem, and they present an approach to composition that makes use of a heuristic based planning system to guide the selection of services and resources.

The approach taken by Blythe *et al.* incorporates a knowledge base consisting of data about the problem area and the available grid resources - such as knowledge about how components operate, the characteristics and availability of files and resources, and policies that may exist to control the access to such files or resources. The planning system can then access this information during the composition process in order to search for a solution that matches a user's requirements, to ensure that such a solution is feasible given the current resources and environment available, and to enable the system to adapt to changes in the environment.

A similar approach by Wu *et al.* [30] makes use of the SHOP2 planning system [31] to solve web service composition problems. SHOP2 is a hierarchical task network (HTN) planning system and the authors suggest that the similarity between task decomposition in HTNs and process decomposition in the OWL-S process ontology will enable the system to be modified to solve workflow composition problems. By viewing services as actions with requirements and outcomes, and providing a suitably described

objective the planning system is able to evaluate the available services and compose a solution. The difficulty with this approach is in how the user actually specifies what they want to achieve with web services and how the system translates this into an acceptable planning problem, and whether this can be done without the user already knowing all of the detail that would enable them to perform the task manually. The authors have provided only limited information on the manner in which the problem they wish the planner to solve is submitted to the system, stating simply that their implementation provides *"An interface which lets users specify the request for a service".* [32]

McIlraith *et al.* [33] present another approach to web service composition by providing a means for semantic markup of services in such a way as to make them machine-understandable and "use-apparent", as well as enabling these services to be supported by an automated composition approach. The intention is to provide semantic information not just for the available services, but also for the users who may be composing the services, as well as for "agent procedures" - essentially a repository of previously composed services made available for reuse. McIlraith *et al.* promote the use of ConGolog, a "high level logic programming language", to search the available space of services that could be composed to meet a user's given requirements. ConGolog makes use of situation calculus [130] to identify the consequences of introducing any one service into the composition and therefore decide on which services are appropriate for use with the existing services. The decision to store semantic information about users' constraints and preferences is interesting as this could lead to better decision making on which services should be suggested for composition.

Medjahed *et al.* [24] propose a technique to aid in composing services by providing both a means to discover whether services are composable, and a system to automate the composition of those services. Composability rules are introduced to assess whether available services are compatible. These rules consider the semantic and syntactic features of the components, e.g. message types, functional descriptions etc. Medjahed *et al.* describe WSDL descriptions of services as insufficient for the purposes of the semantic web as they only provide syntactic information about the services they describe. In addition they propose the use of an ontology to represent service definitions that extend this syntactic information with more semantic features. For

example, syntactically a service has a name, binding and operation, but the ontology introduces extra information - a description, category and purpose. Automatic composition is achieved by allowing users to provide a high level specification of the processes desired from the composed services utilising CSSL (Composite Service Specification Language). The system inspects the user's description and its repository of available services in an attempt to locate services that provide the desired functionality. The composability rules are then used to test whether the set of located services is able to be composed successfully. If several potential plans are generated the system uses quality parameters to decide which generated plan is best.

2.5.3 Assisted Workflow Composition

A more recent development in both workflow and web services composition that aims to support the process beyond the basic manual approach is to begin providing systems that can guide or assist the user through that manual composition process. This assistance can be provided in a number of ways, including the provision of suggestions to the user based on the current state of their composition, the flagging up of potential mistakes that have been introduced into the workflow such as the connection of incompatible components, and the identification of requirements still needed to configure or complete a workflow. We shall describe these in the current subsection.

The challenge with an assisted approach is two-fold – how to define the resources and composition process in such a way as to enable the provision of suitable assistance, and how to avoid continually providing the user with unhelpful assistance.

Sirin *et al*. [34] describe an approach taken to allow the semi-automatic composition of services by inspecting the semantic properties of services and presenting acceptable services to the user at each step of the composition process. They argue that whilst we may still be some distance away from achieving fully automated composition of services that is both accurate and useful, the ability to describe available services has reached a stage where it is possible to enable a semi-automated composition process where a human user is involved in the decision making process. The approach currently aims to support two applications of service composition - the translation of French to English and the use of sensor networks. The identification of suggested services for the user's

consideration is achieved by performing matchmaking based on the OWL-S semantic properties of services. An exact match is defined as where two services have the same OWL class for a property, and a generic match as where they have matches based on one being a subclass of the other. Generic matches are further ranked based on the distance between classes within the hierarchy. The approach provides further opportunity to "filter" the list of suggestions returned by the system where the user can choose to rank suggestions based on non-functional attributes of the services. The authors argue that there is a barrier in overcoming the differences between the concepts people use to think about both their problems and the services available to them, and the concepts that computers use to interpret these, and that their approach helps to overcome this barrier.

The idea of identifying useful suggestions based on the "closeness" of available services properties is one which is expanded upon in the research presented in this thesis. This work explores whether recording and inspecting additional data about the available services than was considered by Sirin *er al.* would make it possible either to eliminate further services from the list of suggestions, or to provide improved ranking making it easier to identify which are the “best” options. Also this work seeks to demonstrate that by taking into account information such as past use of services and user profiling, the suggestions could be tailored even further.

Kim and Gil [12] have developed the CAT (Composition Analysis Tool) system as another means to overcome challenges encountered in composing services through the analysis of semantic properties of those services. The CAT approach defines both a Task and a Domain ontology to describe the available services based on their implementation details as well as more semantic data. The composition process allows users to select abstract components from the task ontology and then guides them through the process of specialising these abstract components. Alongside this specialisation the CAT system also has defined requirements that dictate when a workflow is "complete" or finished, and based on these requirements will prompt the user with suggestions that will take the current workflow closer toward being "complete". The approach defined by Kim and Gil has limitations in that the suggestions provided by the system could quickly become unwieldy, as there may be hundreds of possible routes to take or specialisations to make. By incorporating more metadata,

including checks against previous activity and by ranking the suggestions given, it could become possible to guide the user more effectively toward the "right" decisions.

As discussed in Section 2.1.1 the WINGS/Pegasus system offers another approach to assisted workflow composition. This system provides the capacity for users to express their high level requirements as an abstract composition, and the system will then use this information to generate possible implementations of that abstract composition. Additionally this approach makes use of a "workflow template" concept whereby users can choose to start expressing their requirements using an existing template as the starting point, for example if they are wishing to perform an analysis using a common method but wish to specify some additional requirements specific to their needs.

A further assisted composition approach is presented by Cerezo *et al.* [120]. Their approach which was developed after the practical work was completed on this thesis follows similar themes as those which will be explored in this work. Cerezo *et al.* propose a technique whereby available resources are defined within an abstraction hierarchy based on their function; these abstract resources can then be composed into a *conceptual* workflow by users. Following this, semantic descriptions of resources can be used to map a conceptual workflow into an *intermediate representation*, that is a workflow containing both abstract and conceptual resources. Finally this intermediate representation can be converted into a concrete workflow which can be executed using an existing SWS. Differences between the system described in this thesis and that proposed by Cerezo *et al.* include that the semantic information considered by their approach is static and based on the activities fulfilled by a resource and the input and output requirements it presents, whilst this information is also considered in the approach described in this thesis, it is expanded to include additional information such as a user's past interactions.

2.6 Summary

The previous sections have provided an overview of the existing state of scientific workflow systems and composite web services, with a focus on the different approaches which are being explored within these (and related) communities to the problem of how to compose resources to achieve more complex goals. As discussed in the introduction to Section 2.1, despite differences in the manner in which resources are

described and implemented between the fields of scientific workflows and web services, there is sufficient similarity in their overall goals for the approaches to composition being explored in one field to be of benefit to the other related ones. Following several years of development, scientific workflow systems have progressed from early tools supporting the composition of a small number of resources to achieve the goals of a small set of users, to a stage where users have a choice of several mature systems which can be used to identify, sequence, and execute a wide selection of resources to support work across many domains. However we have shown that challenges remain to enable these systems to provide further support for the successful composition of workflows when users are unaware of the resources required to complete their workflow, the overall operation they wish to complete with their workflow, or simply the single resource that will complete their workflow, as also identified in the literature. [7, 13, 37]

In addition several emerging approaches in the fields of assisted and automated composition have been discussed in this chapter. The approaches described constitute the first steps towards formulating a set of techniques that can be utilised in order to remove the burden of workflow composition from a user. These approaches seek to enable a user to simply indicate to the system the outcomes that they wish to achieve, along with any other relevant restrictions or considerations they wish to be taken into account, and the system will then use this information to generate a suitable composition that meets these criteria. These approaches investigated have identified some of the key challenges and developments which must be made in order for a service or workflow composition system to support either assistance or automation, namely the need for a suitably rich system with considerable enhancements to the descriptions and definitions relating to available resources to enable the system to identify those which can achieve the user's goals, and finally a new mechanism through which the user interacts with the system in order to fulfil their composition requirements.

The move toward assisted and automated composition approaches seeks to overcome several of the remaining issues identified with existing scientific workflow systems, reducing the level of knowledge that a user must have about the implementation details of the workflow they wish to compose. However, these new systems still have significant limitations, and the work in this thesis seeks to address these. For example,

many approaches have focussed on developing a whole new resource definition, composition and execution framework in which to operate, losing the benefit of the considerable work that has gone into existing systems, along with the history of templates of example compositions which users have created for those systems. This thesis aims to utilise composition assistance when working with the resources and sequencing capabilities as provided by existing workflow systems, concentrating on Triana, Taverna and Kepler. As such, those systems which define an entirely new approach to resource definition and sequencing are not a suitable starting point for the present work; however ideas from such systems which have been described in this chapter have informed the direction taken by this work, including the idea of defining workflows from a high level abstract perspective, the incorporation of further semantic information into workflow resource description, and the idea of providing assistance to the user to explore the space of possible workflow compositions which could achieve their goals.

The literature surveyed in this chapter has demonstrated that the field of scientific workflow systems is an active area with many projects working to solve the complex issues that users involved in workflow composition and usage encounter. Whilst many problems have been solved or are seeing ongoing research to alleviate them, such as the provision of systems to successfully connect resources and allow them to interact with one another, the definition of languages and frameworks in which to provide new resources, the establishment of mechanisms through which completed workflows can be shared with others, and the development of systems to allow multiple SWSs to interact with one another, there are remaining issues which this thesis seeks to address.

These remaining problems centre primarily around the process of composing the workflows themselves; whilst projects such as SHIWA and myExperiment can provide considerable benefit by directing users toward pre-constructed workflows, there remains a need for users to be able to create these workflows in the first instance. It is in this composition process that we still identify problems which restrict users from successfully constructing workflows, problems such as an inability to translate their high level goals into the set of low level steps which are required by the SWS, difficulty in locating the components which will achieve their overall goals, difficulty in identifying

how those components should be correctly sequenced, and a lack of assistance provided by existing workflow composition systems to reduce these difficulties. The computer-assisted composition approach presented in this thesis is aimed at addressing these remaining problems.

By extending the existing Triana, Taverna and Kepler systems to incorporate a computer-assisted composition approach the user can obtain the benefit of working with a mature workflow sequencing and execution framework that has been used across a number of domains and is capable of providing access to a large pool of resources, whilst still obtaining the benefits that are provided by utilising an assisted approach to composing those resources.

3 Requirements and Design of New Workflow Composition Features

In Chapter 1 we enumerated a number of aims which this work seeks to achieve: to provide a framework through which resource metadata can be used to assist in workflow composition, to develop a UI to demonstrate how this assistance may be presented to users, and to establish a mechanism through which such assistance can be offered across a number of existing SWSs.

This chapter derives a set of requirements from these aims, and outlines the design of the new SWS features which have been investigated in order to achieve these aims, and to test the hypothesis described in Chapter 1. In subsequent chapters we provide information about how these features were implemented and present the results of testing the effectiveness of these features.

3.1 Overview

As discussed in Section 2.1, SWSs have been in development for a number of years, with this development primarily focusing upon extending the functionality and capabilities of these systems. This continued development has provided a selection of SWSs with powerful capabilities for co-ordinating and executing a wide array of tools and data elements. As such, the purpose of this thesis is not to reject or directly modify these existing approaches to achieve this functionality but instead to attempt to address the related problem of how users interact with these systems to make use of their functionality.

As set out in Section 1.2 the primary aim of this work is to provide a mechanism through which users of SWSs can be afforded assistance during the process of workflow composition, in order to overcome the identified drawbacks with the way in which this process is achieved in existing SWSs. The processes through which composition is achieved in the existing SWSs used in our work are largely similar, involving the user manually selecting and sequencing components from a list provided by the system, although the implementation obviously varies significantly. This approach places a relatively high requirement on the knowledge which a user must possess before they

can successfully compose a workflow. They must be aware of the specific components which provide the functionality they require and the manner in which those components interact - including the types of data which each component works with and how data can be usefully and meaningfully passed between them. Additionally these systems do not provide a sufficiently exploratory approach to workflow composition to support users who may not initially be certain of the goals they wish to achieve.

Whilst each of the existing SWSs provides the user with a degree of support in the form of information regarding available components (such as a textual description of their usage, or some basic information about the format of data which they produce or consume), the degree of assistance provided is limited, and users must discover this information for themselves during the composition process. The specific nature of the assistance provided by each SWS is described in greater detail in Chapter 4, where we identify the additional knowledge required to support the level of guidance provided by our approach to assisted workflow composition, and further information regarding the composition process that is used by each system is provided in Chapter 6, when describing a new API which can interact with these systems.

In light of these issues the goal of this work was to explore ways through which scientific workflow composition could be made a less challenging problem for users, particularly those with limited experience working with the software. Whilst the final list of aims described in Chapter 1 focusses on a number of elements which have been explored to satisfy this goal, the early stages of this work placed more direct focus on the idea of developing a new UI for workflow composition, exploring whether altering the manner in which the user interacts with the underlying workflow engine to compose and execute their workflow could result in a system which overcomes some of the challenges identified in Chapter 2. Investigation into aspects of Human Computer Interaction (HCI) such as the role of intelligence and automation in UIs [37, 38] identified that the idea of a "Workflow Composition Wizard" was an approach which could potentially be explored as a means to simplify the composition process. Due to the limited capacity for a wizard approach to support the richness of workflow composition possible within existing SWSs, this approach was discarded in favour of exploring the benefit which a more interactive, knowledge-based approach to composition assistance could provide. However, a wizard based approach to workflow

composition is still an idea which merits exploration, and could potentially be used in conjunction with the knowledge-based approach described in this thesis.

As a result in order to overcome the issues identified in composing workflows in existing SWS the approach presented in this thesis is to provide the user with context-sensitive assistance during the composition process. As the core sequencing and execution functionality provided by existing SWSs is of a mature standard, the intention is to provide a self-contained extension to these systems, which provides this assistance when using the resources of the existing systems to compose workflows. This assistance is to be provided in the form of suggestions for how the user can progress their composition. Additionally the approach aims to enable users to specify their composition as a sequence of high level goals, and then to provide assistance in order to translate this into a complete, executable workflow. The suggestions which the system offers are provided in three forms:

1. Suggestions for components which the user could add to their composition.
2. Suggestions for how to specialise the "abstract", high level components which the user has selected into concrete, executable components, and
3. Suggestions for connections which they could create between concrete components which are already present.

As stated previously these techniques will enable us to provide an assisted approach to facilitate composition of workflows utilising the components and underlying capabilities of a number of existing SWSs.

3.2 Requirements

Through discussion with Dr Rich Williams, the contact from Microsoft Research Europe who were the sponsors for this work, as well as supervisors from within Cardiff University , a number of requirements were identified. It is desirable to satisfy each of these requirements if such an approach to assisted composition is to be achieved in a manner which is applicable across multiple existing SWSs. The system must be able to achieve the following:

1. Exist as a separate entity to existing SWSs
2. Interact with the workflow composition functionality provided by multiple SWSs
3. Inspect the current state of a user's composition
4. Hold suitable knowledge of the available workflow components in order to facilitate suggestions
5. Generate useful suggestions to present to the user based on their current progress
6. Interact with existing SWSs in order to facilitate the execution of workflow components.

The approach to assistance which is to be provided by the system is intended to operate in interaction with multiple existing SWSs, enabling users to be offered assistance whilst still working with the resources and composition capabilities provided by those systems. As such, requirements 1 and 2 define that the system must be independent of any particular SWS, but able to interact with them to achieve key functionality such as the insertion and connection of components and the defining of component properties. It will be necessary to expose the underlying functionality of the existing systems in such a way that the development of an extension to these systems can be abstracted from their individual implementations.

As the aim of the system is to provide the user with assistance in progressing their workflow composition, requirement 3 is essential as the system must first be aware of the progress, if any, the user has made thus far. To this end the system must monitor both the components which the user has inserted into their workflow composition, and the manner in which those components have been sequenced and interconnected.

Requirement 4 is the basis from which useful suggestions can be provided to the user. By providing the system with knowledge of the purpose, configuration requirements, and data types involved with each component, the system can identify which components may be of relevance to the user in his or her current context.

If the system has knowledge of both the state of the user’s current workflow composition, and the characteristics of the components which are available within the SWS, requirement 5 is that the system should have a suitable set of mechanisms for

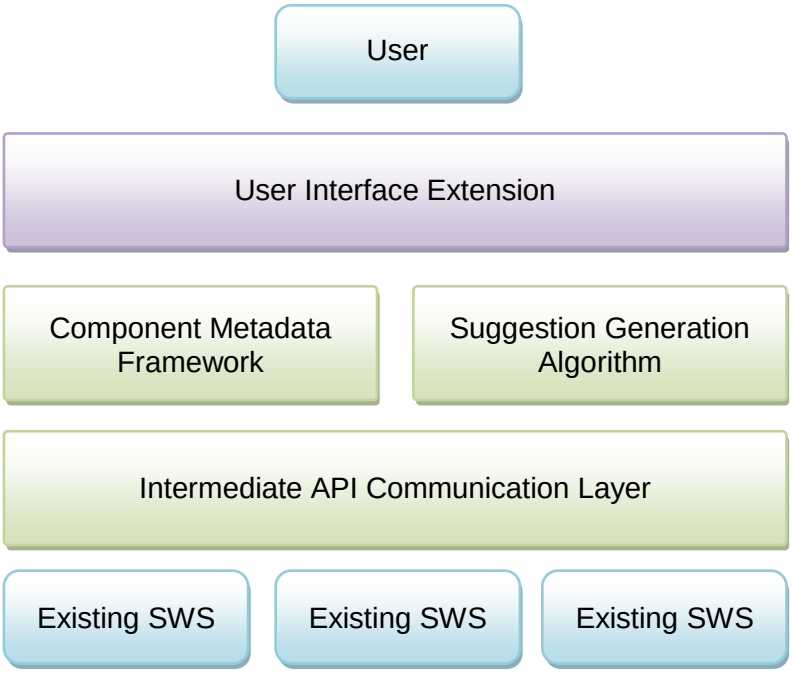
converting this knowledge into suggestions for components which the user could include within their workflow, or connections which could be implemented between existing components.

Finally requirement 6 focusses on enabling the workflow compositions, which have been created by following the suggestions made possible by requirements 1-5, to be successfully enacted by the execution engine provided by each of the existing SWSs. The results obtained will therefore be identical to those that would be achieved if the workflow were constructed in the original SWS itself.

3.3 Key Features

In order to satisfy these requirements a number of key features are required - a clearly defined framework for storing knowledge about workflow components, a set of algorithms to generate composition suggestions using this knowledge and information about the current state of the user's composition, a user interface which presents the user with these suggestions during the composition process, and an API which defines how a system such as this UI can interact with existing SWSs in order to achieve the functionality required to compose and execute workflows. Figure 3-1 shows an outline of this system architecture.

Figure 3-1 Overview of System Architecture



3.3.1 Component Metadata Framework

There has been considerable research into the mechanisms through which assistance can be afforded to users during their interaction with workflow composition systems, with significant challenges being identified with respect to the amount of benefit each approach provides. van Nimwegen *et al.* [47] claim that, although the common approach of reducing the options available to a user at any given moment (the "greying-out" of choices) can have benefits, it is also possible for this practice to hinder a user's ability to work effectively as users may not be aware of why options have been removed, and may fail to discover key functionality that the system can provide. This can reduce their ability to learn how to work with the system independently. To this end a particular focus of the new approach presented here is upon ensuring that where guidance is provided to the user it is of genuine benefit, and that the user is not prevented from exploring options other than those the system directs them toward.

Birnbaum *et al.* [48] suggest that the ability of an intelligent system to provide assistance to the user is dependent on how well the system has managed to model the task which is being performed and the domain in which this it is being used. In order for

a SWS to provide the user with information which is both relevant and helpful it must have a clear understanding of the workflow components that are available to a user, in terms of both the function which they perform, and the configuration and sequencing which enables their use, as well as knowledge of the goals the user wishes to achieve so that these can be matched to the capabilities of these components.

Accordingly, the approach taken in this thesis is to develop an ontology which stores a selection of metadata relating to each individual component. Whilst existing SWSs store a certain amount of information relating to the components which they provide, the purpose of the ontology is to contain a more extensive representation of the components, relating each to the purpose for which it can be used, as well as expanding on the more functional information which current SWSs provide. Furthermore, defining a consistent set of metadata elements which are required for each component enables this information to be more effectively queried in order to help identify those components which may be of benefit to the user.

Storing this information in an ontology allows for a more structured representation of the properties of each available component than what is currently available. For example, rather than simply providing a textual description of the purpose of each processor, as existing SWSs do, we can develop a *hierarchy* of "component tasks". This hierarchy can describe the functionality of components both at a high, application-orientated level, such as the fact that a component is involved in visualisation or data modelling, and at a lower level, capturing specific details such as the particular type of visualisation technology or modelling algorithm which is used. As components may be used across several application areas or domains the hierarchy should not preclude a component having multiple "super classes". By storing components within this hierarchy we can enable users to specify their workflow composition in terms of their high level goals by creating an abstract workflow from these high level components, then use the task hierarchy to identify the low level components which could achieve those goals. In addition, the hierarchy allows us to begin to establish relationships between components, identifying those which are involved in similar areas of usage and which may be suitable to use together. A similar technique is used to provide a hierarchy of "data types", defining specific characteristics of the data which is produced or

consumed by each component, beyond the basic information (String, Integer, etc.) information which is made available in existing SWSs.

An additional kind of information which is stored within the ontology relates to patterns of usage: the frequency with which the user interacts with components, storing how often a particular user uses each component, and in addition how often they create a connection between a pair of components.

Storing this additional information about each individual component can help users to identify which components they could use within their composition as well as assisting in identifying the correct manner in which to connect those components. In addition, imposing a defined structure for the metadata allows the knowledge stored within the ontology to be machine understandable. It is this feature which enables this metadata to be used to generate suggestions during the composition process.

3.3.2 Assisted Workflow Composition

In order to support the user during the composition process the approach taken in this thesis has two main areas: firstly to enable users to specify their workflow in terms of high level abstract functionality as well as by directly selecting the required components, and secondly the ability to provide suggestions to indicate the next step which users could take, in terms of components which they could introduce into their workflow, or connections which could be made between those components already included.

By relating components to a task hierarchy, users who are unsure of the specific component they require can begin by inserting abstract components from this hierarchy into their workflow. The system can then query the ontology to identify which components implement the high level functionality the user has identified and provide suggestions for which to use.

The algorithms which generate suggestions are based upon two primary factors: the current state of the user's workflow composition, and the knowledge contained within the ontology about each component available within the SWS. By storing the component metadata within an OWL (Web Ontology Language) ontology [49] it is

possible to use a variety of techniques to inspect this information; for example the system can use the SPARQL query language [52] to discover which components have compatible connections with those which the user has already selected.

In addition to the algorithms for generating suggestions the system also incorporates a mechanism for ranking the suggestions provided. A number of factors are utilised to rank suggestions, including how closely related elements are within the ontology. For example when suggesting components which could be connected to one another, if one candidate matches the exact data type required it will be ranked higher than another which only matches at a more abstract level of the hierarchy. Another aspect which is used to rank suggestions is the user's history of interaction with components; if they have regularly connected two particular components together then this would become a more highly ranked suggestion in the future.

3.3.3 User Interface

The primary aim of this research is to explore the potential benefit that the provision of suggestions during workflow composition can have on the user's ability to construct their desired workflow. However, in order to demonstrate this benefit a basic prototype user interface has been developed. This interface is not intended to be a feature complete workflow composition environment and has not undergone significant development of usability analysis, the aim is simply to illustrate how the knowledge contained in the metadata ontology could assist a novice user during composition.

The interface will enable the user to compose workflows as they would using an existing SWS, giving them access to a selection of components and providing a space in which to organise the relationships between those components. However the interface will also provide the user with a means to select abstract components from the component task hierarchy, these can be included within the composition like any standard component.

The user interface will begin providing the user with suggestions as soon as their first component is added to the composition. Suggestions will be provided to the user across three categories:

- Specialisation – how abstract components can be specialised
- Addition – additional components that could be added
- Connection – how existing components can be connected

Each of the suggestions which the system provides will be presented as an ordered list, with each entry including a description of the reason why it has been included by the system.

3.3.4 Intermediate API and Implementation

As previously discussed, the purpose of the approach taken in this work is not to replace existing SWSs, but to provide the user with an improved mechanism for working with the capabilities which these systems provide. To this end, a key feature of the approach is the ability for the assistance provided by a suggestion based UI to be made available across a number of existing SWSs.

In order to achieve this, an intermediate API has been created which defines a set of calls which the new UI can utilise to achieve set functionality provided by the existing SWSs. By providing such an API the implementation of the UI does not need to take into account the differences of each of the underlying SWS, so the same approach to workflow composition assistance can be utilised regardless of which SWS is being used. An implementation of this API has been created as an intermediate layer between the UI and those existing SWSs, if changes occur to an underlying SWS, or if support for a new SWS is required, it is only the implementation of this API which will need to be updated.

The benefit of developing such an API is also not limited to enabling the assisted composition UI which is explored in this work; the API could be used to support a variety of extensions to existing SWSs' functionality.

Specification of the API requires the identification of a number of key elements of functionality which are required to be exposed from the existing SWS, such as the mechanisms for listing the available components, creating new workflow compositions, adding components to a composition etc. The API implementation must translate calls

for this functionality from the UI into the required form to achieve that functionality within the SWS which is being used.

A factor which makes the implementation of such an API reasonably straightforward, once specified, is that each of the existing SWSs which are being targeted is implemented in the Java programming language. This helps reduce the complexity of the API and ensures that it can be kept as minimal and transparent as possible to any subsequent extension which makes use of it. We provide details of the API in Chapter 6.

3.4 Summary

This chapter has outlined the requirements for an assisted approach to workflow composition. We have discussed what is required if a system is to be developed which can present the user with helpful suggestions throughout the workflow composition process, and how such assistance could be provided when composing workflows within a number of existing SWSs. This chapter has introduced a number of new features which will satisfy these requirements: a framework for storing metadata about available workflow components, a mechanism for using such metadata to generate composition suggestions, an interface through which such assistance can be provided, and an API which enables this assistance to be provided across a number of existing SWSs.

The following chapters will look in greater detail at each of these features, explaining their motivation with respect to the operation of existing SWSs, and providing further information regarding their design and implementation.

4 Component Metadata Framework

This chapter provides a detailed description of the framework which has been developed in order to store relevant metadata about workflow components available within the SWSs Kepler, Triana and Taverna. As outlined in Chapter 3 this metadata is recorded in order to be able to generate suggestions which can support users during the workflow composition process. The following sections outline the extent to which such support for users during the composition process is provided in existing SWSs, introduce the key elements of metadata which we intend to record about workflow components, describe how an OWL ontology has been developed in which to record this metadata, and finally outline how this ontology has been populated with information about components from the existing SWSs.

4.1 Overview

In order for a SWS to provide assistance to the user during the task of workflow component composition the system must have knowledge of the elements involved in that process, primarily the capabilities and requirements of use of the components that are to be composed. By maintaining a repository which records both the basic requirements for each component and more semantically rich information (e.g. the relationships between components and the purpose or function of those components), suitable inspection of this repository can provide useful insight to assist users in creating workflows.

A major role of the (meta)data within the repository is to act as information to bridge the gap previously identified in Chapter 2 between the user's knowledge of what they wish to achieve with their workflow and the knowledge required to successfully compose that workflow within an existing SWS. The traditional approach taken by SWSs is to provide the user with a list of components with which to compose their workflow, and the mechanisms (such as a workflow construction “canvas”) through which they can sequence and assign properties to those components.

This primarily manual approach to workflow composition places considerable requirements on the user if they are to successfully compose a workflow, specifically

requiring them to have detailed knowledge about both the workflow they wish to construct and the SWS they are using to construct it. In more detail, using this traditional approach the user is required to have knowledge of the following:

- The complete workflow goals or process they are trying to achieve

  Current SWSs provide little support for the exploration of possible solutions to a user's problems, or helping the user to formulate his or her problem in terms of a workflow. Without knowledge of exactly what they wish to achieve within their workflow composition the user will struggle to proceed, as current SWSs place the onus on the user to specify all the details of the workflow they wish to create. It is already recognized (e.g. by Deelman and Gil [90]) that this knowledge is something which many users may not possess before beginning their composition

- The specific components that are required at each step

  As discussed in [7], even if the user is able to describe the goals of the workflow composition they wish to create, and thus overcome the first difficulty described above, this is still not enough to enable the composition of their workflow. The user must still be able to identify each individual component which is required by the SWS to complete composition of the workflow.

  Berkley *et al.* [13] describe how this problem is compounded by the lack of clarity regarding the naming of components and the lack of relevant metadata made available to assist the user in identifying which components are useful. As a result the user could potentially be left in a situation where they do not know which components are required to proceed and the information made available to them is insufficient to help to overcome this difficulty. The authors provide a specific example: the user may be inclined to assume that a component entitled "interpolator" can perform a generic interpolation operation where in fact it is only to be used for interpolation of a specific set of data. They additionally state that the problem commonly occurs in which the name of a component is abbreviated, for example in this case "interpolator" becoming "int", which could further confuse the identification of that component's applicability.

- The precise manner in which those components need to be sequenced

  If the user is able to identify the components which are required, there is still a further challenge presented by current SWSs – understanding the manner in which those components must be connected to successfully achieve the user's goals. This requires knowledge of the order in which these components must be sequenced, as well as the specific input and output elements of each component which need to be connected to each other. As discussed by Qin and Fahringer [89] this is especially difficult when working with SWSs where many component input and output ports use the same, generic types (e.g *string,* file). Deelman and Gil [90] also make similar observations.

  The information regarding components which is available within current SWS can be of assistance in overcoming this difficulty. However, it has been noted [7, 13, 34] that this information is not of sufficient detail or semantically rich enough to genuinely benefit the user; in addition this approach requires users to seek out the information themselves.

  Again this problem is also made more difficult by the fact that although users may indeed understand the task they wish to perform, their understanding may be at a more abstract level than that required by the SWS [7, 59, 77, 90]. The conceptual view of a workflow that a user has in their head may compress the steps to be achieved by several components into one step, or *vice versa* – in this way there are substantial challenges in identifying and structuring these components.

This high level of knowledge required for a user to successfully compose a workflow using the traditional manual approach is one of the central problems which this thesis addresses. Existing SWSs provide a limited range of means through which the user can attempt to overcome the problems identified above. These are primarily based on providing descriptions or profiles of the components that are available. The user can inspect this information in an attempt to overcome their composition challenges. The following subsection describes the information that is made available to the user in each of our chosen scientific workflow systems, as well as detailing any other techniques provided in order to help users overcome the identified challenges.

4.2 Assistance Provided by Existing SWSs

As described in Chapter 2, the approach explored in the present thesis focusses on assisting composition within three existing SWSs: Kepler, Triana and Taverna. As we will go on to consider in detail in Chapter 6, workflow composition within each of these identified SWSs is achieved through similar means. The user is provided with a mechanism through which to select their desired components; in each case this is provided as a hierarchical list. Following component selection both the Kepler and Triana systems allow the user to arrange and connect their components within a blank "canvas" region of the user interface, creating connections by dragging links between appropriate ports of those components. In Taverna these connections are made through the use of an additional dialogue window listing the components currently present in the workflow composition; from here the user can select a component and through the use of a drop down menu indicate which other component they wish to connect it to. Figure 4-1 is an illustration of the Kepler UI as it looks during this composition process; other UIs are similar.

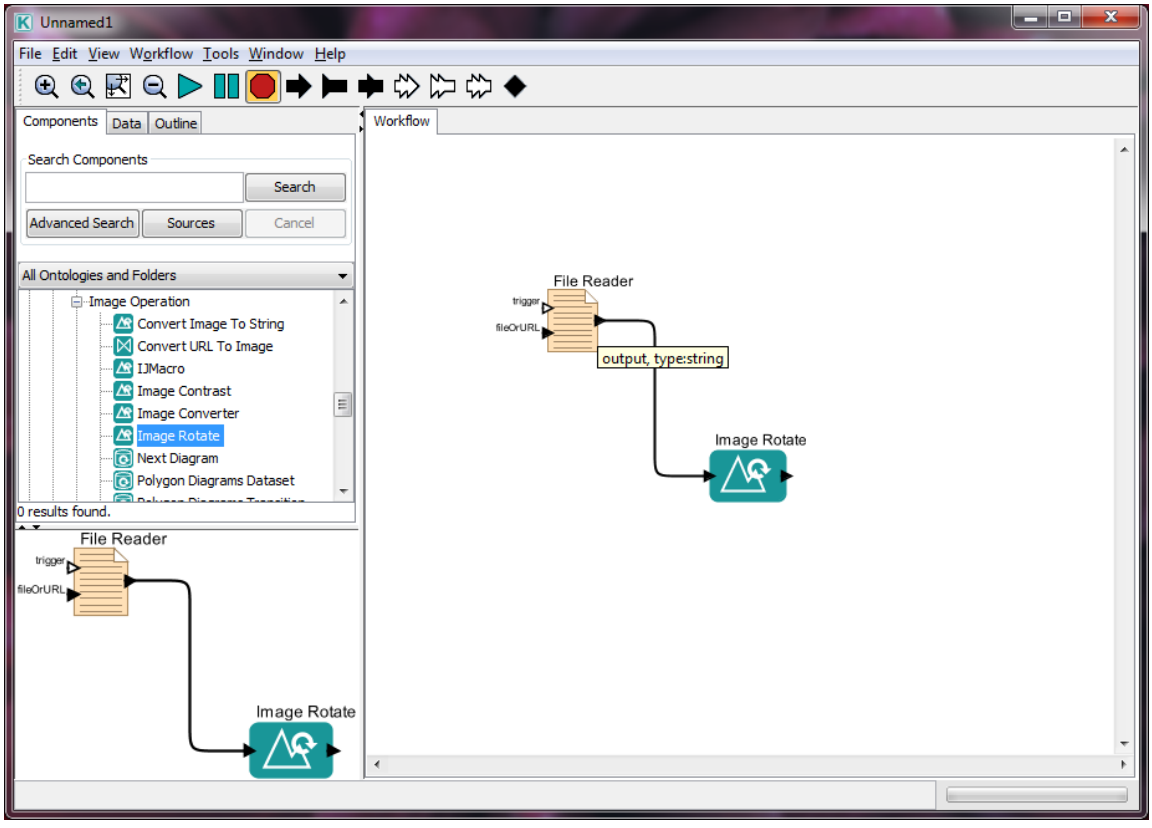


Figure 4-1 Screenshot of Kepler UI during composition

In order to assist the user in overcoming difficulties which may be encountered during composition each SWS provides a range of approaches designed to provide either information or assistance.

4.2.1 Component Port Types

In order to establish successful connections between components within each SWS the data types of those components connected ports must match. Each system makes use of a variety of types which are mapped to each component's input and output ports; for example the Kepler SWS has port types such as String, Integer etc. In order to help users establish whether components they wish to connect have matching port types, each SWS provides a means of inspecting the port types of those components which the user has inserted into their workflow composition. For example, in the Kepler UI the type of a port can be displayed by highlighting that port on the workflow canvas or by viewing the component documentation; in this way the user can identify whether the two components are syntactically compatible. However the usefulness of this information is limited as a result of the fact that although two components may have syntactically matching ports this does not mean that connecting them will provide any useful result. For example the Triana system provides components "DeSerialize" and "HistoryWriter" which have ports with the type "Object", but whilst their ports are of matching types connecting them would not provide a useful result. Similar problems exist in both Taverna and Kepler; for example in Kepler a large proportion of component ports use the "String" port type, meaning that from the system's point of view most components can be successfully connected even if their composition would be of little benefit and individual components may fail to process the string provided (e.g. because a molecular sequence string is expected).

Whilst identifying the port types of components is important in order to determine whether two components are syntactically compatible, it is not capable of identifying whether those components are semantically suited for connecting to one another. As seen with the previous example it is also possible that two components with compatible ports will fail to work in conjunction with one another as the specifics of the data communicated between them leads to compatibility issues. In addition before any

compatibility checking can be performed the user must first have identified the components they believe are candidates for connection.

4.2.2 Component Listing and Naming

The manner in which components are listed for selection within each SWS is designed to provide a degree of assistance toward this identification of desired components. In the Taverna SWS components are listed according to their "Provider". In this way, if a user is aware of the provider of the resource they require they will be able to quickly identify this from the list. Triana takes a similar approach; however here components are listed according to their domain of usage, such as "ImageProc" for components used in the field of Image Processing. As a result if the user knows the relative “domain” of task they wish to perform they can inspect this area of the component list to locate the components they need. Kepler lists available components in a manner which combines both of the techniques of Triana and Taverna, grouping the components under the categories Components, Projects, Disciplines, and Statistics.

Whilst this information may be able to assist the user in identifying components, it still requires the user to be aware of the domain, project or discipline to which their desired component belongs. In addition this relies on there being agreement between the users' and the system's views of which domain a certain component belongs to, and this can be made even more difficult for components which may be routinely used in multiple fields. If the user is unaware of the provider or domain of use of their required component, the conventions used to list components within each SWS will offer them no benefit.

In addition to the manner in which the components are organised within lists and menus, a further feature of existing SWSs that can potentially help the user is the component names themselves. Each SWS has adopted an approach whereby components are given descriptive names, such as the ImageReader component from Triana. However as noted by Berkely *et al.* [13] there are cases where this naming can introduce ambiguity over the function of a component, and there are instances where the tools and analyses used in separate fields will have the same name but perform different operations. Sirin *et al.* [34] also note that the naming of components may not always offer enough information to enable users to discern their purpose. As a result,

whilst naming components in relation to the task they perform and providing a listing of components that indicates the domain in which they are used is of benefit in identifying suitable components for a workflow composition, there are still limitations to this approach. As an example the Kepler SWS includes components with names FileReader, FileFetcher, and SimpleFileReader. If attempting to read an input file as part of a workflow composition a user may find it difficult to determine which of these performs the task they require. Furthermore, embedding implied semantics into a component's name does not assist in any reasoning tasks performed by a computer to assist a user in selecting suitable components.

4.2.3 Component Descriptions

Each of the existing SWSs provides further information about each component available, which the user can inspect. This information provides the user with a description of the component's usage. Such information is of benefit to a user who is unsure of which component they require, and could assist them in confirming whether their chosen component is going to perform the function they require. However the manner in which these descriptions are currently provided by the systems is of limited benefit. A major drawback to this approach is that the onus is placed on the user seeking out the information; they must first identify components which they suspect may be of relevance to their workflow composition and inspect the descriptions of those components to determine whether this is the case.

Given the large number of components provided by each SWS, and the limitations described previously in how these are presented to the user, this process would be very time consuming for the user. In addition the nature of the components provided by these SWSs, coming from various different providers and sources, means that the information provided in component descriptions is of varying detail. Some components such as GARPPreSampleLayers from the Kepler system provide great detail about their function, limitations and usage, where others such as JobManager or Parameter currently provide no documentation at all. As in the discussion in the previous section, a fundamental problem remains that the computer cannot help the user by inspecting these descriptions, as they are not semantically or ontologically based.

4.2.4 Assistance Provided by Existing Systems - Summary

These ad-hoc approaches that the user can take to overcome the gaps between what they know and what the system needs them to know in order to complete a workflow composition rely on exploiting the information and metadata that the SWS provides to describe the available components and the user's current knowledge of the system, its components and their ultimate goals. The information which is provided within current SWSs about a component, its usage and properties can potentially be of great use in helping a user identify which components are required to successfully achieve the goals of their composition, by making clear the purpose of a component, as well as enabling a user to connect components so that they execute correctly, by stating the properties and requirements of each component.

However, in its current state the information provided is insufficient to assist in all circumstances, and is only utilised passively and indirectly, with the user inspecting this information and drawing their own conclusions. No mechanism is provided to actively assist the user in locating and sequencing a set of components to achieve their requirements.

4.3 Metadata Assisted Composition

The approach outlined in the remainder of this chapter sets out to provide a standard set of metadata which is stored relating to each component available within a given SWS. The importance of high quality metadata to the facilitation of workflow composition and re-use was identified at an early stage as an area of focus during the research conducted for this thesis, and, as described in Chapter 2, it is something which has seen increasing recognition within the SWS community. Both in parallel with, and since the completion of, the work practical work for this thesis, a number of projects have undertaken to explore the benefits that high quality metadata can have for SWSs [78, 81, 82, 120].

Through providing an ontology which explicitly records a standard set of information relating to components and their relationships with one another, an approach to workflow composition assistance has been developed which uses this information as a means to support users with decision making during workflow composition; this

approach will be presented in the following subsections. By storing a standard set of metadata relating to components this new approach can avoid situations encountered in existing SWSs where the differing quality and consistency of descriptions can prevent the determination of compatibility between components or the suitability of a component to achieve a desired goal. Furthermore, standardizing this information and recording it within an ontology makes it possible for a system to begin automatically extracting useful knowledge about the components represented.

The following subsection will introduce the properties of components that are desirable for automatically deducing their suitability and compatibility for composition, and how these can be represented within an ontology to enable the extraction of knowledge required to assist users in workflow composition.

4.3.1 Component Metadata

There are a number of main tasks that a user must perform in order to successfully achieve their goals in a SWS. Primarily these are the identification of required components, and the suitable sequencing of these components to provide the desired output or results. Metadata regarding elements such as the components themselves, their relationships with one another, as well as their usage by a particular domain or user, can be utilised to assist in these tasks.

In the following sections we define a standard set of metadata properties to be stored regarding workflow components and introduce several approaches which are designed to utilise this metadata to assist the user to compose their workflows.

As explained in Chapter 3, these metadata elements are designed to enable the system to provide three varieties of workflow construction assistance to the user:

- Assistance in translating a high level concept of the tasks they wish to perform into a lower level that matches that of the components provided by the SWS
- Assistance in identifying the additional components required for their composition
- Assistance in identifying the manner in which the selected components should be connected and sequenced.

The following approaches and concepts are used in order to enable the metadata to support such assistance:

- **Abstract Components / Task Hierarchy** – The purpose of metadata of this sort is to relate components to the tasks which they perform, recording the tasks as a hierarchy with the specific component itself being the leaf node before working upward through progressively more generic tasks which the user can relate to their workflow composition goals.
- **Port Data Objects** – By relating each component's input and output ports to the specific data which it consumes and produces, rather than arbitrary types such as String or Integer as used in current SWSs, the user, and the system, can be provided greater information with which to determine whether components are compatible with one another, and also whether they are suitable for contributing to a workflow which will satisfy the user's requirements.
- **Composition Suggestions** – By inspecting the information stored in the component metadata the system can identify components which may be suitable for inclusion in the user's workflow composition, offering these to the user as a set of suggestions for changes to make to their ongoing composition.
- **User Interaction History** – By recording information about the decisions a user makes during composition, the system can utilise this information to tailor future suggestions to the user's requirements. The present approach limits the usage of this history to the specific user that has generated the history, although the impact of widening this to others is discussed in Chapter 10.

4.3.2 Metadata Structure

Information relating to a component, such as its name, domain of usage or provider, can be useful in assisting a user to identify the components required to complete the composition of their workflow. However, as we have seen in the previous sections, whilst this information can be of benefit, the current implementation of such metadata has two main drawbacks – there is a lack of depth and consistency in this information across the vast range of components provided by current SWSs, and the metadata provided is only able to assist the user if they inspect this information manually and draw their own conclusions regarding the best step to take.

By defining a standard set of metadata properties which are to be provided across all components it becomes possible to allow the system itself to inspect and reason with this information in order to provide the user with assistance during workflow composition. In addition, by extending this set of metadata to include properties not currently present in existing SWSs the accuracy and usefulness of the assistance the system is able to provide can be further enhanced.

The items listed below represent the specific metadata properties which need to be recorded in order to support the means of assistance introduced previously:

- Component Metadata
  - Name – *The component's name*
  - Task(s) Performed - *A description of the task the component performs*
  - Project(s) – *Names of projects for which the component was either created or in which it has been routinely used*
  - Provider – *The designer or developer of the component*
  - I/O Connection Ports – *The input and output capabilities of the component, including the following further details:*
    - Port Name
    - Port Type
    - Port Data Object
  - Connection History – *A list of the components which have previously been connected to this component by the user.*

Figure 4-2 shows a number of items within the ontology which was constructed, in order to illustrate its structure.

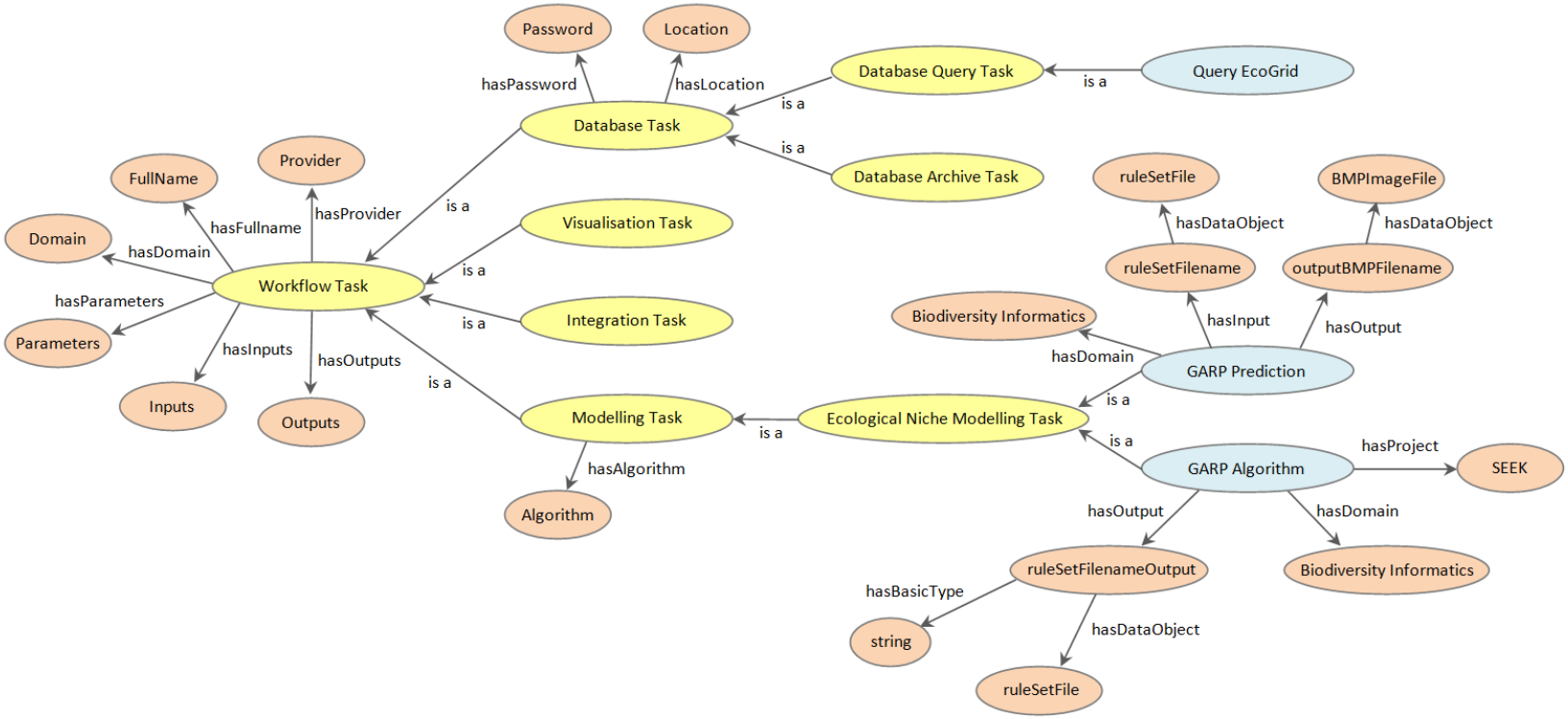


Figure 4-2 Illustration of a number of items from within the ontology

The following sections will discuss the role of each element of metadata in supporting computer-assisted workflow composition.

4.3.3 Component Name

The first element of metadata which is being recorded is the name of each component. As we have seen, the current SWSs attempt to provide descriptive naming of components in order to assist the user in identifying their required components. In order to maintain this benefit, and to ensure that users familiar with the components provided by the existing SWS are still able to locate these components, the name of each component as provided within the existing SWS is retained within the metadata framework proposed in this thesis.

By retaining the name of each component, users will still be able to perform a degree of deduction of those component's operations from their naming, which may provide some limited benefits; however as we have seen in Section 4.2 this information is often insufficient for this task, resulting in the need for additional metadata to be stored relating to the task or tasks which a component performs.

4.3.4 Component Tasks

During workflow composition one of the user's primary concerns is clearly to determine the components which perform the operations they require. To this end a valuable element of metadata to store is the task or tasks that each component performs. Whilst such information can sometimes be deduced in existing SWSs, through either the component's name or its location in the component listings, we have already emphasised (Section 4.2) that the component name will not necessarily give the user a clear indication of the operation that component performs; even worse, it may lead them to believe that it performs another operation altogether.

In addition many components in existing SWSs are listed in such a way as to give the user no immediate information about the operation they perform. For example, whilst some components in Kepler are listed relating to their function, such as the component “Image Rotate” which is listed under the headings “Data Operation->Image Operation”,

giving the user a clear picture of the operation it performs, other components are listed in a less helpful fashion. For example, under the "Data Output->Local Output" headings three components are listed: "FileWriter", "LineWriter" and "TextFileWriter"; the operation of each of these and distinction between them is unclear given their generic names and location within the heading hierarchy. By maintaining metadata relating to the task that every component performs it is possible to avoid the situation where the user is unable to work out the operations that a component performs, and therefore which components are potentially relevant to their current task.

In order to represent this information within our new framework it is necessary to collate the various elements of information relating to the tasks which a component performs that are made available within existing SWSs. As previously discussed, such task information is provided by various sources including the names of components, their textual descriptions and the hierarchical listing of components within the tree of those available. From these sources of information it is possible to derive the overall operation or task that a component performs and to represent this as an individual element of metadata associated with that component.

For example within the Triana SWS the list of available components places each component under a heading which represents the type of operation it performs. One such heading is “ImageProc”, which contains the component “Invert”. By inspecting this listing - the name of the component - as well as its description, we can deduce that the component “Invert” is an image processing component utilised for the task of inverting the colour of a given image. By completing a process of inspecting the component listing from each existing SWS it is possible to discern, with a reasonable level of clarity, the overall operation that each component performs. This can then be recorded as the “Task” element of metadata for the component.

Rather than relating each component to a free-text description of the task it performs, the proposed metadata framework represents the task performed by each component within a hierarchy. This enables components which perform similar operations to be grouped under the same sections of the hierarchy, and allows for the tasks to be described from both a high level, aiming to more closely match the abstract view that a user may have of the tasks they wish to perform, and also from progressively lower

levels which more accurately represent the specific task that a component performs. Figure 4-3 depicts part of this hierarchy, illustrating how Invert, along with several other Triana tasks are represented.

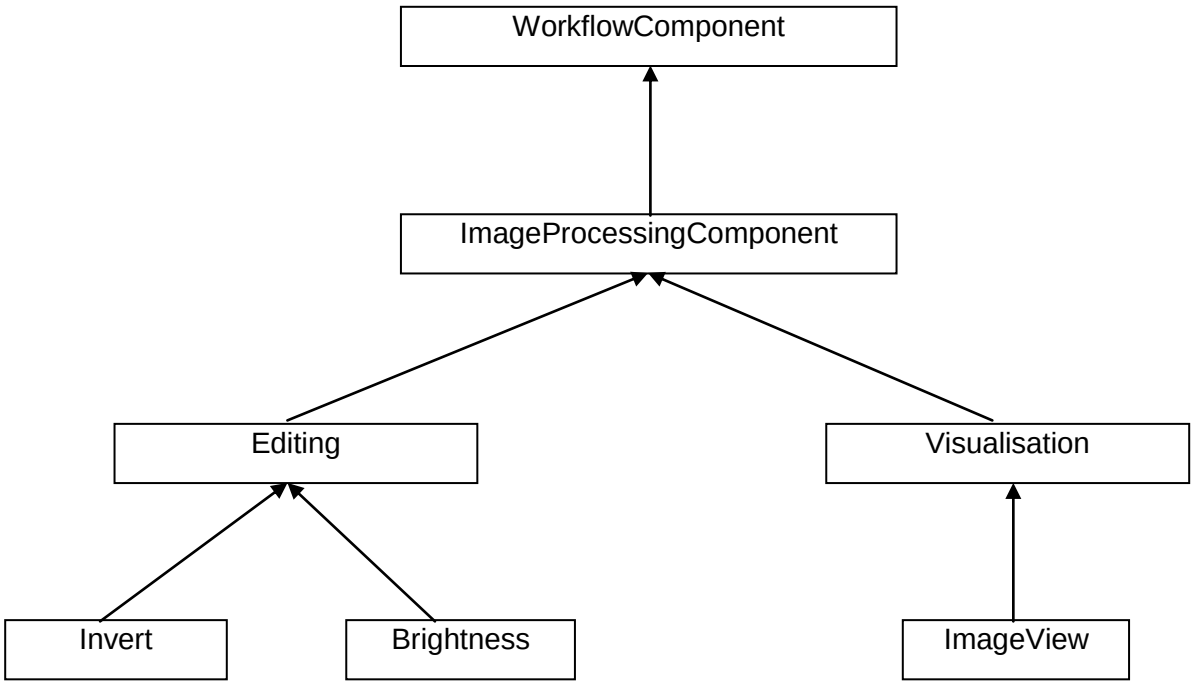


Figure 4-3 Image Processing components within the component task hierarchy

As discussed previously, users of workflow systems have to overcome a large knowledge barrier in order to successfully compose workflows which satisfy their requirements, and often this is due to their view of the workflow being defined at a higher level than that required by the SWS [7, 59, 77, 90]. By introducing levels of abstraction in this way, the framework can support a system whereby users can specify their workflow goals at varying levels of detail, dependant on the knowledge they have about what is required at each step. This representation of component tasks within a hierarchy enables the user to be able to investigate the path from the high level abstract concepts to the concrete workflow components which will achieve the tasks they require, and importantly by maintaining such a structured representation of this information the system itself is able to inspect this task metadata, so that given an abstract concept it can provide suggestions for the user as to which concrete implementation of this concept they should use.

4.3.5 Component Project, Domain and Provider

Currently systems such as Kepler and Taverna list their available components in relation to the provider of the component or to the domain or project in which that component has mostly been used. If the user is aware of a project having goals which match their own, or a resource provider which they have made use of in the past, then this information can be of benefit to the user in identifying the components required for their workflow composition.

From inspecting the information currently provided in the SWS the metadata framework can be populated with knowledge relating to the Project for which a component has been developed, the Domain in which it has been utilised, and the Provider who has developed that component. By recording each of these metadata elements for all components rather than only including the provider information as is currently available in Taverna, or only including the project information for certain components but not others, the user can be presented with a uniform view of the components which are available to them and therefore make more informed decisions about which components to include in their workflow composition. Again, providing this information in a highly structured manner enables the possibility of the system being able to inspect and understand this information itself and as a result be able to identify components which may be suitable for a user's needs.

4.3.6 Component Connection Ports

As discussed previously the input and output ports associated with components in current SWSs are restricted by the type of information they either produce or consume. For example an image viewer component might only accept an input of a given set of image types. As a user progresses with their workflow composition this port type information becomes increasingly useful in identifying which components to connect out of those already included in the composition, as well as helping to identify further components which could be included. For example if a user is constructing a workflow to perform a GPS mapping operation and has already identified a component which can translate GPS co-ordinates into a visualisation then knowledge about which components provide such co-ordinates as their output will be of benefit to them in deciding the next step to take.

However, as discussed previously, existing systems (particularly Kepler) provide only a limited number of port types, meaning that whilst this port type information is of benefit in identifying whether components are syntactically compatible it only has limited benefit in identifying whether the connection of those components will provide a useful and semantically valid outcome. Taking the above GPS example again, if the visualisation tool had a port type of “integer”, knowledge of this would be of little benefit to the user in identifying which component to connect as there are a large number of available components which produce an output of type "integer" and many of these will produce data in a form that is meaningless as input for the GPS visualisation tool. The basic knowledge that a component produces or consumes output of a simple type such as "string" or "integer" is not sufficient in many cases to identify whether the particular output from one component will be sensible as input for another, despite their matching port types.

Such inadequacies with the information available in existing SWSs effectively limit a user's ability to identify both the compatibility which a component may have with other available components, and also the range of situations in which the use of that component may be appropriate. To address this problem it is desirable to store knowledge within the metadata framework about component ports which is targeted at not just assessing whether connections between components are syntactically possible, but also whether such a connection is desirable in terms of producing useful output and helping a user achieve their overall goals.

In order to facilitate this more detailed evaluation of component compatibility a further element of metadata is defined relating to each component port, the Data Object. Whereas the basic type information related to each port within existing SWSs is typically a loose description such as “String” or “Integer”, the aim here is to relate each component port to a more specific definition of the information which it either produces or consumes. For example, if a component has a current output port of type “String”, is there any further information known about the nature of the string which this component produces which could assist in identifying whether it is a suitable input for another component? From a basic level we could assert that this string is a URL, or a file location, or some other form of unique identifier. Beyond this it may also be possible to

infer more specific information, if this is a file location what is the type or extension of the file, is this identifier a scientific name, a geographical location etc.

As with the metadata relating to a component's task, this port data object metadata is maintained within the framework as a hierarchy (Figure 4-4). As we may be able to infer different levels of detail about the information which is produced or consumed by different components, it is desirable for the metadata framework to allow flexibility in the representation of this information.

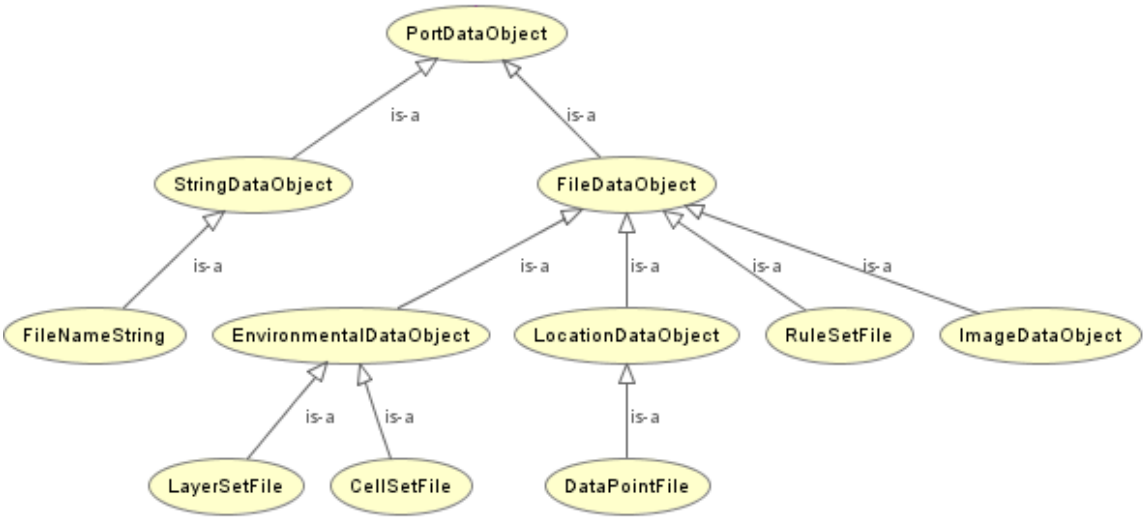


Figure 4-4 A subset of the PortDataObject hierarchy

For example, whilst it may be possible to characterise the output of one component to a very specific level of detail, such as an image processing component which produces an image file of type GIF with dimensions 256x256, another component may only be specified as producing an image which could of varying file types and dimensions. Similarly the level of detail which can be inferred about the input expected by different components will feature similar variation. Recording this port data object information as a hierarchy means that a user can identify the component which produces a 256x256 GIF as being compatible with both a component which expects only this specific type of image as an input and also a component which will accept any form of image as input.

By recording greater detail relating to the information which components produce as output and expect as input the user is given more chance of identifying those

components which are necessary to complete their workflow composition and can have more confidence that the sequencing of those components will produce correct results. Again the recording of such information in a structured manner also makes it possible for such type matching and component identification to be automated by the system.

4.3.7 Component Connection History

The elements of metadata discussed in this chapter are designed to assist the user in identifying the components which are necessary for achieving their goals, as well as the manner in which those chosen components should be connected and sequenced. An additional element of metadata of benefit in this task is knowledge of the components which a specific user has made use of in the past, and the manner in which they have previously connected those components.

The benefit of taking into account the actions that a user has performed previously whilst using a SWS is that, as a part of an assisted approach to workflow composition, such knowledge can assist in narrowing the field of available options to those which are of relevance to the user's context. Given the large, and increasing, number of components which are available in current SWSs, the process of locating those components which are relevant and useful to an individual user is becoming increasingly difficult. In addition, as much of the literature regarding the development of SWSs shows, users are commonly involved in creating workflow compositions for a specific project or domain, meaning it is not uncommon for restricted ranges of components to be required [4,6,8,9]. It is these users who are targeted by this approach.

If the system were made aware of the component connections which a user has made in the past then it would be possible to identify trends and common relationships between components and highlight those frequently used components to the user. In addition, combined with the other elements of metadata such as port types and data objects, this could help the user in identifying other components to connect with those they have used previously to achieve new goals.

An important consideration in this instance is the extent to which such a connection history can be of benefit in terms of end users. A single user, working in a single

domain or within a single project, who makes use of such a system recording their component interactions is going to develop a connection history that may be of significant benefit to their own future work within that domain/project. However, the benefit that this record of their connection history may have on other users' compositions is more difficult to predict. It may be that inter-domain connection histories will be of benefit to individuals working in those, or closely related, domains, and similarly inter-project histories may be of benefit to others working on the same project. Such information could be of particular benefit to new users who have yet to develop a history of interaction with the system, in these instances the decisions which have been made by other users within their field of work may help guide them through composition tasks where they may otherwise encounter difficulty. In addition as the connection history information is only used as a means to generate suggestions about steps to take the user is free to explore other options external to those which they or others have chosen in the past, an option which may be of benefit as users' become more experienced and less reliant on suggestions to progress.

A further complication would be the specificity of a component to a particular domain or project. If a component is used frequently across many domains, and is utilised for similar purposes across those domains, then connection histories relating to that component may be of benefit to users from other domains, whereas the connection history of a component specific to a single domain, or a component that is used in a different context within different domains, is likely to be of limited benefit beyond the domain in which the history was recorded.

In order to take advantage of trends and common relationships which may develop regarding component connections there must be a mechanism to record relevant information during workflow composition. The following elements of information are recorded within the metadata framework for the purpose of maintaining component connection histories:

- **Components** – The two components involved in a connection
- **Ports** – The individual input and output ports which those components are connected through
- **User** – The user who has created this connection

- **Connection Count** – The number of times that this particular connection has been made

Recording the components and ports involved in a connection is essential, as this is the information required to monitor whether the connection occurs frequently. Maintaining just a history of the components connected, without the information regarding the port they were connected through, would have limited value as users developing a workflow will want to inspect the way in which those components have previously been connected. In addition it is beneficial to identify the user who has made this connection, both so that the system can use this knowledge of their connection history to assist them personally as well as the possibility of others who work with, or in the same domain as, this user. Finally it is essential to record the number of times that this particular connection has been made as this information can then be used to determine if a particular connection is occurring more frequently than others.

An additional possibility, but one which has not currently been implemented, is to extend the information stored about the “user” that has made a connection to incorporate further metadata such as the domain(s) in which that user works, and any projects within which they utilise workflow compositions. This information would again be of use in determining other potential benefactors of an individual user’s connections history. In order to assist in determining whether connection histories from one domain or project may be of use within another, the “domain” and “project” metadata associated with a user could be maintained within a network structure. This would represent the relative “closeness” of the domains, with those which are deemed similar, due to involving related work or use of a SWS, able to take advantage of each others' connection histories.

4.4 Representing the Metadata Ontology

In order to be able to use the information regarding components described in the previous section during the workflow composition process it is necessary to define a standard model for storing and interacting with this information. As discussed previously, a drawback with the descriptions and information provided regarding components within existing SWSs is the lack of consistency present within those details provided. By providing a standard set of information regarding all of the available

components it is possible for users to gain advantage from this information during composition. The representation of this metadata is structured via four main concepts:

- "WorkflowComponent" - A base definition of the core attributes required by all components - this acts as a "superclass" for all components and defines attributes discussed previously - the components name, provider, ports etc. Each of these attributes is a unique "class" of its own, describing the attributes required of it (e.g. the type of a port)
- The "Task Hierarchy" - A series of sub classes extending WorkflowComponent. Classes which extend this can be either abstract components representing generic tasks, or concrete components which implement those tasks. Each of these sub classes can extend WorkflowComponent to define additional attributes specific to that task.
- The "PortDataObject Hierarchy" - Each component port has a PortDataObject attribute which associates the port with the specific data that it produces or consumes. Similar to the Task Hierarchy these PortDataObjects are represented as a series of subclasses, identifying relationships between the different types of data which each component processes.
- The "Connection History Repository" - A data structure which stores the necessary information to record the history of components which a user has connected during their use of the system.

Developing a well defined format in which to represent these details enables such information to be machine-understandable, presenting the opportunity for a SWS to provide assistance to users during composition based on this knowledge.

The information for the first three of these concepts (the "WorkflowComponent" representation, the "Task Hierarchy", and the "PortDataObject Hierarchy") are all maintained within an OWL ontology, representing the relationships between components and their metadata. Figure 4-5 provides an overview of these concepts and how they are related within the ontology. A proprietary data structure is used for maintaining the history of connections formed between individual components.

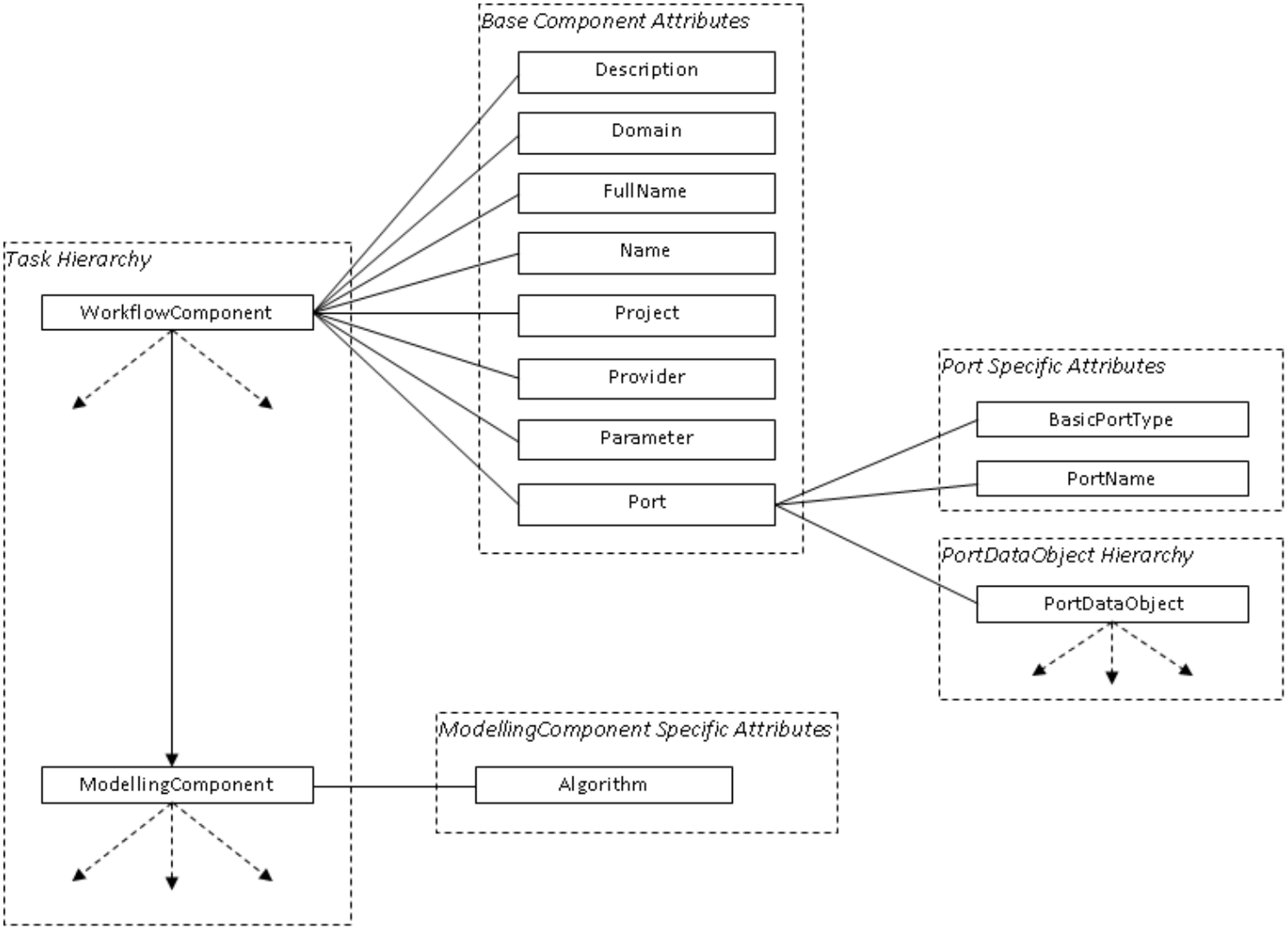


Figure 4-5 Main Concepts within the Metadata Ontology

OWL is a W3C endorsed language for knowledge representation [49]. The OWL language was chosen as it is a well developed language which provides capabilities which are suitable for modelling the relationships between workflow components and their related metadata. Specifically it allows for the easy representation of hierarchies of elements; in this way the hierarchy of component tasks and port data objects can be easily represented. In addition the tools to enable the interaction between OWL and the JAVA programming language are sufficiently mature that the required inspection and reasoning with the metadata would be possible from within the proposed scientific workflow system extension. As discussed in Chapter 2 a number of variants of OWL have been developed such as OWL-S [25], OWL-WS [26] and WSMO/WSML [114] to enable the semantic description of web services. However for this work standard OWL has been used as it was identified as sufficient to represent the knowledge and relationship information required for the component metadata ontology.

The components and related metadata from each SWS will be stored in separate ontologies, as the approach to computer assisted composition explored within this thesis is designed to work with only one SWS at a time. Given the recent developments toward support interoperability between SWSs through projects such as SHIWA, an area for future investigation would be to explore how such an approach to assisted composition could be applied when working across several SWSs simultaneously.

An OWL ontology represents concepts within a domain as a set of “classes” with relationships defined between those classes, and “axioms” are used to define what constitutes an element of each class. “Individuals” are the actual items of data that belong to a class.

Within this structure provided by OWL for representing knowledge it is possible to define classes and relationships to model the workflow components and their associated metadata that are provided for composition within existing SWSs. The following sections discuss how the elements of metadata introduced in the previous section are represented within the OWL ontology.

4.4.1 Component Representation

There is a base set of attributes which all components have in common and must be defined for them to be deemed valid, a generic `WorkflowComponent` class is defined within the ontology to represent these base attributes. This class has several restrictions associated with it. These define the characteristics that an individual must possess in order to be classified as a `WorkflowComponent`. The restrictions associated with the `WorkflowComponent` class are shown in Table 4-1.

Class	Restriction
WorkflowComponent	hasComponentDescription some ComponentDescription
	hasComponentDomain min 1 ComponentDomain
	hasComponentFullName some ComponentFullName
	hasComponentName some ComponentName
	hasComponentProject some ComponentProject
	hasComponentProvider some ComponentProvider
	hasParameter some Parameter
	hasPort some Port

Table 4-1 Restrictions defined for the WorkflowComponent class

These are the basic properties that all `WorkflowComponents` must possess. A component must provide a `ComponentDescription`, giving an overview of the function and purpose of that component, at least one `ComponentDomain`, describing the domain(s) in which that component is utilised, a `ComponentProject`, this describes the project for which a component was developed, a `ComponentProvider`, listing the individual or organization that developed the component, a number of `Parameters`, these are internal settings provided by components to control their operation, and a number of `Ports`, the interfaces through which the component can communicate with other components. Properties such as `Parameter` are not required for each component, as some simple components may not require any specific configuration in order to function, a potential improvement to this approach would be further classify each parameter of a component into those which are optional and mandatory to further inform the system as to the configuration requirements of each component.

Two representations of the component's name are also defined, `ComponentName` and `ComponentFullName`; these are used to represent the name of the component as it is presented to the user, and the fully qualified class name of the component as it is used within the SWS. The latter is required to enable the API to easily instantiate new instances of a component as required.

These base attributes form the starting point for defining a rich set of metadata which can be used to identify each component's suitability either for inclusion within a workflow composition or for connecting with another component. This representation associates each component with a Domain, Project, and Provider, as well as with the relevant Ports that define its interaction with other components. As each restriction within OWL is itself an OWL class it is possible to then define the properties that each of these classes must possess, defining what is required of a `Port`, `ComponentDomain`, etc. In this way it is possible to quickly develop very specific definitions for the properties that members of a class must possess, and so provide a structured representation of components and their properties that can be interpreted and leveraged by an external system.

The definition of a `WorkflowComponent` is the first step in associating individual components with the appropriate metadata that could be of benefit to users in composing workflows. However, as discussed previously, relating components with the tasks that they perform may enable users to more readily identify those components which can assist in achieving their goals and makes it possible for a workflow extension to provide more useful assistance to those users.

Additionally as OWL supports inheritance it is possible to define sub-classes of the `WorkflowComponent` class, where each new class will inherit the properties of `WorkflowComponent` as well as being able to obtain new properties specific to the class. This process of creating sub-classes can be repeated through several iterations, enabling the construction of the component task hierarchy as introduced in Section 4.3.4. In this way, rather than relating a component to the task it performs through the use of a further restriction on `WorkflowComponent`, the task that a component performs can be inferred from its location in the class hierarchy.

As discussed previously the advantage of this approach is that the user and the system can inspect this hierarchy from a number of levels. Hence it is possible to carry out the initial identification of the high level task that is required, before inspecting the areas of the hierarchy that inherit from this task in order to locate a specific component which will perform the required task.

4.4.2 Component Port Representation

The `WorkflowComponent` class defined within the ontology includes the restriction "`hasPorts some Port`". This relates each component to a number of input or output ports which it must possess. The `Port` class itself defines the requirements of what a Port must be, including the various elements of metadata previously identified as being beneficial – Name, Basic Type, Port Type (input or output), and Data Object. This section describes how the Port class and these elements of knowledge are represented within the ontology. The restrictions defined for the `Port` class within the ontology are shown in Table 4-2.

Class	Restriction
Port	hasPortName some PortName
	hasPortBasicType some PortBasicType
	hasPortDataObject some PortDataObject

Table 4-2 Restrictions of the Port class

Beyond this, the distinction between input and output ports is achieved through providing two subclasses of `Port` – `InputPort` and `OutputPort`. As these inherit the restrictions of their superclass `Port` it is not necessary to define additional restrictions.

These Port restrictions define what each Individual must possess in order to be classified as a port. `PortName` is simply the name of the port as declared within the SWS in which it is utilised. `PortBasicType` refers to the simple "type" of information that is accepted through the port, such as a string or integer value. `PortDataObject` refers specifically to the data itself that is involved in an interaction between ports, and is defined in more detail than `PortBasicType`.

As discussed previously the purpose of the `PortDataObject` class is to define more accurately the nature of the data that is transferred between components when they have been connected. This represents the known properties of the data that an output port produces and those that an input port expects to receive. Again using the capability of OWL to represent hierarchies of classes, we can define many subclasses of `PortDataObject`, each representing progressively more restricted types of data. Again as discussed in the previous section this would start from high level abstract types such as a "File" class which could be associated with any port that sends or receives a file, before being specialised into more restricted classes such as "Image File", "JPEG File" and so on.

In a similar manner to the benefits gained from relating each component to an element from the task hierarchy, relating ports to specific elements from this data object hierarchy allows for a greater level of understanding of what data is being transferred between components. With such information being defined for both the inputs and outputs of available components it is then possible for those component connections most likely to have a high degree of compatibility to be readily identified.

4.4.3 Connection History Representation

Connections created during a workflow composition are stored within a basic data structure which is maintained by the user interface. The history of connections is recorded within an expandable data structure, this enables an arbitrary number of elements to be recorded and is able to be extended or reduced in size dependant on the needs of the system.

Each connection itself contains the relevant items of data discussed previously – components, ports, and connection count. This information is stored within each individual connection as basic strings and an integer for the connection count. The user involved in the connection is recorded as a part of the overall Connections structure, representing each user's own connection history within a single structure. The overall structure of the Connections class is represented by Figure 4-6.

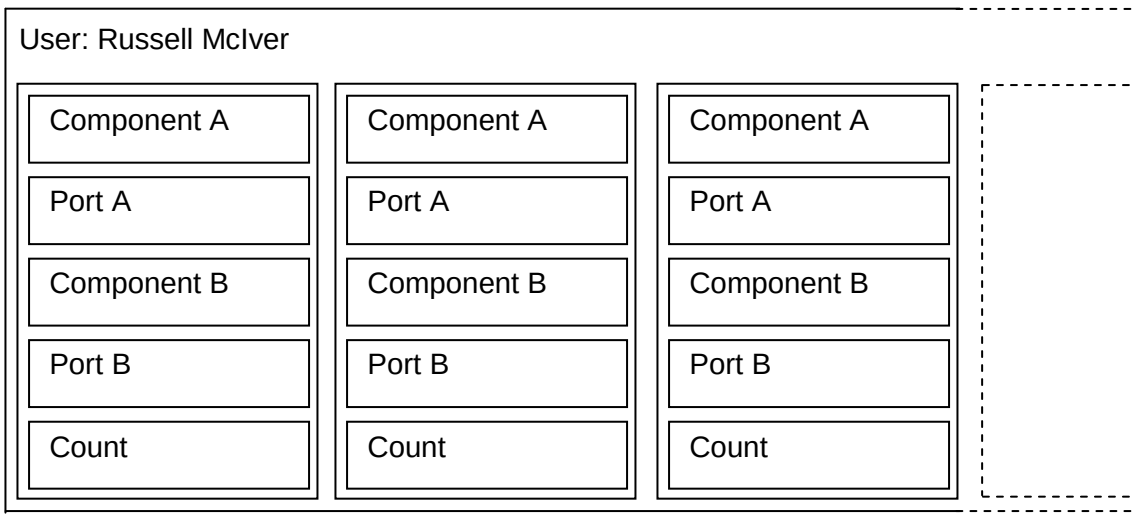


Figure 4-6 Connection history data structure

Storing the connection history within such a structure allows for the relevant information regarding the connections a user has made to be recorded, provides the necessary freedom to expand as the user continues to compose workflows and connect components, as well as allowing for suitable inspection of the information regarding connections made to support assistance with those future workflow compositions.

As the user proceeds to create workflow compositions the connection history data structure will begin to be populated based upon their actions. Initially there is a check to see if the user has previously made any connections; if not then a new Connection is created, populated with the relevant metadata and added as the first item of the user's new connection history:

```
If Connections List Empty
{
   Create new Connection instance
   Populate Metadata(Component 1, Port 1, Component 2, Port 2)
   Set Connection Count to 1
   Add new Connection to Connections list
}
```

If, however, the user has already been using the system then their connections list will already contain a number of entries, in this case the system must inspect its current contents to discover whether this new connection has been made previously and

therefore to increments its connection count, or whether this is a new connection that must be appended to the connection history:

```
For each entry in the Connections list
{
  Compare entry against connection user has just made
  If Entry matches new connection
  {
    Increment connection count
  }
  Else
  {
    Create new Connection instance
    Populate Metadata(Component 1, Port 1, Component 2, Port 2)
    Set Connection Count to 1
    Add new Connection to Connections list
  }
}
```

This representation of the history of connections made by a user is designed to enable the identification of trends in the connections made between components, and to assist in directing users toward those connections which may be of most benefit in future compositions.

4.5 Building the Ontology

The process of populating the component metadata ontology involves extracting information from a variety of existing sources. As discussed previously, information relating to the role and properties of components is maintained within a number of locations within existing SWSs, primarily within component descriptions and naming schemes.

In order to translate the useful information currently available from these sources into structured knowledge within the ontology a number of processes are involved:

- Developing the initial ontology structure to represent concepts such as Components and Ports
- Inspecting documentation of each SWSs available components for required metadata
- Introducing Individuals into the ontology that represent these components, and
- Extending the existing classes and restrictions to incorporate any further details required by these new Individuals.

Interaction with the OWL ontology during this work has been achieved through the Protégé ontology editing software [50]. Protégé provides facilities to model ontologies, allowing users to create and edit the various classes and restrictions required for their ontology, and also provides facilities for visualizing and reasoning with the knowledge represented within an OWL ontology. Initially the classes and restrictions to represent basic workflow concepts such as Components and Ports were created based on the definitions of WorkflowComponent and Port described earlier.

With these classes defined the next task is to populate the ontology with the individuals that represent actual workflow components. To illustrate this process a Kepler workflow scenario from the bioinformatics domain, Ecological Niche Modelling, is utilised. Figure 4-7 provides a graphical representation of this scenario. This workflow involves the use of the following workflow components:

- StringConstant
- GARPPresampleLayers
- GARPAlgorithm
- GARPPrediction
- ImageJ

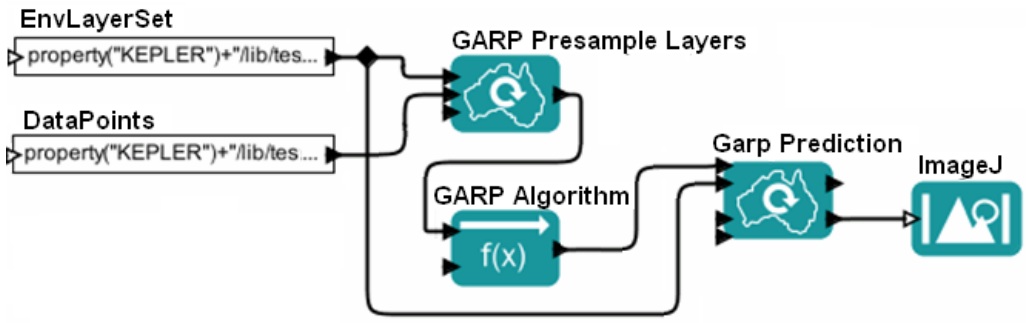


Figure 4-7 Ecological Niche Modelling Scenario in the Kepler SWS

Each component added to the ontology will be represented by a single new WorkflowComponent individual to represent the component itself, as well as a number of new Port individuals to represent each of that components ports. To satisfy the

restrictions defined for these ontology classes (as outlined in Table 4-1 and Table 4-2) we must inspect the documentation provided by Kepler regarding each component to locate the required information.

The first restriction for WorkflowComponent, ComponentDescription, is relatively straightforward to satisfy. Each component within Kepler provides a textual description of its purpose, for example the StringConstant component provides the following description:

> *The StringConstant actor outputs a string specified via the actor's value parameter.*
>
> *Specifying strings with the StringConstant actor is convenient, as the actor does not require that strings be surrounded by quotes. The actor is often used to specify file paths, which can be selected using the Browse button available in the actor's parameters.*
>
> *Specified string values can include references to parameters within scope (i.e., parameters defined at the same level of the hierarchy or higher).*

The remaining elements of metadata about each WorkflowComponent must be populated by attempting to extract the relevant details from such component descriptions as this. The difficulty in satisfying restrictions such as ComponentDomain, ComponentProject and ComponentProvider depends on the level of detail which is provided by the existing documentation. For a component such as StringConstant it can be assumed that this would operate across virtually any domain and so the ComponentDomain restriction can be satisfied by simply stating that the component is applicable to all domains. Additionally, as a generic component provided directly by the SWS itself, the Project and Provider restrictions would both be populated as “Kepler”.

In addition the description for the component informs us that StringConstant includes a parameter “value” used to specify the string which this component will pass on through its output port. This information can be used to populate the Parameter restriction of our WorkflowComponent.

However for other components it can be more difficult to identify the metadata required to populate the ontology. The following is the description provided for the GARPAlgorithm component:

> *The GARPAlgorithm actor reads a set of spatial locations and associated environmental data, and uses a genetic algorithm to create a "rule set" that can be used to make predictions about the presence or absence of a species at various locations.*
>
> *The input data is passed to the actor by the GARPPresampleLayers actor, which generates the environmental data (or "layers") in the appropriate format. Output is usually passed to the GARPPrediction actor, which makes the environmental predictions based on the generated rule set.*
>
> *The actor requires libgarp.so (on linux systems) or garp.dll and libexpat.dll (on Windows systems). Currently, the actor does not work on MacOSX systems.*
>
> *GARP (Genetic Algorithm for Rule Set Production) is a genetic algorithm that creates an ecological niche model representing the environmental conditions where a species would be able to maintain populations. For more information about GARP, see http://www.lifemapper.org/desktopgarp/.*

From this description we can begin to deduce some knowledge regarding the domains in which this component would be utilised, primarily that the GARPAlgorithm component would be used in the Ecology domain. For components such as this which either have a very targeted domain in which they are used or for which there is only limited information provided in their component descriptions, the process of assigning values to metadata restrictions such as the component's domain, project and provider can prove problematic. In these instances further sources of information must be sought, such as following any links to web pages for the developer of the component, contacting the developer directly, or inspecting example workflows where the components have been utilised in order to deduce suitable information with which to populate the ontology.

In order to mitigate the impact of this difficulty a possible improvement would be to allow users of the metadata ontology to propose their own alterations to the metadata held about components, or to extend it where it is presently lacking. This idea is explored more in Chapter 10.

In addition to inspecting the component descriptions as provided in the existing SWS, an alternative source of metadata relating to available components is the manner in which they are listed in within the SWS. By inspecting the component lists of Kepler, Triana and Taverna it is possible to infer the task performed, provider and domain or project of use for many components. This information can then be inserted into the ontology.

In order to populate the component port metadata (PortType and PortDataObject) one source of information is again the component descriptions provided by the SWS. For example some components within the Kepler system give details of their port names and types within their descriptions. As discussed previously the UI of the SWS also provides useful information relating to the components ports. In cases where the component description does not provide the required metadata relating to a component's ports this can be discovered once the component has been included in a workflow composition. For example, in the Triana and Kepler SWSs "hovering" over the component once it has been inserted into a workflow will display the name and basic type of that component's ports.

Populating metadata relating to the PortDataObject restriction is again a challenge, requiring inspection of both the component's port type as described previously as well as other sources of information such as the component's description. For example from inspecting the previous component description we can infer that the GARPPresampleLayers component produces environmental data from its output port, this can then be captured in the ontology. The hierarchical nature of the Data Object metadata means that in instances where the description of a component is unable to provide further information to capture in the PortDataObject restriction this can be left with a more generic data type, with the drawback that this would impede the quality of assistance which could therefore be provided by inspecting the metadata ontology. In addition "dry running" components and inspecting the type or format of data they produce is another possibility for discovering more useful information to be stored in the PortDataObject restriction, providing satisfactory input can be supplied to the component in order for it to run and that the characteristics of the output are suitable to be captured.

Again this very specific information relating to the type of information produced or consumed by components would benefit from the ability of users to be able to modify the ontology, thus improving the accuracy of the metadata which is being stored.

4.6 Summary

This chapter has discussed the drawbacks that exist with the manner in which component metadata is maintained and made available to the user within the existing SWSs and outlined how this is hindering users from overcoming the challenges which are presented to them when attempting to compose workflows. A new framework for workflow component metadata has been described, with the goal to provide a common set of metadata elements across all workflow components.

The manner in which this metadata is captured and represented within the ontology has been discussed, focusing on ensuring that such knowledge could be leveraged in order to enable an automated system to use this information to provide guided assistance to the user during workflow composition. In the following chapter the mechanisms which are utilised in order to provide this assistance are outlined.

5 Computer-Assisted Workflow Composition

5.1 Overview

This chapter presents a novel approach to workflow composition assistance which makes use of the component metadata discussed previously in order to generate suggestions which can be selected by the user during workflow composition. The metadata framework discussed in the previous chapter has been designed to enable the information which a user would previously have manually inferred from the SWS to be machine-readable in order to support the identification and suggestion of steps which a user may wish to take in order to complete their composition.

The system is able to provide the user with suggestions for components to add into their composition, connections which can be made between those components, and refinements which can be made to any "abstract" components which are present within the composition. In order to generate and suitably rank these suggestions a number of mechanisms have been utilised:

- Inspecting the content of the component metadata ontology
- Inspecting the current state of a workflow composition
- Generating suggestions for components to add, connect, or specialise within the workflow composition by comparing component metadata properties to identify *compatibilty*
- Evaluating the *desirability* of these suggestions in order to impose a ranking.

Details of each of these mechanisms and the suggestions which they support are provided within the following sections.

5.2 Composition Assistance

The approach to scientific workflow composition assistance described in this chapter operates on the basis of context-relevant suggestions. The system monitors the current state of a user's workflow during composition and uses this information as well as its knowledge of the user's history of interaction with the system, the domain in which the user is operating, and components stored within the ontology in order to provide suggestions of additions or alterations the user could make to their workflow in order to achieve their desired outcomes.

Within this approach the user is presented with a demonstration user interface much like that of an existing SWS[1]; they can select components to insert into their workflow from a list and are provided with a space in which to organise and connect these components. However, in our approach there are important differences and extensions in order to support the user when composing a workflow. The key differences present in the prototype interface are as follows;

- Users can insert into their workflow either abstract components from the "Task Hierarchy" described in Chapter 4 or concrete ones as provided by existing SWSs
- A tabbed list of suggestions is provided on screen allowing the user to view suggestions for specialising abstract components as well as components to add or connect within the workflow.

The details of the user interface provided to demonstrate this suggestion based approach to composition are covered in Chapter 7.

In providing users with informed suggestions for steps they can take to progress their workflow, the aim is to reduce the level of implementation knowledge required of users in order to create workflows which perform their required tasks, allowing them to focus on the purpose of their workflow rather than on how this is achieved within the SWS. In addition this approach seeks to remove the problem of users becoming “stuck” within the process of creating a workflow, where previously they may not have possessed the knowledge required to identify how to proceed; the suggestions can assist in overcoming this obstacle.

The suggestions which can be provided by this approach are grouped into three categories:

- Suggestions on how to specialise “abstract components” with corresponding concrete ones

[1] The interface was designed to be similar so as to take advantage of familiarity and limit the amount of relearning required from the user

- Suggestions for new components to insert into the workflow, and
- Suggestions for connections to create between components currently in the workflow

This chapter will discuss how each of these types of suggestions is generated within the system, including how the information within the ontology is inspected and the reasoning utilised to turn this information into relevant suggestions.

5.3 Mechanisms for using the ontology

In order to begin the process of providing suggestions to the user the system must be able to retrieve the information that is stored within the metadata ontology. As described in Chapter 4 the various elements of metadata relating to components are maintained within an OWL ontology. This provides a standard model for the representation of knowledge about components and similarly a standard means for interacting with that knowledge.

The existing workflow systems supported by the workflow extension API presented here are all implemented using the Java programming language. Accordingly, it is convenient for both the API and the workflow extension itself also to utilise Java for their implementation. Inspection of OWL ontologies from within the Java language is achieved using the JENA framework. JENA [51] is a framework for developing semantic web applications within the Java programming language and it provides a means for Java applications to interact with both OWL and Resource Description Framework (RDF) representations of information as well as providing support for the SPARQL [52] language used to query these representations.

Using JENA API OWL ontologies can be represented within Java as OntModel objects; the Java application can interact with this OntModel object in order to perform basic operations such as the retrieval of data regarding classes and individuals present within the ontology. For example, it is possible to retrieve all classes which are present within an ontology using the `listHierarchyRootClasses()` function provided by JENA.

In order to achieve more complex, selective interactions with the ontology, JENA provides support for the SPARQL query language. SPARQL queries typically consist of

one or more triple patterns, logical operators, etc., and enable the complex querying of knowledge within an OWL ontology, again accessed using the OntModel class. The specific SPARQL queries utilised to extract knowledge from the ontology will be discussed in the coming sections.

Once information has been retrieved from the ontology it is then possible to analyse the knowledge available about workflow components in order to generate suggestions to provide to the user. This analysis of component and user metadata aims to help automate the processes which users perform in order to identify steps to take during traditional manual composition; inspecting component descriptions to determine whether their purpose matches a user's goal, and checking descriptions and port types to identify whether components are compatible with one another. Furthermore this analysis makes use of information such as the domain of a user and of components, the past interactions between users and the available components, and further detailed information regarding data transferred between components, all of which is also maintained within the ontology.

5.3.1 Retrieving Component Metadata

The retrieval of information from within the ontology is performed using the facilities provided by the JENA framework. The goal of this information retrieval is to acquire the elements of component metadata that are relevant for discovering whether those components are suitable for inclusion within a user's workflow composition. This testing of components' suitability for inclusion addresses two criteria: a component's *compatibility* for inclusion within the current workflow, and a component's *desirability* for inclusion within the current workflow, requiring the use of the following elements of metadata stored within the ontology:

- Component Metadata:
  - Name
  - Task
  - Provider
  - Domain
  - Project
- Component Port Metadata:

- Basic Port Type
- Port Data Object Properties
- Past connections

Once retrieved, this component metadata can then be analysed and compared with other components' metadata in order to determine an individual component's suitability for inclusion within the workflow. The following sections discuss the manner in which this information is retrieved from within the component ontology.

5.3.2 Retrieving Component Task Metadata

As a user's goals during workflow composition are to correctly sequence a set of components which will achieve their desired tasks, the ability to discover the task which each component performs is of paramount importance. Inspecting the metadata ontology for information relating to the task which a component performs takes advantage of the manner in which this information has been stored in the ontology, namely within a task hierarchy. Representing components as part of a task hierarchy directly relates each component with the tasks it performs, and the hierarchy supports multiple inheritance to satisfy the situation where a component can be used to perform multiple tasks. In addition, by maintaining various levels of abstraction within this hierarchy it is possible for the system to identify components that perform a user's required task even if the user is only able to provide a very high level identification of the task they require.

Information from within the task hierarchy can be retrieved using a combination of JENA functions and SPARQL queries, depending on the direction you wish to traverse the hierarchy. JENA is used to retrieve the "super classes" of a given component, representing the abstract tasks it performs, SPARQL is used to retrieve all of the "sub classes" of a given abstract, representing the components which implement that abstract task

Through these means it is possible to retrieve the task(s) performed by an individual component, but also to retrieve all of the components which are an implementation of a given abstract task. This information can then be utilised to discover whether an

individual component is suitable for inclusion within a workflow composition, and which components are suitable as implementations of a given abstract task.

5.3.3 Retrieving Component Provider, Domain, Project Metadata

As discussed previously, knowledge relating to the domain or project for which a component has been designed and knowledge of the provider or developer of that component are useful in determining whether a component is suitable for inclusion within a workflow, as well as whether two components are desirable to connect together.

Again using SPARQL queries it is possible to retrieve this information from the component metadata ontology. As described in Chapter 4, the Provider, Domain and Project elements of metadata are defined as “restrictions” on the WorkflowComponent class, and are therefore present for all components within the ontology. These restrictions are represented within OWL as triples, e.g: “WorkflowComponent hasComponentProvider some Provider”. This manner of storing component properties as restrictions means that given the component for which metadata is required and the individual restriction which we wish to discover, the extraction of such information can be achieved using simple SPARQL SELECT queries.
Using queries of this type enables the retrieval of any value related to a given component's restrictions. As before, once this information regarding a component’s Provider, Domain, and Project has been discovered it can be used to evaluate that components suitability for inclusion within a workflow composition.

5.3.4 Retrieving Component Port Metadata

As discussed in Chapter 4, metadata relating to a component’s ports is also stored within the ontology as this is also required to accurately determine a component’s suitability for inclusion within a workflow. The “basic type” of a port is required to determine whether two component ports are syntactically compatible with one another; the “port data object” property relates each port to a type of object that is transferred between components connected via that port and is therefore useful in further determining whether components are compatible, and finally the “connection history” of a component and its ports enable the discovery of trends within the connection of

components and ports and is useful in deciding which connections are most likely to be appropriate.

The “basic type” and “data object” elements of metadata are recorded within the ontology as restrictions of the Port class. Therefore their values can be retrieved from the ontology in the same way as used to discover the provider, domain, and project restrictions of a component. The `getPortPropertyValue` method is functionally identical to the `getComponentPropertyValue` method, simply executing a relevant SPARQL query to retrieve the value of a given Port restriction.

However as the “data object” metadata is represented within the ontology as a hierarchy, similar to the component task hierarchy, this element can then be further inspected, identifying its "super classes” to discover where in the hierarchy a given data object resides. This enables the identification of data objects which are closely related but not necessarily identical, and therefore the identification of ports which may be suitable for connection due to dealing with closely related data, but are not exact matches.

The metadata relating to the history of component and port connections is not maintained within the OWL ontology. Instead as described in Chapter 4, this connection history is stored in a data structure which records the components and ports involved in a connection, and the number of times that connection has been made by a given user. In order to retrieve the value representing the number of times a connection has been made this data structure can be searched for a combination of components and ports, and will return the value regarding how many times this combination has been previously connected, if one exists.

Through the methods described within this section it is possible to inspect the knowledge within the component metadata ontology and to retrieve those elements of information that are required in order to determine the suitability of including components and connections within a workflow.

The following sections discuss how, once this information has been retrieved, it can be utilised in order to assess whether components are suitable for inclusion within a

workflow, and how this can then been presented as a set of suggestions for steps the user can take in completing their workflow composition.

5.4 Generating Suggestions

Within existing SWSs the decisions a user makes in order to compose a workflow involve the discovery of information relating to the available components, comparing this with their knowledge of the goals they wish to achieve with their composition, and attempting to decide which components can be used to achieve these goals. The reader will recall that our purpose is to show how a system can assist users in this decision making process, removing the challenge of locating and reasoning with this metadata relating to components and simply presenting the user with the most suitable steps available to them to progress their composition. In this context, the previous section discussed the manner in which this information relating to components can be extracted from the component metadata OWL ontology. In this section we now describe the algorithms utilised to process this information and generate suggestions for the user to consider.

As described in the introduction to this chapter, we envisage three types of suggestion which can be made to the user:

- Suggestions on how to specialise “abstract components”
- Suggestions for new components to insert into the workflow, and
- Suggestions for connections to create between components currently in the workflow

The manner in which each of these categories of suggestion is generated will be explained in the following sections.

5.4.1 Generating Specialisation Suggestions

By providing users with a selection of abstract components with which they can specify the overall goals and tasks that they wish to achieve within their workflow, we aim to overcome the challenge presented by the high level of knowledge required of users when using the manual composition approach provided by existing SWSs. As described

in Chapter 2 these challenges have been highlighted by a number of papers [7, 13, 34, 59, 71, 72, 77, 89, 90].

As described in Chapter 4, a concrete component is an individual which belongs to a subclass of an abstract component. Abstract components exist merely within the component ontology, in contrast to which a concrete component is a direct representation of a real, executable component from with a SWS. In this work the term "Specialisation" has been utilised to define the process through which abstract components selected by the user are converted into concrete, instantiable components. This first category of suggestions is designed to provide the user with choices of how their selection of abstract components can be specialised in order to achieve their overall goals. A workflow can only be executed once all of the abstract components selected by the user are specialised into concrete components and connected in a valid sequence.

As discussed previously the component metadata ontology used in this approach maintains components within a task hierarchy. This relates each component with the task it performs, and it also relates each task with other tasks achieving related or similar goals. Generating suggestions for how to specialise selected abstract components involves the inspection of this task hierarchy to discover those implementable components which are sub-classes of the given abstract component. Furthermore, by maintaining component tasks within a hierarchy it is possible to introduce intermediate steps to this specialisation process, allowing the user to make their selected components iteratively more specialised, investigating the series of sub-tasks related to the abstract task they initially selected, until they have located an implementable component which they are satisfied will achieve their desired outcome.

The previous section highlighted how, given an abstract component, the individual components that implement that abstract component could be discovered. The suggestions for specialising abstract components are generated from this set of components that are sub-classes of the abstract component within the ontology.

For example, if the user were to choose to insert the “ModellingComponent” abstract component into their workflow, inspection of the ontology shows the following as specialisations of that abstract component:

- EcologicalNicheModellingComponent
  - GARPAlgorithm
  - GARPPreSampleLayers
  - GARPPrediction

By utilising the techniques described in the previous section the system is able to retrieve this set of specialisations and is therefore able to present all of these components to the user as options to choose from. The user can select to either convert the abstract component directly to a concrete implementation such as GARPAlgorithm, or take the intermediate step of specialising to another, lower level abstract component such as EcologicalNicheModellingComponent.

If the user chooses to insert a number of abstract components into the composition before they have chosen to specialise these then the process of the system inspecting the ontology to locate further specialisations of that component will repeat, providing the user with suggestions on how to specialise all of their chosen abstract components.

5.4.2 Generating Addition and Connection Suggestions

Once the user begins the process of inserting implementable components into their workflow, the system can begin inspecting these and, based on those components currently present, suggest other components that may be suitable for inclusion within the composition and connections which can be made between those components. In this way if the user is unaware of which further components are required for their composition or the manner in which those components could be connected they can explore the list of suggestions available and discover a route toward completion.

In order to decide which suggestions to provide to the user the system initially takes into account several elements of metadata related to the available workflow components. These are as follows:

- Component Metadata:
    - Provider
    - Domain
    - Project
- Port Metadata:
    - Basic Port Type
    - Data Object

Using this information, in the same manner as the manual approach taken by users in choosing the components and their sequencing required for composition in existing SWS, the system can identify which components and connections are suitable to suggest to the user.

The overall manner in which this identification occurs is similar for the process of suggesting both additional components to add to the composition, and those connections to make between existing components. Suggestions for components to add to a composition are generated using the following steps:

```
For each component present in the workflow composition
{
   Retrieve workflow component metadata
   For each available component in the metadata framework
   {
      Compare workflow component and available component metadata
      If metadata matches
      {
         Add available component to suggestions list
      }
   }
}
```

The process for generating suggestions for connections to create between components already present in the workflow composition follows the same approach, except that rather than comparing the metadata of each component present in the workflow against those in the metadata ontology, they are compared against the other components present in the workflow.

5.4.3 Identifying Matches

As discussed previously the metadata stored within the ontology is intended for use in two different ways: identifying both the compatibility and the desirability of connecting

components. Each of the elements of metadata serves a different purpose in identifying a component's suitability for inclusion or connection within a workflow composition. This suitability is determined based on both the *compatibility* and *desirability* of adding or connecting a component, where compatibility relates to whether a component is capable of simply interacting with others within the composition, and desirability assesses whether such an interaction would produce a helpful result. The compatibility of components is primarily determined by the BasicPortType associated with each of that component's ports. If two ports have a matching BasicPortType then they are deemed to be logically compatible, and can be connected within the SWS without generating a type-mismatch error. For example the GARPAlgorithm component and the GARPPrediction component each have ports which possess the BasicPortType of "string" and can therefore be identified as compatible.

However, this logical compatibility does not necessarily mean that connecting two components will produce a satisfactory output, or that simply using this measure of suitability will generate useful suggestions. Returning to the GARPAlgorithm and GARPPrediction components, the complete set of ports provided by these are as follows:

GARPPrediction

- **hasInput:** ruleSetFilename
  - **hasPortBasicType:** String
- **hasInput:** layerSetFilename
  - **hasPortBasicType:** String
- **hasInput:** outputASCII
  - **hasPortBasicType:** String
- **hasInput:** outputBMP
  - **hasPortBasicType:** String
- **hasOutput:** outputASCIIFilename
  - **hasPortBasicType:** String
- **hasOutput:** outputBMPFilename
  - **hasPortBasicType:** String

GARPAlgorithm

- **hasInput:** cellSetFilename
  - **hasPortBasicType:** String
- **hasInput:** ruleSetFilename
  - **hasPortBasicType:** String
- **hasOutput:** ruleSetFilenameOutput
  - **hasPortBasicType:** String

Each of the ports provided by these components is of the basic type “string”; and therefore a simple suggestion based only on this factor would identify every port of one component as being compatible with every port of the other. This limited checking would result in a large number of suggestions being generated, as well as generating a number of suggestions whose implementation would be inadvisable or meaningless. For example connecting the ruleSetFilenameOutput port of GARPAlgorithm to the outputBMP port of GARPPrediction, while logically compatible, would not provide any useful result when executed.

To this end, further elements of metadata from the ontology are included within the process of generating suggestions in order to further assess the desirability of adding components to the workflow composition.

5.4.4 Incorporating Further Metadata

As detailed in Chapter 4, the DataObject element of metadata relates each component port with the object that is passed between ports when this component is connected. The idea is to further define the information that is involved when ports are connected, for example the GARPAlgorithm component discussed previously has a port ruleSetFilenameOutput, with a BasicType of “string”. This limited information regarding what is involved in this port’s execution can be further defined, as suggested by its portName, to state that the DataObject involved is a “filename”. With the knowledge that GARPPrediction also has ports that involve “filename” DataObjects, a more accurate determination of which ports are compatible between these two components can be achieved, reducing the number of matches between these components as the ports outputASCII and outputBMP are not related to filename DataObjects and therefore not compatible with the remaining ports.

The refinement of the DataObject metadata can be expanded further; these ports can be defined by the *type* of files to which these filenames refer, so that the component properties are now as follows:

GARPPrediction

- **hasInput:** ruleSetFilename
  - **hasPortBasicType:** String
  - **hasPortDataObject:** RulesetFile
- **hasInput:** layersetFilename
  - **hasPortBasicType:** String
  - **hasPortDataObject:** LayersetFile
- **hasOutput:** outputASCIIFilename
  - **hasPortBasicType:** String
  - **hasPortDataObject:** ASCIIFile
- **hasOutput:** outputBMPFilename
  - **hasPortBasicType:** String
  - **hasPortDataObject:** BMPFile

GARPAlgorithm

- **hasInput:** cellSetFilename
  - **hasPortBasicType:** String
  - **hasPortDataObject:** CellsetFile
- **hasInput:** ruleSetFilename
  - **hasPortBasicType:** String
  - **hasPortDataObject:** RulesetFile
- **hasOutput:** ruleSetFilenameOutput
  - **hasPortBasicType:** String
  - **hasPortDataObject:** RulesetFile

Taking this refinement of the DataObject metadata the number of matches between these components ports is further reduced, identifying two matches - between GARPPrediction's ruleSetFilename port and GARPAlgorithm's ruleSetFilenameOutput port, and again between GARPPrediction's ruleSetFilename port and GARPAlgorithm's ruleSetFilename port. By including the PortType metadata associated with each port in this process we can reduce this to only one match, as both GARPPrediction's ruleSetFilename port and GARPAlgorithm's ruleSetFilename port are of the type InputPort and two inputs cannot be connected, leaving the only match between:

GARPPrediction

- **hasInput:** ruleSetFilename
  - **hasPortBasicType:** String
  - **hasPortDataObject:** RulesetFile

GARPAlgorithm

- **hasOutput:** ruleSetFilenameOutput
  - **hasPortBasicType:** String
  - **hasPortDataObject:** RulesetFile

Again, similar to the method of maintaining the tasks performed by each component, recording the DataObject metadata within a hierarchy enables the introduction of several levels of abstraction for the "objects" represented. Taking the previous example

of the GARP components, the Rulesetfile, Layersetfile and Cellsetfile information all refer to XML files, therefore each of these DataObjects can be represented as a sub-class of XMLFile. Similarly BMP File can be represented as a sub-class of ImageFile. In this way partial or potential matches can be identified between ports that possess DataObjects which are closely related on the hierarchy. For example if one port has ImageFile as an output DataObject and another has BMPFile as an input DataObject the system can identify that these may be compatible as BMPFile is a sub-class of ImageFile, or at least identify that only a simple conversion step may be required for these components to interact.

Further to utilising the DataObject metadata in order to support the discovery of suitable components to include in a workflow, knowledge of the Provider, Domain and Project metadata associated with each component is also beneficial. As before, once the basic compatibility check of comparing component ports' "basic type" information has been performed, the provider, domain and project metadata can assist in highlighting those components most desirable to suggest.

Although not necessarily applicable to every component available within a SWS, metadata such as the domain for which a component has been developed, or in which it is most commonly utilised, can help identify those components which when connected together will give beneficial results. For example the GARP components discussed previously have all been developed as part of the SEEK project, for use within the Biodiversity Informatics domain. Including such information within generation of suggestions will assist in identifying that these components are ideal to be composed with one another, as well as being ideal for composition with other components designed for or commonly used within that domain.

Additionally, the inclusion of the history of connections made between components is of assistance in determining which components are desirable to connect in the future. Given a situation such as that described previously where many components are logically compatible, including a user's history of interactions with those components in the suggestion process can identify trends and promote the suggestion of only those components which have previously been connected with one another.

5.4.5 Suggestion Ranking

Whilst metadata such as a component's domain of use or its developer can be of benefit in identifying those components which may be desirable to include or connect within a workflow, the manner in which this metadata influences the suggestions must be carefully handled. A fundamental constraint is that components must be logically compatible with one another. A component which is utilised both within the same domain and project as another, but does not share a compatible basic port type must not be suggested as it will not be compatible, whereas a component that whilst not being of a matching domain or project, does share basic port type compatibility can be suggested as it is compatible.

Metadata relating to the *desirability* of connecting a pair of components to one another is therefore always a secondary factor, and is not used to promote non-compatible components above those which are compatible. The priority of metadata in the ranking of suggestions generated is as follows:

1. Basic Port Type
2. Port Data Object
3. Connection History
4. Domain, Project, Provider

In this way the additional metadata can assist in differentiating between available additions which are of equal logical compatibility, but will avoid logically compatible components from being swamped by those which only possess "desirability" matches.

The ranking of suggestions provided to the user operates as follows:

1. Components are first ranked based on their Basic Port Type:
   - *Those components whose ports match are presented as suggestions to the user, those which do not match are not included in the selection.*

2. The Port Data Objects associated with these suggested components are then inspected:

- *Components whose PortDataObject is an exact match are assigned a ranking of 1*
- *Components whose PortDataObject is a generic match (i.e they both share a super-class that is not the generic BasicType) are assigned a ranking of 2.*
- *Components whose PortDataObject only match on the BasicType are assigned a ranking of 3.*

3. Within their ranking assigned at step 2 suggestions are then ranked based on their connection history:
   - *Components are ranked higher based on the number of times which they have previously been connected*
4. Finally, within the ranking assigned at step 3, suggestions are ranked based on their domain, provider and project metadata
   - *Components which have either a matching domain, provider or project are ranked above those which do not.*

In this way, as far as is possible using the metadata available, the system seeks to provide users first with suggestions of components and connections which are logically sound (they have ports which are compatible, and desirable, they process data of matching types), but where possible they have also been utilised previously and belong to matching domains, projects or providers.

5.5 Summary

This chapter has discussed the manner in which the component metadata ontology outlined in Chapter 4 can be used in order to provide a user with assistance during the process of composing a scientific workflow. A process by which the user is presented with suggestions for steps to take in order to progress their workflow has been introduced, with an outline of the algorithms which generate these suggestions provided. Finally the mechanism through which these suggestions are ranked in order to ensure that the user is presented with suggestions which are perceived as most beneficial to their workflow composition has been described.

6 API Definition and Implementation

This chapter addresses the problem of engineering the assisted workflow composition environment in a way that allows it to be connected to a range of heterogeneous scientific workflow systems. It discusses the design and implementation of an API to facilitate the implementation of software that enables the use of new interfaces to extend the capabilities of existing scientific workflow systems without dependence on those systems' implementation details other than those necessary to implement the API, and assuming that the Java source code is available for inspection. It is this API and the implementation of it as a intermediate layer between the existing SWS which enables the suggestion based approach to composition introduced in the previous chapters to be utilised without modification of a SWS and with more than one existing SWS. The architecture of existing scientific workflow systems will be discussed, establishing the common features presented by these systems. These common features, together with the requirements for a SWS extension, such as the assisted composition approach presented in this thesis, are used to define an intermediate API to act as a mediator between an implemented SWS extension and any existing SWS. Accordingly this API is one of the significant contributions of the present thesis, although it should be noted that we have deliberately restricted the API's scope to those features which are essential to our current purpose.

6.1 Overview

Taking a high level view of existing scientific workflow systems there are a number of common features present across all systems:

- A repository of workflow components from which users can select and compose their workflows,
- A mechanism for asserting connections or links between the data inputs and outputs of these components,
- The capability to coordinate the sequenced execution of these components, providing the user with feedback on any results.

This fundamental similarity between the features of existing workflow systems presents an opportunity for enabling any proposed SWS extension to function in conjunction with a number of existing systems. It is noted that the capabilities of each individual SWS offer functionality beyond the common features identified here, however in order to provide an approach capable of working in co-ordination with multiple SWSs it was deemed appropriate to limit the functionality to these shared features. We discuss this limitation further later in this chapter.

By inspecting the source code of existing SWSs to establish a unified level of functionality exposed by each existing workflow system, along with a clear definition of the functionality to be provided by the SWS extension, an intermediate software wrapper can be developed whose role is to provide a standard interface for our software to communicate with, and act as an extension of, these existing systems. In this way the differing implementations of underlying workflow systems can be hidden behind a common interface, enabling any SWS extension such as our computer-assisted composition approach to provide the same level of functionality across all supported systems.

In this chapter the above-listed common features of SWS are broken down into the individual tasks which a user must perform in order to construct a workflow. The manner in which this core functionality is achieved within the three targeted SWS (Kepler, Triana and Taverna) is described, as this information can be utilised to define the main structure of an intermediate API for communication between a SWS extension and each of the existing SWS. By understanding both the functionality that is needed in our system and the manner in which that functionality is achieved in the underlying systems the API can successfully direct requests to the code in the underlying SWS implementation which can fulfil that request.

In this way the SWS extension becomes decoupled from the existing systems and is able to function without being dependent on one particular implementation. This has the benefit of allowing the same functionality to be achieved across each of the existing systems, even in situations where the implementation of such functionality differs between those systems, as well as enabling alterations and development of the extension to progress without concern over how to interact with the underlying SWS

implementation. The API also means that there need only be a single implementation of our computer-assisted composition SWS extension, rather than developing the same software three times, customised to function with each of the existing systems. Beyond this there is the possibility that the SWS extension can be utilised with further existing, or future, SWS. In this case the only changes needed would be to update the implementation of the intermediate layer based on the API rather than to further specialise the SWS extension.

6.2 Requirements

As mentioned in the introduction, the purpose of the intermediate API is to provide a standard interface through which a given SWS extension can interact with the underlying functionality of existing workflow systems as required. In order to establish such an intermediate API it is necessary to fully define the interfaces between which it is going to operate, in this instance the functionality to be exposed by the existing workflow systems and the functionality that is to be enabled in the SWS extension.

The overall aim for the SWS extension described in this thesis is to provide a new means for the end users of a SWS to compose their workflows, providing automated assistance to the user during this process. Through this the extension seeks to overcome some of the challenges presented by the manual composition approaches taken by existing SWSs. As identified in Chapter 2 these challenges primarily relate to the level of knowledge required of the user to successfully create a workflow: by providing an extension to these existing SWSs that offers assistance during workflow composition we can help to reduce these challenges. To this end the functionality of the extension is fairly straightforward to summarise – anything that is currently required of a SWS when a user is composing a workflow will be required of the extension. Taking the steps that are involved in a user's interaction with existing SWS during composition we can assume a number of requirements, these are shown in the Use Case diagram in Figure 6-1.

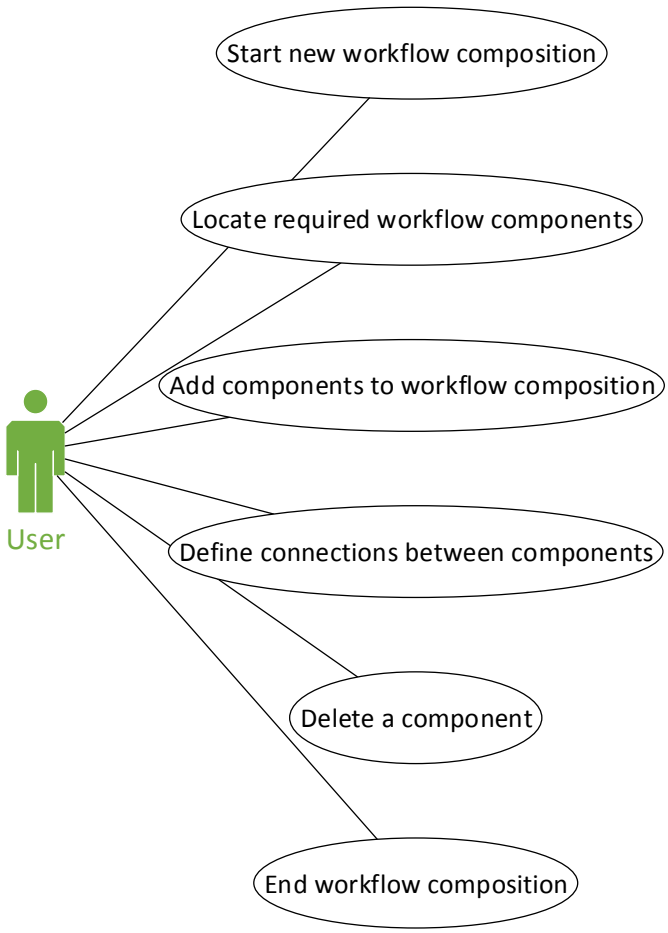


Figure 6-1 Use Case diagram showing key workflow composition activities

Additionally, as well as supporting these composition tasks the SWS extension and API must allow for the execution of a composed workflow within the underlying SWS, including support for the user to input data required during execution and providing feedback regarding the outcome of the execution; whether through reporting mechanisms within the SWS itself, or through displaying the output of any visualisation or reporting components which the user has included within their workflow.

As mentioned previously it is important to note that the SWSs which the extension is intended to interact with do provide the user with further options during composition in addition to those listed above, for example the Kepler system makes available a selection of “Director” components which the user can utilise to alter the manner in which the workflow will execute, and each of the systems provides the capabilities for the user to implement features such as branching and looping over components during

execution into their workflows. Such features are not included within the capabilities of either the extension or the API as it is described in this work. In the case of the "Director" components provided by Kepler this is in order to maintain a generic set of capabilities which the API and extension will support so as to maintain compatibility with each of the chosen SWS. In the case of operations such as introducing loops into the workflow composition, these have been excluded from the API and extension as a means to control the scope within which the work was conducted.

Additionally the manner in which control activities such as looping are achieved can differ between individual workflow compositions, depending on the granularity of the components which are being used. For example some compositions may required the user to specifically implement loops through the SWS interface, others may utilise components which implement the loop within their own processing. Furthermore the specific set of sample workflow operations which have been utilised to evaluate the approach we present do not require features such as branching an looping. However, consideration of how these additional features could be incorporated into the system is included in Chapter 10.

These tasks involved in the composition of workflows are therefore what any approach to composition must be capable of. In order to enable an extension to be able to perform these tasks the necessary functionality of the existing workflow systems can be exploited. To achieve this across a number of existing SWSs we need to identify the specific operations within each existing system that provide the functionality to support the composition requirements identified. Where there are differences in the manner in which these operations are achieved within the existing systems the API must overcome these to present a single, unified interface to the extension. In this way the extension can operate independently of the individual existing SWS.

By breaking down the tasks involved in workflow composition into the underlying operations that are required to achieve them within a workflow system we can create a more detailed Use Case diagram to capture this functionality (Figure 6-2).

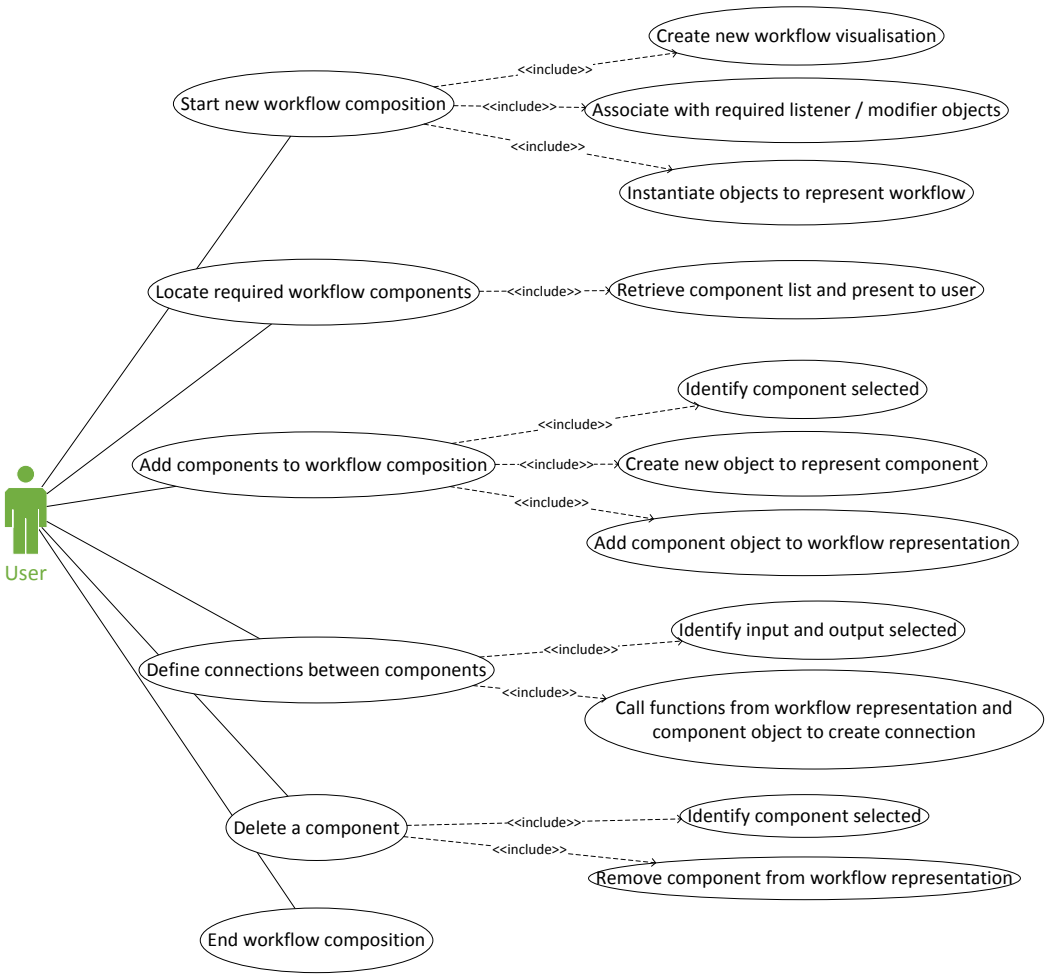


Figure 6-2 Use Case diagram showing functionality required to achieve composition activities

In addition to the composition functionality described in Figure 6-1 three additional key operations which are performed by existing SWSs are as follows:

- Executing a composed workflow
    - Call relevant operation(s) from the object representing the workflow
- Accept user input during execution
    - Display relevant UI elements to accept input
- View the workflow results
    - Retrieve the output provided by “endpoint” component objects of the workflow

The scope of the API must include these additional tasks as it would be expected that any extension which builds upon the API will seek to support this functionality.

This breakdown of the tasks involved in composing a workflow represents the interface between the underlying systems and the proposed SWS extension; the subtasks represent the functionality that must be exposed by the API in order to allow the extension to perform the main tasks. If a user of the extension wishes to add a component to their workflow then the API must translate this call to the relevant operation of the underlying workflow system which can add the component to the current workflow.

Taking the tasks required for workflow composition it is possible to build a model of how each of these tasks is achieved within an existing SWS. This process involves inspecting the implementation of each SWS and identifying the operations, and sub-operations, that are involved in performing each of the tasks required. Once the elements of the implementation that achieve the required functionality have been identified they can then be successfully exploited by a separate, external application.

As discussed in Chapter 3 the three existing workflow systems with which the API is intended to work, Triana, Taverna and Kepler, are all implemented within the Java programming language. As such the API does not need to factor in any possibly more complicated interaction which would be required if working with a SWS developed in an alternative programming language.

The core functions performed by each of the three SWSs discussed in this chapter are highly similar; details of the manner in which this functionality is achieved in each of our identified existing workflow systems are provided in Appendix A. Whilst some differences are present in the manner in which a user achieves these functions when using the system, from the perspective of the underlying implementation, there is sufficient similarity in how these functions are achieved to make it possible to expose this functionality to our proposed extension system through the means of an intermediate API.

6.3 API Implementation

The previous sub-sections have outlined the manner in which the core functionality required by a user when performing workflow composition is achieved in each of our chosen SWSs. In order to make possible the provision of a generic extension across all of these systems an API must be defined which acts as an intermediary between such an extension and each of the underlying systems. Its main role is in providing a uniform interface for the extension to access the functionality it requires from any of the underlying SWSs' implementations. This section describes the functions which are implemented within the API, establishing how the functionality from each SWS can be invoked, based on the details described previously, and how the result of performing this action is then relayed back to the workflow extension.

The main elements of the API are the interface provided to generic workflow system extensions, the controls for identifying which underlying system is to be used during a session, and the routines for exposing the functionality of the underlying systems. The following sections will look at how each of these is achieved within the API.

6.3.1 Interface with Generic Workflow Extensions

The interface between the API and the SWS extension is concerned with enabling a user's actions when interacting with the extension to achieve the desired result within the underlying SWS. In this way requests generated by the user performing some action within the extension are translated into calls upon the appropriate functionality within the underlying SWS in order to achieve the task the user is performing, with the result produced being returned in such a way as to enable the extension to display the outcome.

The requests that the extension can make are based upon its functional requirements that have previously been identified:

- Start a new workflow composition
- Locate available workflow components
- Add components to the workflow
- Define connections between components in the workflow

- Execute a composed workflow
- Accept input from the user during execution
- View the workflow results

From these requirements a set of standard calls to the API have been defined which allow the extension to access the functionality of the underlying SWS that satisfies these requirements. It is through standardising this aspect of the interface between the API and the extension that the extension can operate independently of each of the existing SWS, achieving the same functionality from each with this set of standard calls to the API.

The set of calls that the extension can make to the API are defined in Table 6-1. Each is designed either to completely achieve one of the above functional requirements, or to achieve a sub-goal of the requirements:

API Call	Function
setup(selectedSystem)	Initial setup to establish which underlying SWS is to be utilised
newWorkflow()	Create a new blank workflow representation for use
getWorkflowVisualisation()	Return a visual Java component which provides a system dependent visualisation of the created workflow
getComponentList()	Retrieve the list of components available for composition within the SWS
addComponent(component)	Adds a new instance of the selected component to the current workflow
connectComponents(componentA, componentB)	Create a connection between two components within the workflow
executeWorkflow()	Executes the composed workflow

Table 6-1 Overview of functionality exposed by API

The number of calls presented by the API is limited, since the main functionality involved in maintaining and interacting with the workflow is still handled by the underlying system. At this level the API is effectively acting as a proxy to enable a SWS extension such as our computer-assisted composition interface (see Chapter 7) to access the functions required from the existing SWS to achieve the goals of composition.

These calls enable the SWS extension to access the operations of the underlying systems that are required to achieve the necessary composition tasks for which the SWS extension has been designed. The role of the API in this instance is to translate each call from the SWS extension into the necessary calls to the appropriate underlying workflow system, as defined by the setup() call. The following section looks in detail at the manner in which the API is used to perform this translation.

6.3.2 Exposing Underlying Systems' Functionality

Given the explanation in the previous section of how each element of functionality is achieved in the underlying workflow systems, the API is designed as three strands, one for each of the underlying SWS - Kepler, Triana and Taverna. This arrangement is illustrated in Figure 6-3.

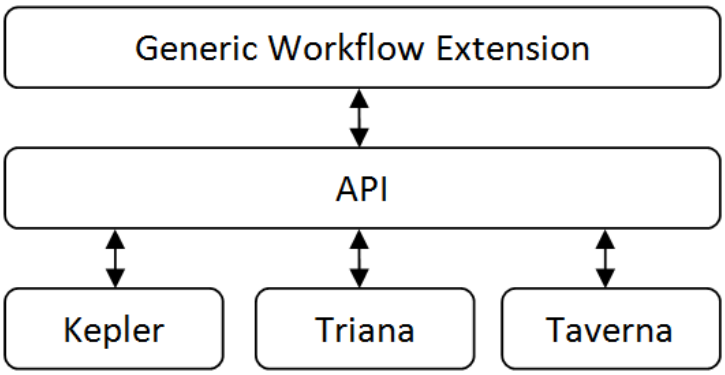


Figure 6-3 Architecture of the API

A single implementation of the API has been developed consisting of three separate strands, as opposed to an implementation per SWS, the decision was taken use this approach so as to reduce the effort required in modularising the implementation, enabling easier reuse of code between each strand, and in order to facilitate possible future development to support workflow composition with additional SWS, or where a SWS extension may wish to interact with multiple SWSs simultaneously, for example if composing a workflow where control passes between two systems during execution. Consideration of such developments is provided in Chapter 10. Within these strands the API contains the appropriate calls to the underlying system that will achieve the

functional requirements, as well as any additional implementation required to enable the results returned to be compatible with the extension.

During the initial setup() call to the API the extension defines which underlying SWS is to be used, setting a flag in the API to reflect this choice of SWS. From this point on all calls from the extension to the API are passed through the strand relating to that SWS.

The following sections describe how each of the functions exposed to the workflow extension are implemented within the API in order to achieve the required functionality from each of the underlying SWS.

6.3.2.1 Creating a New Workflow

To create a new workflow composition the SWS extension invokes the newWorkflow() function, as this is simply informing the API that we wish to create a new workflow there are no further parameters which need to be passed. On receiving this command the API will invoke the newWorkflow() implementation for the SWS which has been determined by the setup() call.

As discussed in the previous section each of the SWSs utilises its own individual data structures to represent the workflow being constructed. In creating a new workflow the API must simply invoke the appropriate commands to create new Objects of the Classes which represent those data structures. For Triana the API will perform the relevant functions to establish a new TaskGraph to represent the workflow, and for Taverna the API will create a new ScuflModel.

To achieve the same in the Kepler SWS the API will first create new Workspace and TypedCompositeActor objects to represent the workflow, however as discussed in Section 6.3.1 establishing a new workflow in Kepler also requires the creation of Configuration, Manager and Director objects. The Configuration object is created from a Specification, an XML file containing various configuration parameters.

The action of starting a new workflow does not require any response to be relayed back to the workflow extension, other than the confirmation that the new workflow has been created successfully.

6.3.2.2 Acquiring a Workflow Visualisation

As with creating a new workflow the process of generating a visualisation of that workflow only requires the workflow extension to call the function getWorkflowVisualisation() with no further parameters required. Once this has been called the API will again call the functionality of the relevant underlying system as dictated by the setup() function.

In order to enable the workflow extension to display the workflow visualisation the role of the API in this instance is to provide a response to the extension in the form of a visual Java component which can be used to display the workflow as the extension sees fit. To this end each of the visualisations created in the underlying SWS will be wrapped in a JPanel object.

As described in Section 3.3.1 the Triana SWS allows for the creation of a JPanel representing the workflow through the ApplicationFrame function AddParentTaskGraphPanel(). In Taverna an instance of the ScuflSVGDiagram class can be created and using the function attachToModel() with the current ScuflModel as a parameter, this ScuflSVGDiagram can then be added to a JPanel before being returned to the workflow extension. Similarly in Kepler an ActorGraphFrame class can be created from the current workflow and using the function getJGraph() a JGraph can be created which represents the current workflow composition. This JGraph can again be added to a JPanel and returned to the workflow extension for display. As this visualisation is constructed from the same classes used to display the workflow composition within the underlying SWSs the user of the extension will be able to interact with it in the same manner as they would in the original SWS, for example to adjust the scale of the visualisation or to re-arrange components.

6.3.2.3 Retrieving a List of Components

As with the previous functions the process of retrieving a list of available components from the chosen SWS does not require the workflow extension to include any additional information in its call to the function getComponentList().

Similar to the API's role in unifying the response to the getWorkflowVisualisation() function call, here the API must ensure that the list of components which is returned to the workflow extension is presented in a standard format irrespective of which SWS is providing the information. For the purposes of displaying the list of components within the extension this information is provided as a JPanel.

Section 3.3.2 describes the manner in which each SWS maintains its list of available workflow components, a ToolTable Object for Triana, a ScavengerTreePanel for Triana, and an EntityTreeModel for Kepler. As each of these objects are visual Java Swing components they can be simply added to a JPanel and this can then be passed back to the workflow extension for displaying to the user.

6.3.2.4 Adding a Component to the Workflow

Unlike the previous operations the addComponent function requires the workflow extension to provide additional information when called. In order to instruct the API, and consequently the underlying SWS, to insert a component into the current workflow composition the extension must include the name of the component to be added in the function call.

Section 3.3.3 describes the functionality which is required to insert components into the current workflow composition. In both Triana and Taverna this is a relatively simple operation with the name of the component to be added being used to create a new instance of this component (as an instance of Tool in Triana and Processor in Taverna) before inserting this new component into the current workflow.

However, as described in Section 3.3.3, Kepler does not provide a straightforward mechanism for inserting components into the workflow. New components in Kepler are created by constructing a new object of the Actor class which represents that component, however there is no function provided with which to achieve this. As a result the code from the API must generate a constructor based on the name of the class to be instantiated, this is achieved through the following steps:

```
Class[] proto = new Class[2];
proto[0] = CompositeEntity.class;
proto[1] = String.class;

Object[] params = new Object[2];
params[0] = myWorkflow;
params[1] = componentName;

Class actorClass = Class.forName(componentName);
Constructor ct = actorClass.getConstructor(proto);
AtomicActor actor = (AtomicActor) ct.newInstance(params);
```

Here myWorkflow represents the TypedCompositeActor which we have created to represent the current workflow composition and componentName is the name of the component which is to be added to the composition as passed to the API by the workflow extension. Executing this code will generate a new Actor object for the chosen component and add it to the current workflow.

As with the function newWorkflow() the only information which the API returns to the workflow extension is the confirmation that the operation completed successfully, following this the workflow visualisation is then be updated to display the new component.

6.3.2.5 Connecting Components

The function connectComponents() is called by the extension in order to create a connection between two components which have been inserted into the workflow composition. As with addComponent() this function requires the extension to pass information to the API relating to the components which are to be connected.

The extension must provide the API the names of both the source and destination components which are to be connected, along with the relevant “ports” from those components which are to act as the point of connection.

Section 3.3.4 details the mechanism through which this functionality is achieved within each SWS. Whilst there are differences between these approaches from a user perspective the underlying implementations are similar. In the case of Triana a

CableInterface object is created using the two component ports as arguments, in Taverna a new DataConstraint object is created again using the component ports as arguments and confirmed using the addDataConstraint function, and in Kepler IOPort objects are created to represent each component - port pair and then connected using the TypedCompositeActor function connect().

Again as this function is simply altering the state of the current composition there is no need to return any information to the workflow extension beyond the confirmation that the action was successful.

6.3.2.6 Executing the Workflow Composition

In order to execute the composed workflow the generic workflow extension must call the function executeWorkflow() from the API. As discussed in Section 3.3.5 each of the identified SWSs has different approaches to achieving this functionality from the point of view of the end user, however from the perspective of the underlying implementation there are sufficient similarities to enable the API to successfully achieve this operation in each of the systems. As this function is simply instructing the SWS to execute the workflow which the user has been composing there is no requirement for the extension to supply additional information when calling the executeWorkflow() function.

A similar process is followed when executing a workflow within each of the SWSs, involving the creation of a new object which is used to control the execution of the workflow through the use of one of its functions. When interacting with the Triana system the API must create a new instance of the LocalServer class, passing it the TaskGraph representing the workflow we wish to execute, and the ToolTable which represents the components available within the system. Once created the run() function of the LocalServer can be called to execute the workflow.

As discussed in Section 3.3.5, to execute a workflow in the Taverna system the API must create a new EnactorProxy object and in turn use this to create a new WorkflowInstance and run its compileWorkflow(). In Kepler the object created to execute the workflow is called the Manager and its run() function is used to initiate the execution.

The processes described for executing workflows in both the Triana and Kepler system will automatically generate new windows within which any output from that workflow execution will be displayed. In order for the functionality provided by the API to remain uniform across each system it is therefore necessary for the API to ensure that a call to the executeWorkflow function will also generate windows to display the results of execution when Taverna is used as the underlying SWS. As described in Section 3.3.6 this is achieved through the use of the EnactorInvocation class. In order to create a new window displaying the workflow output the API creates a new object of the EnactorInvocation class, using the WorkflowInstance object created in order to execute the workflow. As the EnactorInvocation class is a sub-class of the Java JPanel class the API is then able to create a new window and add this EnactorInvocation to its contents in order to display the outcome of the workflow execution for the user.

Beyond performing the necessary operations to display the outcome of the workflow execution to the user, the executeWorkflow() function also returns confirmation of whether the execution operation was successful to the extension.

Through implementation of the functions described in the previous sections the API is able to access the necessary functionality from within the underlying SWS and translate the results of this, in a manner consistent for each system, back to a generic workflow extension. In this way it is possible for the user to be presented with a workflow extension which can offer features and functionality beyond that of the existing SWS and through interacting with that extension achieve their goals of composing and executing a workflow composition irrespective of which SWS is being used to perform the actual resource composition and execution.

6.4 Summary

This chapter has presented the concept of a SWS extension as software which can be utilised by a user to alter and improve the means through which they interact with SWSs. In order to facilitate the use of such SWS extensions, and in order to enable a user to make use of these extensions irrespective of the SWS which is to perform the composition, this chapter has discussed the introduction of an intermediate API to expose the functionality of each of the identified SWSs in a uniform manner.

By identifying the key steps that a user takes when interacting with each of the existing SWSs in order to compose a workflow, and determining the manner in which this functionality is implemented within each SWS, this chapter has defined the set of functions which are provided by the API to a generic workflow extension and has outlined how these functions are implemented in such a way as to achieve the desired functionality within each of the existing SWSs.

7 User Interface

The previous chapters have discussed how an intermediate software layer, implementing a common API, can be used to enable generic extensions to be attached to a number of existing SWSs; how a structured framework of metadata about the components available in those SWSs can be defined, and also how a number of approaches can be used to allow this metadata to assist users in completing their workflow compositions. This chapter introduces a simple user interface which has been implemented to illustrate how each of the ideas discussed in the previous chapters can be used to support workflow composition. As mentioned previously this user interface is not claimed to be the best approach to delivering a suggestion based composition system; it is merely intended as a mechanism to allow us to evaluate the extent to which the ideas explored in this research are effective.

7.1 Overview

The role of the user interface (UI) presented here is to allow a user to compose a workflow by making use of the techniques discussed in the previous chapters. The UI makes use of the generic workflow extension API discussed in Chapter 3 to allow the same interface to sit on top of multiple existing scientific workflow engines, thereby allowing the user to compose workflows using the same methods irrespective of which SWS is being used. The component metadata framework and suggestion generation techniques described in Chapters 4 and 5 are then used to provide the user with suggestions for steps they can take to complete their workflow composition.

In designing the UI, familiarity has been regarded as desirable: the general appearance and approach to workflow composition provided by the UI are similar to those of existing SWSs such as Triana, Kepler and, to a lesser extent, Taverna. The user is provided with a list of items which they may add to their workflow and a canvas upon which to arrange and connect those items. In addition the user has access to controls which allow them to control the execution of their workflow. Figure 7-1 is an image of the current user interface provided by the Kepler SWS, highlighting these main elements.

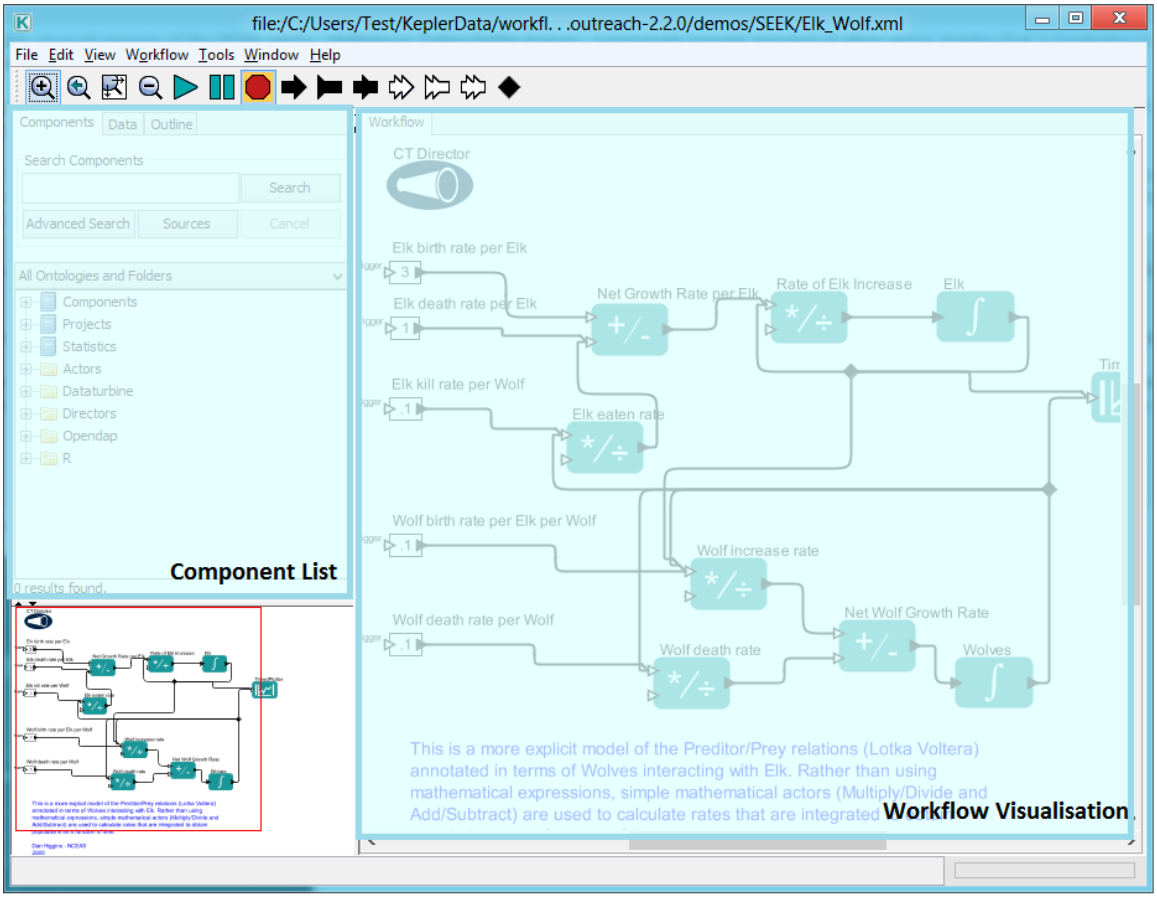


Figure 7-1 Overview of the Kepler UI highlighting the visualisation and component list elements of the interface.

Where the UI developed in this thesis differs from existing ones is in the introduction of a section of the display within which suggestions for steps to take based on the current state of the workflow composition can be shown. This takes the form of a panel on the screen which features three tabs, one each for providing suggestions for additional components to be added to the composition, specialisations of existing abstract components that have already been added to the composition, and connections which can be made between the components that have already been added to the composition. In each case the suggestions take the form of an ordered list, utilising the approach to suggestion ranking described in Chapter 5 to highlight the most useful actions that the user could take in order to progress their workflow composition. Figure 7-2 is an image of the prototype computer assisted composition UI developed for this work highlighting the panel for providing composition suggestions.

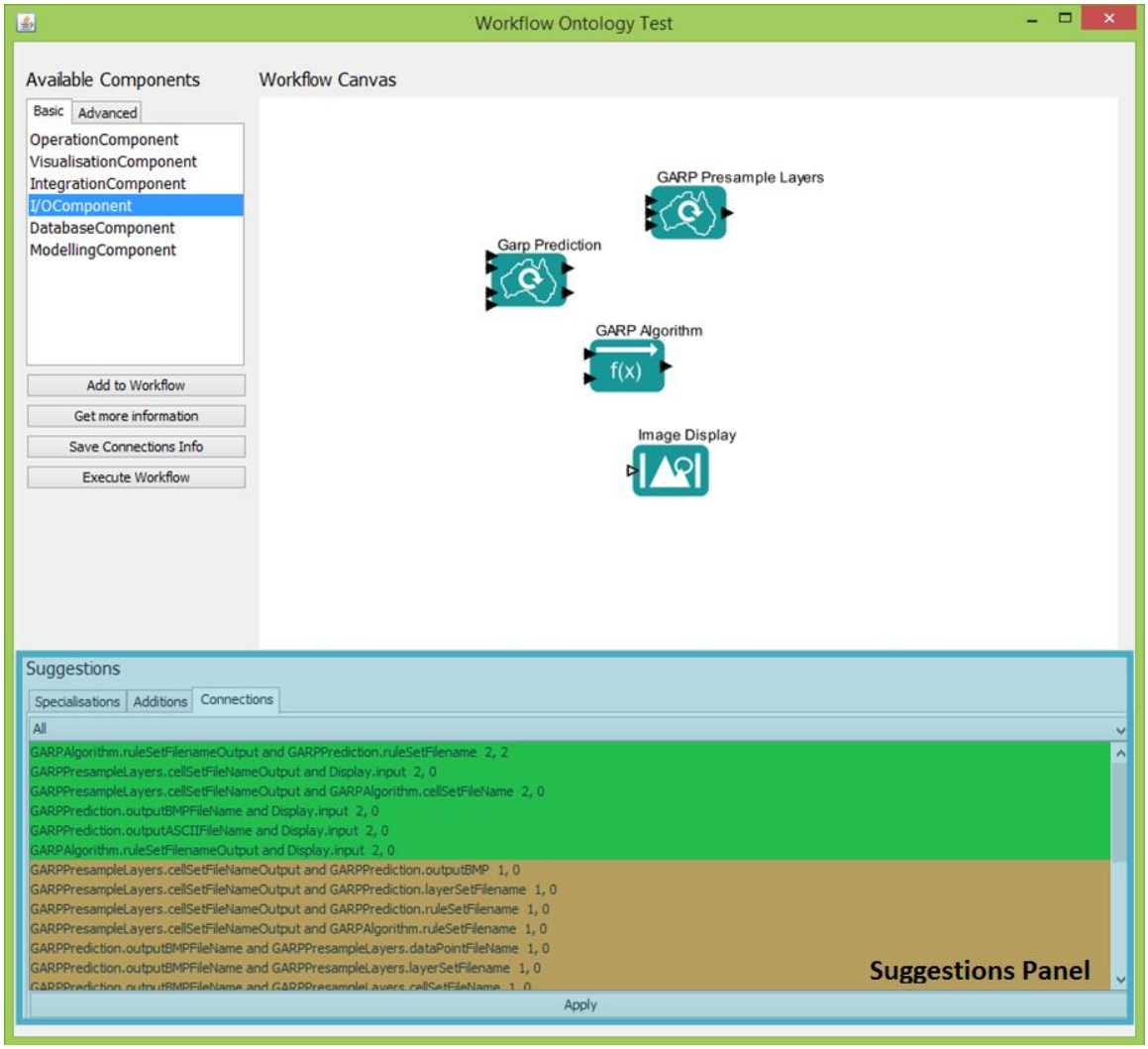


Figure 7-2 Overview of the computer assisted composition UI highlighting the suggestions panel

In this way the UI enables the approach to workflow composition assistance through suggestions generated from component metadata to be explored and evaluated.

7.2 Development

In order to demonstrate the capability of metadata based suggestions to support the composition of scientific workflows, the UI developed in this work must achieve a number of basic tasks, some common to the functionality of existing SWSs, but others specific to this assisted approach to composition. These tasks are as follows:

1. Allow the user to select components to use within a workflow composition

2. Provide a space (such as a “canvas”) for the user to specify the sequencing and connections between those components
3. Enable the execution of composed workflows
4. Allow the user to select abstract components to use within the workflow composition
5. Inspect the component metadata ontology for relevant information
6. Utilise the algorithms described in Chapter 5 to reason with this metadata to generate suggestions for steps the user can take to develop their workflow composition
7. Display these suggestions in a manner which highlights the most relevant steps that the user could take
8. Allow the user to select which suggestions to implement.

7.2.1 Providing Common Functionality

The first three tasks are common activities which the user would perform within an existing SWS; as a result this represents functionality that the UI can utilise from those existing SWSs through the use of the API described in Chapter 6.

As the API provides common functions to access information such as the components provided by a given SWS, or the manner in which that SWS creates a visualisation of a workflow composition, the SWS extension can simply invoke these functions to provide the user access to this common functionality. For example the API call `getComponentsList` will return a visual Java component which can then be simply added to the overall user interface layout to allow the system to display all of the components available within the chosen SWS. Similarly the API call `getWorkflowVisualisation` will return another visual Java component which can be added to the UI to allow the inclusion of a workflow visualisation. Figure 7-3 is an image of the UI which highlights each of these areas of the interface.

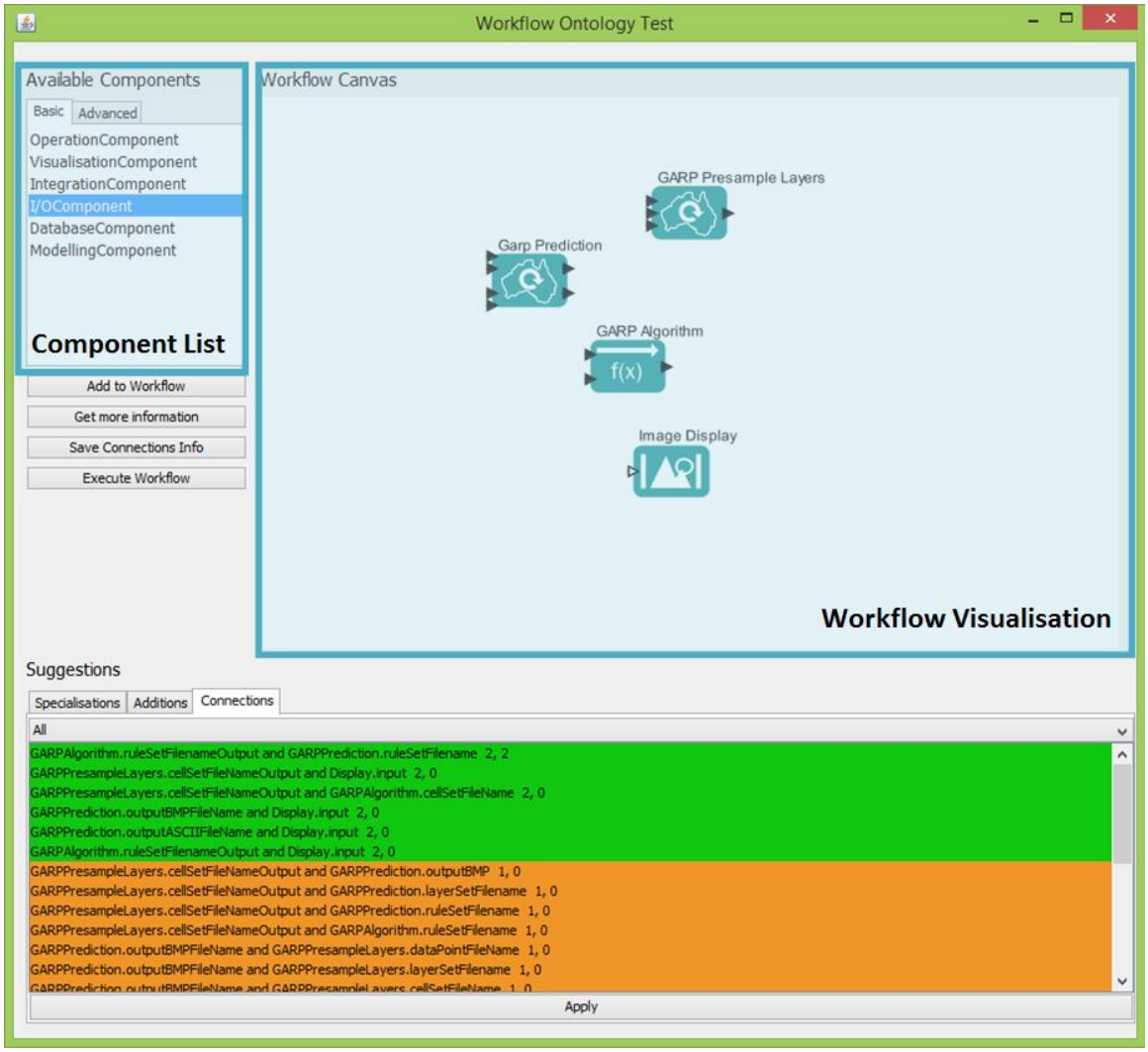


Figure 7-3 Computer assisted composition UI highlighting the component list and visualisation

By utilising these API calls the basic UI layout can be provided, including functionality such as the ability to add items from the component list to the workflow composition, and to implement connections between those components. As discussed in Chapter 3, by including a separate API to provide this functionality these common UI features can be included in the same way irrespective of which underlying SWS is being utilised.

7.2.2 Providing Extended Functionality

In order to demonstrate the assisted approach to workflow composition described in Chapters 3 – 5 the UI must provide additional features beyond those already displayed

in existing SWSs. Features such as the ability to display suggestions for actions which the user can take to develop their workflow, and the ability to insert abstract components into a workflow composition are required of a computer-assisted SWS extension.

As discussed in Chapter 5, the JENA framework allows for basic queries to be submitted to an OWL ontology. This approach can therefore be leveraged by the SWS extension to access information from our component metadata ontology. An initial prototype user interface was constructed which made use of such querying to allow for basic information to be displayed within the UI such as all of the components which hold a certain property, or all components which belong to a certain scientific domain. Whilst this provides some benefit for the user when interacting with the UI, the main benefit is designed to come from presenting the user with suggestions for steps they can implement to progress their workflow composition.

As described in Chapter 5, the basic information present in the component metadata framework can be utilised, in combination with information about the user and the current state of their composition, to identify additional components which could be of benefit if inserted into the composition, and connections which could be made between those components already present. By using the algorithms from Chapter 5 the UI is able to present the user with a list of such suggestions. In order to maximise the benefit of these suggestions the UI presents a panel which organises suggestions into three tabs: suggestions for components to add to the workflow, components to connect within the workflow, and specialisations of abstract components which the user has inserted into their workflow. As the algorithms detailed in Chapter 5 also focus on identifying how useful each suggestion may be if implemented, the UI presents these suggestions as a ranked list, with the most useful suggestions on top. In addition a colour scheme is used to further highlight which suggestions may be of most interest; with those of most benefit coloured green, followed by amber and red as the perceived benefit of each suggestion reduces.

7.3 Creating workflows via the new User Interface

Using the user interface elements described in the previous section the extension UI allows the user to perform all of the tasks they need to successfully compose

workflows. This section outlines the manner in which the UI can be used to compose a workflow, and details the sequence of actions which a user would perform in order to achieve this.

7.3.1 Identifying Components

As shown in Figure 7-3 the user interface provides a panel which allows the user to select components for inclusion within their workflow. The panel includes two tabs. The first tab lists each of the components available in the current SWS, presented as they would be in the original SWS – for example the Kepler components are listed under headings which represent their function or their source. The second tab lists components as they appear in the abstraction hierarchy within the component metadata framework.

When starting to insert components into their workflow the user can either use the first tab to select specific components and place them in the workflow, or use the second tab to insert abstract components which represent the overall activity they wish to achieve, with the capability to specialise these components as the composition progresses.

This two-tabbed system allows the user to insert components into the composition which represent their knowledge of what is required to fulfil their needs. For example if the user already knows all of the components that are required then the UI allows them to insert all of these directly into the workflow. However if they are unsure of a number of components required to complete some steps of their requirements, then they are able to select abstract components which represent those steps. Finally if the user is unsure of any of the components required to complete their requirements, then they can simply select a number of abstract components with the system then assisting in turning these into the concrete components required.

7.3.2 Creating Connections

As discussed previously, connections between components are established using the same drag and drop approach as used in existing SWSs. Once added to the composition, components will appear in the workflow visualisation section of the UI.

From here the user is able to re-arrange those components as well as create connections between them by dragging lines between the required input and output ports of those components.

In addition the system provides the user with a list of suggestions for connections which could be made between those components which currently exist within the workflow, the user can choose to implement any one of these suggestions and the system will create the desired connection.

7.3.3 Utilising Suggestions

As outlined in the previous section, the UI includes a panel at the bottom of the screen which provides the user with suggestions for steps which they could take to further develop their workflow. These suggestions are provided in a list which ranks them based on their potential usefulness.

Based on the current state of the workflow composition the UI will be able to display different types of suggestions. If the user has inserted an abstract component into the workflow then the system will provide suggestions for how this component could be specialised into a concrete, run-able component. If the user has already inserted a concrete component into the workflow then the system will provide suggestions for other components that could be added to the composition based on their ability to interact with the existing component. Finally, if the user has inserted a number of concrete components into the workflow then the system will provide suggestions on how, if possible, those components might be connected.

The user is able to explore these suggestions from within the UI to discover the change that they would like to implement. As described previously the suggestions panel is split across three tabs, one each for addition, connection, and specialisation suggestions. Once the user has selected a suggestion they must press the “Implement” button to insert this change into their composition. This will update the status of the composition to reflect the change which has been made, refresh the workflow visualisation panel to display the updated composition, and generate a new set of suggestions based on the new state of the workflow.

The user is free to continue making further changes to their composition by selecting further suggestions to implement, or alternatively by manually selecting components to add using the components panel and creating connections between components using the visualisation panel.

7.4 Summary

This chapter has introduced the user interface which has been developed to demonstrate the capability of metadata generated suggestions to provide assistance to a user during workflow composition.

In order for a SWS extension to enable computer-assisted composition of workflows using the capabilities of existing SWSs, the API described in Chapter 6 has been used. The SWS extension which has been developed presents the functionality of the existing SWS within a new UI, using the API to allow the user to utilise existing functionality such as adding and connecting components manually, as well as using the metadata ontology and suggestion algorithms from Chapters 4 and 5 to allow the SWS extension to prompt the user with suggestions for components to specialise, add and connect within their workflow. The manner in which this functionality is achieved within the prototype computer assisted composition UI has been outlined and the UI has been demonstrated as being suitable for testing the suggestion based approach to workflow composition outlined in the previous chapters.

8 Evaluation

This chapter discusses the manner in which the knowledge-based approach to workflow composition outlined in the previous chapters has been evaluated. This computer assisted approach to composition is evaluated with respect to two primary factors:

- The ability to support the composition of workflows. *This is established through describing the process by which a selection of workflow scenarios can be successfully composed using this approach*

- The benefits and drawbacks this approach presents with respect to existing SWSs. *This is investigated through a comparison of how effectively these existing systems support the composition of the same workflow scenarios.*

The approach taken in both instances is to provide a walk-through of steps required to compose a given workflow scenario using either the computer-assisted workflow composition approach presented in this thesis, or the manual approach provided by existing SWSs, This chapter summarises the outcome of these walk-throughs and discusses the main findings which these show; specific details of the walk-throughs themselves are included in Appendix B. Section 8.1 explores existing approaches to evaluating composition systems to establish that this is a recognised approach.

Additionally the value of each element of metadata utilised in supporting the computer assisted composition approach is established, inspecting how the inclusion or exclusion of each element affects the quality of assistance provided. Finally a minimal user evaluation has been performed by providing the prototype system to Dr Rich Williams, the contact from Microsoft Research Europe, and requesting structured feedback after directing Dr Williams to complete a number of composition scenarios.

Before this evaluation is described the following section examines existing approaches to evaluation of workflow composition systems, looking at how previous SWS projects have evaluated their work, and how approaches to assisted composition in the field of

web services have been evaluated. This review of existing evaluation approaches demonstrates that the technique of using walkthroughs of representative scenarios to establish the effectiveness of a system is appropriate, especially when there is not a large set of users available who can participate in usability trials.

8.1 Existing Approaches

In order to establish a suitable means of evaluating the effectiveness of the computer-assisted composition approach described in this thesis, it is appropriate to consider the means through which other approaches to workflow composition have previously been evaluated, and their applicability to the approach proposed in this thesis.

A common means of establishing the effectiveness of existing approaches to workflow composition has been to utilise a set of scenarios as test cases, illustrating how the approach in question can successfully support the implementation of each scenario. For example Kim *et al.* [12, 39] provide no formal evaluation of the CAT system; instead the work describes the approach taken, with the authors defining criteria to assess whether a workflow is “complete”. These criteria are then utilised to illustrate how the CAT approach can successfully generate complete workflows.

Demonstrating how a number of example compositions can be successfully created when using a given approach is similarly used in the field of web service composition to evaluate the effectiveness of a given approach. For example both McIlraith and Son [45] and Sirin *et al.* [34] illustrate the validity of their respective approaches by providing a walkthrough of how their systems can be used to compose solutions to existing composition problems. A further method of evaluation used by McIlraith and Son is to provide a theorem that will determine the “correctness” of their interpreter, to illustrate that this element of their approach is effective. Weske [53] presents an early thesis on the development of workflow management systems. Here again evaluation consists of providing examples of how the system can be used to successfully manage workflows.

Medjahed *et al.* [24] describe a set of algorithms which can automate the creation of web service compositions when provided with high level descriptions of the required outcome by a user. In addition to using the above evaluation approach of demonstrating how set scenarios can be achieved within the system, Medjahed [40] additionally offers

an analytical model to study the performance of these composition algorithms. This provides a way to evaluate the efficiency of the proposed algorithms based on the total composition time when utilising the algorithms. This is defined as the time taken to complete each of the checking algorithms in sequence; $T = TST + TSS + TDS,$ where TST is the time for checking syntactic composability, TSS is the time for checking static semantic composability and TDS is the time for checking dynamic semantic composability. Medjahed calculates the time taken in both best case and worst case scenarios; the average for each of these times is then taken. Medjahed provides a breakdown of the factors that influence each of these algorithms, illustrating the extent to which they can be considered constant, resulting in a model to calculate composition time based on the number of operations performed.

Plock [41] discusses synthesising programs with respect to “reactive systems”. These are non-terminating programs that continuously receive external input and provide a response. In this work Plock presents a method for synthesising such programs from given requirements using Live Sequence Charts (LSCs). LSCs define the ordered requirements for the finished program and consist of messages and conditions. The approach described takes LSCs as an input and generates a resulting program that satisfies the LSC requirements. Evaluation takes the form of illustrating that a successful program can be synthesised from the given requirements, provided one exists, and also an investigation of the time this takes when provided with LSCs displaying a variety of properties.

Lämmermann [54] discusses an approach to dynamic web service composition that builds upon work from the field of SSP (Structural Synthesis of Programs). The approach presented extends SSP in order to deal with synthesis in a dynamic environment, as opposed to traditional SSP where certain contextual information is required to be specified in advance. Here evaluation again consists of showing example composition scenarios being successfully completed by the given approach, coupled with various “performance measurements” to illustrate how efficient the described approach is. Lämmermann utilises a selection of service composition scenarios that “correspond to practical service composition problems”. Performance measurements provided are based on three aspects – the run time of the synthesis, that of the program

extraction and finally comparing the resulting runtime of a synthesised program against that of a “hand-coded” approach.

Peventhan [55] presents two approaches to assisting workflow development using Grid services. The first makes use of “experiment specific” workflow activities, pre-formed components or sequences of components to represent either specific experiment activities that users would wish to perform or the overall structure of an experiment. Users can modify and combine these to achieve their specific goals. The second approach illustrates how using database management features can assist in the development of workflows. This work uses real world problems relating to wind tunnel experiments as an example scenario to illustrate how the two approaches given are successful.

8.1.1 Selection of Evaluation Method

A common feature in the methods of evaluation used in the research discussed in the previous section is the use of existing workflow scenarios to illustrate the validity of the approach, and a study of how their approach supports the creation of workflows/service compositions which implement these scenarios. Beyond this the evaluation approaches surveyed have also attempted to undertake some kind of performance measurements. Such measurements include how quickly a scenario can be composed, some measure of how *good*, such as how efficient or accurate, a solution presented by the approaches is against one generated in a traditional manner, as well as some idea of performance against any existing approaches that are aiming to do the same.

The approach to evaluation taken within the present work builds upon the methods we have just outlined. The primary approach of establishing whether a composition method is conceptually sound is achieved by illustrating how workflow scenarios can be successfully composed using the assisting approach described in this thesis, with comparisons drawn between this approach and those existing, manual approaches to workflow composition. In addition, in this thesis we analyse effects and benefits of the mechanisms utilised to provide a computer-assisted workflow composition approach, establishing a range of measures to evaluate how effective and useful each element of the approach is in achieving the goal of workflow composition.

Given the foregoing considerations, the aspects of the computer-assisted composition approach which will be evaluated in this chapter are as follows:

- Whether a scenario can be implemented completely by following the suggestions provided by this approach
- How effectively the ranking of the suggestions highlights the components and connections required for a scenario
- How following suggestions can overcome difficulties which may be encountered when composing scenarios using existing SWSs.
- The value that each kind of metadata from the ontology has on the quality of suggestions provided.

8.2 Evaluation

The following sections describe the results of the various methods which have been used to evaluate the metadata assisted approach to workflow composition presented in this thesis. The primary method of evaluation which is used involves identifying a set of realistic workflow composition scenarios and using these to illustrate whether the suggestions which the computer-assisted workflow composition approach provides can direct the user toward the required outcome, and if so, how effectively those suggestions achieve this goal. These same scenarios are then employed to illustrate how using an assisted approach to composition, through the provision of suggestions of steps to take, can provide benefit over a strictly manual composition approach.

In addition, these composition scenarios are also used to identify the value of the various elements of metadata which are utilised to provide composition suggestions to the user. This is achieved by taking key steps in several scenarios and illustrating how the suggestions provided to the user would be affected if different elements of metadata were removed from the suggestion generation process.

8.2.1 Scenarios

The validity of a ‘proof of concept’ approach to evaluating assisted workflow composition is dependent upon how representative the chosen workflow scenarios are. Those workflows must be representative of typical scenarios that will be encountered by

genuine end users. In order to satisfy this requirement the workflow scenarios utilised in this evaluation are a selection of the example workflows which are provided with the Kepler SWS and their selection was made in discussion with an expert in the field (Dr Rich Williams). These examples incorporate a mixture of simple operational scenarios such as retrieving information from files and manipulating data within a workflow, as well as more complex scenarios specific to the domain of biodiversity informatics such as the species distribution example encountered in Section 4.5.

The following sub-sections provide an overview of each of the composition scenarios which have been identified for use within this evaluation.

8.2.1.1 Scenario A

This first scenario represents an example of a simple mathematical process which can be performed using a workflow system. Primarily this involves defining a variable, or set of variables, to use within a mathematical operation, specifying the type of operation to perform, in this instance the Remainder component is used, and providing a means of testing the result of the operation. Although of limited complexity this scenario is relevant as it characterises a workflow which a user may wish to create when familiarising themselves with a SWS and additionally represents the basic building block on which more complex workflows expand. Figure 8-1 represents a completed composition for scenario A.

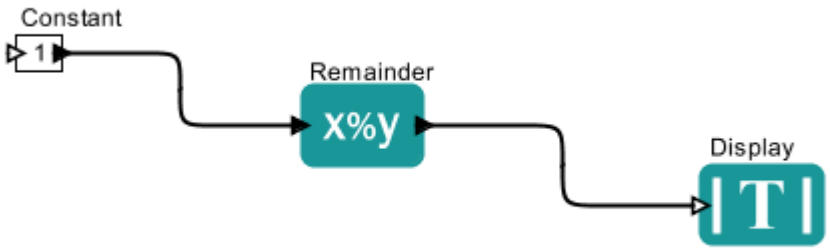


Figure 8-1 Composition Scenario A

8.2.1.2 Scenario B

This scenario represents a workflow task that involves the user including an existing data file as an input to the operation they wish to perform. In this case the task performed is a basic image processing task, reading an image file, rotating the image and displaying the result. As with the previous example this scenario is relatively simple but is a useful demonstration of a basic workflow using domain specific components, in this case image processing. Figure 8-2 represents a completed workflow for this scenario.

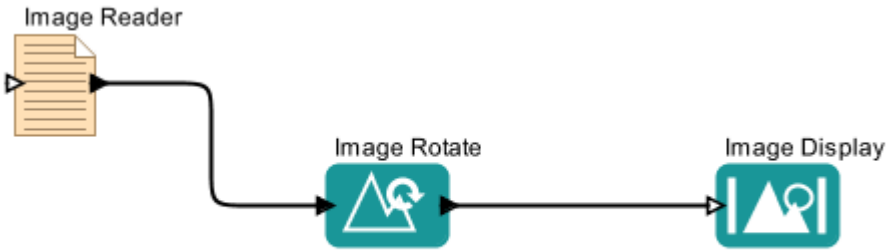


Figure 8-2 Composition Scenario B

8.2.1.3 Scenario C

This scenario is the first which is a specific example from the domain of biodiversity informatics and represents a workflow to model the distribution of a species based on the GARP algorithm. Figure 8-3 represents a completed workflow for this scenario.

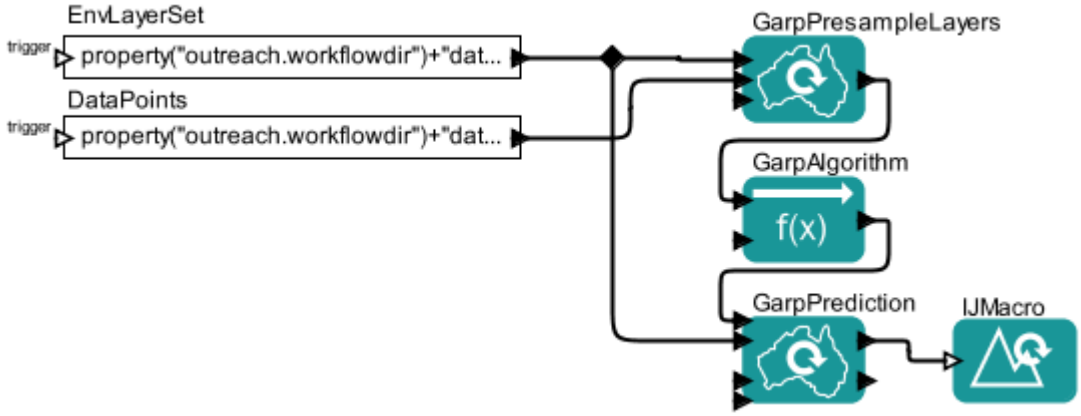


Figure 8-3 Composition Scenario C

8.2.1.4 Scenario D

This scenario is the second example from the domain of biodiversity informatics and represents an extension to Scenario C. Where the input data in Scenario C is hard-coded, scenario D makes use of an external database to read in the species distribution data. Figure 8-4 shows a completed workflow for this scenario.

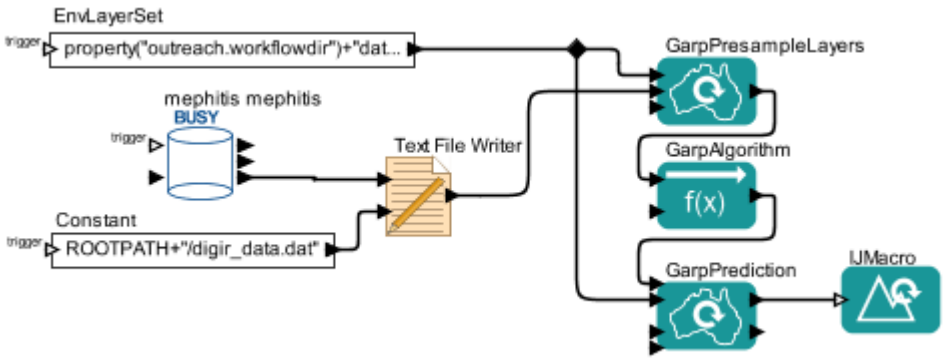


Figure 8-4 Composition Scenario D

8.2.2 Can the Scenarios be Composed Fully?

As described in Chapter 1 the primary focus of the research detailed within this thesis is to establish an approach to scientific workflow composition which builds upon existing SWSs, allowing metadata regarding users, domains and available components to assist in the composition process. As a result, the most fundamental element in evaluating the research is to establish whether this goal has been achieved: can such an approach be utilised to compose scientific workflows?

This section discusses the effectiveness with which the knowledge based approach to workflow composition can be used to successfully compose each of the identified scenarios. As discussed previously, the overall process of composing workflows with this assisting approach is as follows:

- User selects abstract components from those available to define, as much as possible, the structure and goals of their workflow
- System inspects the selected abstract components and provides suggestions for how to specialise these to concrete, executable components

- User selects which suggestions to implement
- System inspects the workflow and provides suggestions for how to connect the components selected
- User selects which suggestions to implement
- Repeat process of suggestions and implementation until workflow is completed to the user's satisfaction.

These steps form the basis of composition within the prototype implementation, with progress occurring through this dialogue between system and user regarding the options that are available to take the workflow towards completion. Figure 8-5 provides an overview of this composition approach that will be followed during the composition of these scenarios.

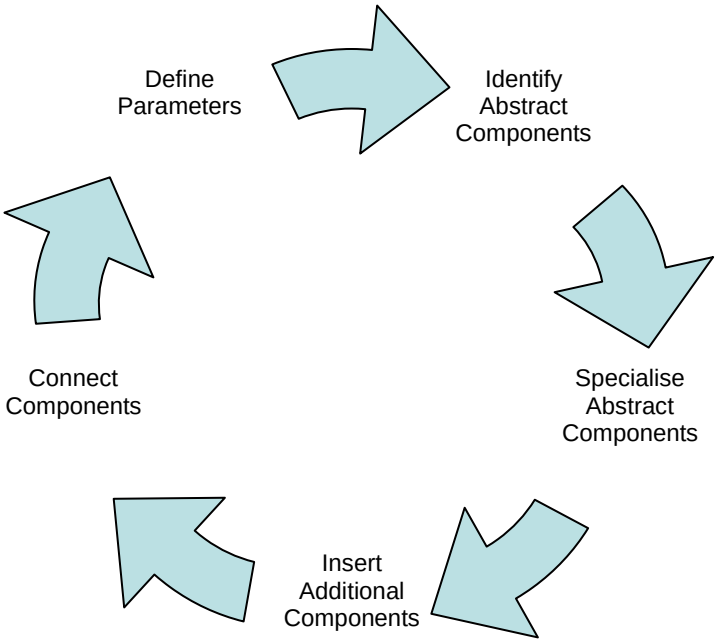


Figure 8-5 Overview of composition process

Detailed walkthroughs of the steps a user would take in order to successfully compose each scenario utilising this approach within the prototype user interface described in Chapter 7 are provided in Appendix B; a discussion of the benefit the approach has during each composition is also provided within this appendix. Overall these walkthroughs demonstrate that the suggestion based approach to workflow composition

described throughout this thesis is able to support the user in successfully creating desired workflow compositions. At a basic level the walk-throughs show how the use of suggestions can reduce the options available to the users, and potentially guide them towards those which may be of most benefit to them. For example in Scenario C the benefit of the PortDataObject metadata is demonstrated; limiting the field of suggested components to add and connect within the composition to only those which deal with processing a compatible form of data to that which is already in use. The benefit of using abstract components to begin composing a scenario is also demonstrated throughout the walkthroughs, enabling the user to select an abstract component which they believe may be of benefit in achieving their goals, and then offering suggestions for how this could be converted into the specific component the user required.

These walkthroughs have also identified some limitations of the approach. Each of the chosen scenarios requires the user to perform some configuration of individual components' properties in order to achieve the desired outcome. Our approach is not currently able to identify that this is required, nor assist the user in the configuration process which may be required to successfully execute their workflow, for example the need to configure the source image to process in Scenario B. Additionally the walkthroughs have illustrated how under certain circumstances the system must rely on the history of interactions a user has completed previously in order to help identify the most useful suggestions, and how without this information the level of support that could be provided would be minimal.

8.2.2.1 Conclusion

This section illustrates through the composition of a number of real world scenarios that the suggestion based composition approach described within this thesis presents a viable approach to scientific workflow composition. The scenarios highlight how presenting users with a selection of high level abstract components can assist in identifying the required components for a composition. Additionally these scenarios illustrate how the use of suggestions can help guide the user through the options that are available to them at each stage of a composition, highlighting those which may be of most benefit in moving a composition closer to achieving a users provided goals.

The following sections look in detail at the benefits this approach provides over the manual composition approach provided by existing SWSs, as well as the effect the metadata stored regarding components has on the quality and effectiveness of suggestions provided.

8.2.3 Comparison with Existing SWSs

Beyond establishing that our computer-assisted composition system is a viable means of successfully composing scientific workflows, it is also of benefit to examine how effective such an approach is in comparison to existing SWSs. Chapters 2 and 4 described some of the drawbacks that have been identified with the approach to composition and support of the user provided by existing SWSs such as Kepler, Triana, and Taverna. This section investigates whether the computer-assisted composition approach outlined in this thesis is of benefit in overcoming or alleviating these drawbacks. This is explored by performing the composition of Scenario C in the Kepler SWS and identifying points during this process where the user may become stuck or make mistakes. These points are then used to demonstrate how the computer-assisted approach to composition could help users overcome or avoid such difficulties.

This evaluation mechanism makes use of only Scenario C. Scenario A and Scenario B are omitted as these are of insufficient complexity to effectively highlight the difficulties that a user may encounter using an existing SWS, and Scenario D is also unused as it's similarity to Scenario C would produce effectively the same results. The Kepler system was utilised for this evaluation as it is from this SWS that the composition scenarios have been selected and it is components from this system which have been used to populate the prototype implementations component ontology. As described in Chapter 4 similar problems have been identified with the composition of workflows within each of the SWSs evaluated, so the improvements identified in the computer-assisted approach will be applicable to each SWS.

8.2.3.1 Effectiveness

In order to establish whether an assisted approach to composition has advantages over the manual approach provided by existing SWSs it is first necessary to define an assessment of the effectiveness of a given approach. In this work effectiveness is defined as a measure of how readily a user can identify the components required for

their composition, the difficulty in identifying how to connect and sequence those components (taking into account factors such as sources and quality of help or information that are provided to assist in achieving these tasks), and the frequency with which a user makes a mistake or becomes stuck during their composition.

A mistake is defined as the inclusion within a workflow of a component or connection that is not necessary to achieve a user's goals; similarly a user is defined as being "stuck" when they are unable to identify a path that can take them closer to their complete workflow and when the assistance or help available within the system is unable to readily assist them in overcoming this obstacle.

This assessment of effectiveness is applied to both the existing SWS Kepler and the prototype assisted composition implementation, measuring how effectively each is able to compose a typical workflow scenario that a user may be required to perform.

8.2.3.2 Kepler Approach

Appendix C provides a walkthrough of the process involved in composing Scenario C using the Kepler SWS, highlighting the steps a user must complete in order to successfully complete the composition, the thought processes that user will go through in order to make decisions for actions to take during the compositions, the information or guidance that the Kepler SWS provides to assist the user in this process, and any problems which are encountered during the composition process.

The problems identified from the walk-through in Appendix C are summarised here, grouped based on the step of the composition during which they occurred:

Step 1: Identifying Components

- **Problem A** - Identifying required components to achieve goals
- **Problem B -** Difficulty using search facility to assist
- **Problem C** - Misleading component listing hierarchy
- **Problem D** - Inconsistency of documentation
- **Problem E -** Difficulty identifying source of input

Step 2: Connecting Components

- **Problem F** - Difficulty correctly sequencing components

Overall the walkthrough illustrates that whilst the Kepler SWS provides a number of facilities to assist in the identification of components required for a composition scenario, and to provide the knowledge required to determine how those components should be sequenced, such as enabling the discovery of the basic type of data processed by each components' ports and providing access to descriptive documentation of each component, these facilities are limited in that each requires considerable effort from the user to locate the information necessary to assist them.

Furthermore several of the problems identified can only be resolved if either the user already possesses the knowledge required to compose the scenario manually, or they are prepared to inspect every available component, port description, and documentation page provided, and based on this information is then able to make the correct decision on how to progress. For novice or inexperienced users the effort required to first inspect this information, and then to come to the appropriate conclusion about how to proceed, could be prohibitive to successfully composing the scenarios they are working with.

8.2.3.3 Computer-Assisted Composition Approach

The following sections will outline how the computer assisted composition approach presented in this thesis can help in avoiding or overcoming the problems identified in the previous section.

8.2.3.3.1 Step 1: Identifying Components

Problems A, B and C

Each of the three problems A, B, and C identified can be alleviated by the inclusion of Abstract Components within the assisted approach. These components are defined at a higher level than standard workflow components, in order to more closely align with the high level goals that users are able to identify as requirements for their composition. By providing a selection of such components the user is able to inspect a smaller list of potential elements to include within their composition, and by relating these components to higher level tasks users can more readily associate these with the goals

of their scenario. In this instance the list of available abstract components provides the following choices:

- Operation Component
- Visualisation Component
- Integration Component
- I/O Component
- Database Component
- Modelling Component

Inspecting this list and comparing the available choices against the user's list of identified tasks for Scenario C, the user can identify the following matches:

- Accessing data files – **I/O Component**
- Performing calculations with modelling algorithms – **Modelling Component**
- Producing a graphical output of the result – **Visualisation Component**

By allowing the user to initially define the workflow at a higher level the challenge of immediately identifying the precise components required to satisfy a scenario is removed, instead allowing the user to translate their list of high level scenario goals into a selection of abstract components within the workflow.

Following the inclusion of these abstract components the system provides guidance for the user in identifying the specific components required to achieve their functionality in the form of specialisation suggestions, a selection of implementable components that satisfy the tasks described by the abstract components. As this list of suggestions is limited to components related to these abstract tasks the challenge of identifying which components to use has immediately been reduced for the user: rather than searching a complete set of available components the user need only inspect a subset of components that are already more likely to achieve their goals.

Problem D

In the manual composition approach presented by Kepler the user wishing to consult component documentation can encounter difficulties due to the inconsistent and

incomplete nature of the information available for each component. In contrast by providing guidance during composition the computer-assisted approach within this thesis alleviates the need for users to manually consult such unstructured documentation. In addition by storing a defined set of metadata across all components within an ontology, if a user does wish to manually consult this information it will be consistent and complete across all components.

Problem E

Problem E highlights one of the drawbacks of the search facility provided by Kepler: it simply searches based on the names of a component. As a result a user could search and locate a component that, if inspecting its name, looks like it could achieve their desired goal, however this component may either be incompatible with the current state of their workflow, or may not achieve the functionality which the user assumes it does.

As stated previously using the search term “File” whilst searching for a suitable component to facilitate the inclusion of a data file as input to the workflow, provides a series of results which, whilst seemingly suitable, would not achieve the user’s goals.

By incorporating metadata about the specific data which each component processes into the system and using this to evaluate which component connections should be suggested to the user, the computer-assisted approach limits the possibility of incompatible components being suggested. As the system inspects each component’s BasicPortType and PortDataObject metadata before including them in the set of suggestions the user can be confident that those suggestions provided are compatible with their composition.

Additionally by only searching on a component’s name the facility provided by the Kepler system can potentially provide the user with a large number of results. For example a search for the term “output” would return over 50 results, with the user potentially being unable to readily identify which of these would achieve their desired goal, or which would be compatible with their composition. Again by incorporating a selection of component metadata into the provision of suggestions the assisted approach is able to either reduce the number of suggestions provided, or ensure that those suggestions of most interest to a user are promoted to the top of the list. The

benefit of each element of metadata in improving the quality of the suggestions the system provides is investigated in Section 8.2.4 of this chapter.

Step 1 Identifying Components Summary

By providing users with a selection of abstract components which can be used to build a workflow at a higher level the assisted composition approach reduces the difficulty of identifying the components that are required to achieve a user's composition. Additionally by then presenting the user with suggestions for how to specialise those abstract components the system supports the user in translating these abstract components into the specific executable components required to achieve their goals.

8.2.3.3.2 Step 2: Connecting Components

Problem F

In the Kepler SWS the user must rely on their own personal knowledge in order to identify the manner in which the components they have included within their workflow are to be sequenced and connected. The only assistance provided by Kepler in order to help a user discover how to connect their selected components is in the description of each component's port types. By inspecting the component documentation a user can discover the types of data produced or consumed by each component and from there can decide whether or not to connect them. This has several disadvantages.

Firstly the user can only identify if the components are "logically" compatible, that their port types match. However, there are many connections of components which have matching port types, but this does not mean that they will necessarily provide useful output if connected. There is limited support for helping a user discover how beneficial a connection may prove within the Kepler SWS.

Secondly, within a composition that includes many unconnected components there may be several potential sequences and connections that can be implemented between them. For example within Scenario C, if inspecting only a component's port types, the user could identify connections between StringConstant and all of the GARP components, yet the Scenario only requires that two of these components be connected

to StringConstant. Again it could prove difficult for the user to identify the correct connections to specify using only the limited information that the Kepler SWS provides.

As the suggestions provided within the assisted approach take into account metadata such as a component's Domain of use, its PortDataObject and its previous history of connections, the system is more able to identify not only those component connections which are compatible but those which are also desirable for a user to implement.

8.2.3.3.3 Summary

In contrast to the manual composition walk-through of Scenario C outlined in Appendix C this section has illustrated how the functionality provided by the assisted composition approach detailed in this thesis can be used to avoid the issues which are encountered when using manual systems such as Kepler. Primarily this is achieved by providing the user with the ability to identify the high level goals which they wish to achieve in their composition and then providing suggestions for how these abstract components can be converted into a completed composition. This process reduces the amount of knowledge that the user must initially possess regarding the specific components and their sequencing which has been demonstrated to be a potential problem when using the traditional composition approach.

8.2.4 Value of Metadata Elements

As described in Chapter 5, the approach to workflow composition investigated in this thesis makes use of a number of elements of information in order to generate suggestions to provide assistance during composition. In particular this approach takes advantage of the history of past interactions a user or group of users have had with the system in order to improve the quality or effectiveness of the suggestions provided.

In order to assess the value that each element of metadata has for the provision of suggestions this section looks at the quality of suggestions provided during the composition of each of the scenarios identified in Section 8.2.1. Initially the value of each element is investigated by inspecting the suggestions that are provided by using that element in isolation, following this the suggestions provided by using multiple

elements of metadata are inspected to discover the value of more complex combinations of metadata.

In addition a further series of scenarios is investigated, this time to discover the value of building a history of user interactions and incorporating this into the provision of suggestions. In this instance workflow scenarios are composed in two different situations, firstly with the recording or interactions enabled and so available to affect the suggestions that are provided in the future, and secondly with this recording disabled and the generation of suggestions based solely on the static elements of metadata within the ontology. By taking a series of workflow composition scenarios from a single domain, such as bioinformatics, and composing these sequentially the benefit of storing users' interactions can be investigated as each successive composition will provide a more complete history of interactions such as would be developed over time by a user working in a single domain.

Whilst no real world user testing has been conducted to demonstrate how a history of user interaction affects the suggestions provided by the system, the scenarios which are being used to replicate such a history are real world scenarios which are used to replicate this interaction history in the absence of real data.

8.2.4.1 Suggestion Quality

In order to assess the benefit each element of metadata has on the suggestions provided it is necessary to first define how to measure the quality of a given set of suggestions. Similar to the concept of 'effectiveness' defined previously, the quality of a set of suggestions is determined using four evaluations:

- **A -** Ratio of ideal suggestions included to ideal suggestions not included.
- **B -** Ratio of compatible to incompatible suggestions provided.
- **C -** Ranking of ideal suggestions within the set.
- **D -** Total number of suggestions provided.

Ideal suggestions are unique to each individual scenario, and to individual steps of those scenarios, and are a representation of the components that are required for a

successful composition of that scenario. For example a successful composition of Scenario C described in Section 8.2.1 involves the components: GARPAlgorithm, GARPPrediction, GARPPresampleLayers, StringConstant, and ImageJ. These components would therefore be identified as ideal suggestions for inclusion in a composition of Scenario C.

Incompatible suggestions are defined as those for components which are not able to be connected to the current state of the workflow. Finally the ranking of ideal components within a suggestion is evaluated as the requirement is for the most ideal components to be ranked above those of less benefit to the scenario.

Appendix D provides a description of how these scores are calculated during composition of a workflow scenario.

8.2.4.2 Value of Static Metadata Elements

The elements of information stored by the system in order to provide suggestions are described in Chapter 4. In summary the primary elements of static metadata utilised for this process are as follows:

- Component Metadata:
  - Provider
  - Domain
  - Project
- Port Metadata:
  - Basic Port Type
  - Data Object

In addition the system also uses the dynamic connection history metadata, built up as the user interacts with the system. The effect this dynamic metadata has on the provision of suggestions is considered separately following the present section.

In this section the scenarios introduced previously are utilised to illustrate the suggestions that would be provided by the system at key points during their composition. Initially the suggestions generated by using each element of metadata in

isolation are explored, to understand the benefit that the metadata elements have individually. Following this the metadata elements are utilised together in the generation of suggestions to identify where the combination of metadata elements has an effect on those suggestions provided.

8.2.4.2.1 Scores for Individual Metadata Elements

Each of the individual elements of metadata the system records about available workflow components can be used in isolation to generate suggestions to assist the user during composition. By taking each of the scenarios introduced previously, generating suggestions based on a single element of metadata individually, and calculating the quality of those suggestions the benefit of each metadata element can be demonstrated.

Appendix E provides details of how the suggestion scores for each scenario have been derived. In summary this calculation involves taking each scenario in turn and performing the following steps:

- Inserting each of the components required for the scenario in turn,
- After each component is inserted generating suggestions based on each individual element of metadata in isolation
- Using the scoring mechanisms outlined in Section 8.2.4.1 to generate a score for the quality of suggestions provided by each metadata element.

Table 8-1 shows the average suggestion scores for each of the scenarios A to C. As noted in Appendix E Scenario D has been omitted from this stage of the evaluation as it is very similar to Scenario C in that essentially the same components are utilised, therefore the suggestions scores presented would not substantially differ from those of Scenario C.

	Scenario A				Scenario B				Scenario C			
	A	**B**	**C**	**D**	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
Provider	100	67	62	141	100	100	58	141	37	60	23	34
Project	0	0	0	0	0	0	0	0	37	60	23	5
Domain	0	0	0	0	100	78	42	9	53	60	45	15
Port Type	100	100	67	7	100	100	58	141	100	100	52	229
Port Data Object	100	100	67	7	100	100	71	4	100	100	62	138

Table 8-1 Average Individual Metadata Suggestions Scores for Scenarios A - C

The general trend demonstrated by the scores is that as the complexity of the workflow scenario being composed increased, so the quality of the suggestions which can be generated by each metadata element in isolation decreases.

The scores generated for mechanism A and B for the relatively simplistic mathematical operation performed in Scenario A demonstrate that the Port Type and Port Data Object elements of metadata are able to suggest the inclusion of all of the ideal components for the composition, as well as ensuring that all suggestions are limited only to those which would be compatible with the current workflow state. In addition the scores for mechanism C and D illustrate the total number of components being suggested is low (7) and the ideal components are ranked relatively highly (67% of the theoretical "best" ranking). Suggestion quality decreases as we move through scenarios B and C. We begin to see the ranking of ideal components decrease and the total number of components being suggested grows, obscuring those components of benefit to the user.

Additionally these scores illustrate that metadata such as a component's Project or Domain is of limited benefit in isolation; as this metadata is not present for all components it is unable to be used to generate suggestions in many cases, demonstrated by no scores being provided for these elements within scenarios A, and only scores for Domain within Scenario B. Similarly Table 8-1 also illustrates that whilst

the Provider element of metadata is able to identify the components required for both scenarios A and B it was not always able to limit the suggestions generated to those compatible with the composition. Furthermore as the complexity of scenarios increases the Provider scores decrease further; only being able to a third of the desired components for Scenario C on average.

As described in Appendix E the use of metadata such as Port Type in isolation can result in problems due to components being associated with a "null" type. In these instances the component is deemed to be compatible with any other component, resulting in a very large list of suggestions being generated.

Overall these scores illustrate that whilst in some scenarios a single element of metadata may be capable of generating useful suggestions there are severe limitations to this approach in general, either with the range of suggestions which are provided, the accuracy of those suggestions, or the ability for those suggestions to effectively highlight the "ideal" components required for a users composition.

8.2.4.2.2 Scores for Combined Metadata

In utilising multiple elements of metadata in the provision of suggestions this quality can be improved, as further information is available to identify the ideal components which can be included in the workflow, as well as to reduce the number of unhelpful suggestions and increase the ranking of those ideal components. Table 8-2 illustrates the subsequent quality scores for the stages of Scenario C previously covered in Appendix E, Table 10-14, when combinations of metadata elements are used in the provision of suggestions. This tables covers the first three components from Scenario C, a complete table including all components can be seen in Appendix H.

	String Constant				**GARPPresampleLayers**				**GARPPrediction**			
	A	**B**	**C**	**D**	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
Individual Average	40	40	17	208	80	100	56	63	100	100	70	39
Combined	100	100	72	124	100	100	100	5	100	100	100	5
Difference	+60	+60	+55	-84	+20	-	+44	-58	-	-	+30	-34

Table 8-2 Comparison of Suggestion Scores for the first three components of Scenario C

These improved scores illustrate how incorporating several elements of metadata into the provision of suggestions can help improve the quality of those suggestions. As an example upon the inclusion of the GarpPresampleLayers component it is desired for the system to identify both the StringConstant and GARPAlgorithm components as ideal for inclusion within the composition. Appendix E describes how when using each element of metadata in isolation to generate suggestions only PortType was able to identify both ideal components, however as the PortType value “String” is very common neither are ranked highly, and whilst the Project, Provider and Domain elements were able to provide GARPAlgorithm within a very small number of suggestions, they were unable to identify the StringConstant component. Table 8-2 shows how using all of the elements in combination the system is able to identify both components, can provide them with a higher ranking than was previously possible, and reduces the total number of suggestions provided substantially.

Table 10-19 from Appendix H shows this improvement in the quality of suggestions provided is reflected across all of the identified key points for Scenario C. An exception is in the case of the ImageJ component. Here the limited information available regarding this component means the combined metadata score remains the same as that of the individual metadata elements.

In order to show the extent to which this improvement in suggestions quality is achieved in various circumstances Table 8-3 shows the average suggestion scores for scenarios A, B, and C, first when using each metadata element individually and then when the combination of metadata is used to generate suggestions.

	Scenario A				**Scenario B**				**Scenario C**			
	A	**B**	**C**	**D**	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
Individual Avg	60	53	40	52	80	76	46	74	65	76	48	132
Combined Avg	100	100	67	7	100	100	71	4	88	100	87	85
Difference	+40	+47	+27	-45	+20	+24	+25	-70	+23	+24	+39	-47

Table 8-3 Comparison of Average Suggestion Scores for Scenarios A and C

Overall introducing multiple elements of metadata into the process of providing suggestions has a beneficial effect on the quality of suggestions provided. Primarily this benefit is shown in improving the ability for the system to identify all of the ideal components required at each stage of the composition; where individual elements of metadata are each able to readily identify a subset of the ideal components it is frequently only when used in combination that all components are identified.

Secondly the inclusion of multiple elements of metadata enables the system to more readily identify those components which are not ideal for the user to include in the workflow, this has the benefit of reducing the number of suggestions provided and therefore reducing the complexity of the work the user has to do in identifying those components which they wish to use.

8.2.4.2.3 Summary

This section has utilised three of the workflow composition scenarios introduced previously to demonstrate the value that each element of component metadata has on the quality of the suggestions that the system is able to provide to the user.

By first illustrating the quality of the suggestions that the system can provide when using each individual element of metadata in isolation this chapter has shown that different elements provide different benefits, some are useful in identifying components which are compatible with one another, and other are beneficial in determining if it is desirable to connect two components together.

In addition by comparing the quality of suggestions provided by each element of metadata in isolation with the quality of those suggestions provided when a combination of metadata is used it has been shown that by using a number of sources of information about workflow components the system can better determine which components are useful for inclusion within a workflow composition. The following section takes this idea further, exploring the extent to which the history of interaction which a user has had with the system can improve the quality of suggestions further.

8.2.4.3 Value of User Interaction History

As discussed in Chapter 4, in addition to the static metadata that is maintained within the component ontology, this approach to providing workflow composition suggestions also makes use of knowledge relating to the history of interactions a user has had with the system. This interaction represents the history of suggestions which the user has implemented over the course of using the approach.

The previous sections have demonstrated that static metadata such as a component's port types, abstract component tasks, and domains of use can be effective at identifying components which are both compatible and desirable to include within a composition. However a record of a user's interaction with the system, the history of component connections and additions they have made, can be of benefit in situations where this static metadata is either too generic or too limited to effectively identify the ideal components to suggest for including within a composition.

Additionally the dynamic nature of this interaction history enables the suggestions provided by the system to improve over time. As a user working within a domain interacts with only a limited set of the available components, and will consistently connect these components in the same manner, by monitoring their interaction the system can begin to favour those suggestions which the user repeatedly chooses to implement.

In order to assess the effect of maintaining a history of a user's interaction with the system the following section describes the suggestions that will be provided by the system during the composition of our previously identified scenarios C and D. Scenarios A and B are omitted from this evaluation as in order to show the benefit of the user interaction history we require scenarios which include common components. As discussed in Section 8.2.4 whilst the interaction history generated in this evaluation is not based on real world user testing, the use of a set of representative scenarios from the bioinformatics domain enables this evaluation to approximate the interaction history of a user from that domain.

Scenario C

Scenario C represents the niche modelling example introduced previously, incorporating the components StringConstant, GARPAlgorithm, GARPPresampleLayers, GARPPrediction and ImageDisplay. This scenario uses species presence data and that relating to environmental factors to provide a visualisation of the geographical distribution of those species.

As discussed in the previous section the use of a combination of static component metadata is able to provide effective suggestions for composition. However, with initially no data relating to users' past preferences for component connections, there are a number of limitations with the suggestions the system can provide. For example when composing Scenario C the suggestions provided upon including the StringConstant component within the composition are unable to effectively highlight that the user may wish to connect this component to either GARPPresampleLayers or GARPAlgorithm – whilst their BasicPortType metadata means they will be included in the list of suggestions, because they are from a different Domain, Provider, and Project to the StringConstant component their ranking within the list of suggestions is low. A similar problem is encountered with the system not effectively highlighting the potential connection between GARPAlgorithm and ImageJ.

Following the composition of this scenario a user from the domain of bioinformatics may be required to compose further scenarios utilising a similar set of components, the following sections illustrates how the connections made during the composition of Scenario C can assist in providing suggestions for this new scenario.

Scenario D

As introduced in Section 8.2.1 Scenario D performs a similar operation to Scenario C, calculating species distributions based on data retrieved from an external database. As this scenario is performing a similar task to that of Scenario C the majority of the components involved are the same. Figure 8-6 is a visualisation of Scenario D.

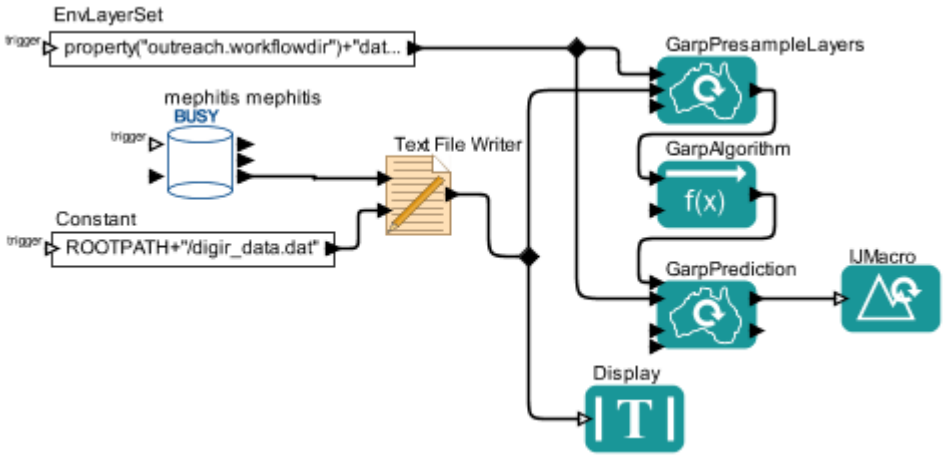


Figure 8-6 Composition Scenario D

As the user has previously interacted with many of the components in Scenario D the system can use this historical information in order to improve the relevance of the suggestions provided.

Incorporating the knowledge of previous connections between the GARP components utilised within this scenario has enabled the system to provide a higher ranking to suggestions involving those components within the composition of Scenario D. Where previously the system was unable to effectively identify the potential for connecting the GARPAlgorithm component to the ImageJ component, during this composition ImageJ is ranked as the number one suggestion, as it is the only component currently available that has been successfully connected to GARPAlgorithm in the past. A similar effect is seen in suggestions provided for the connections between the three GARP components, as these suggestions were already of a high quality the extent to which their previous connection can improve the suggestions is reduced.

However, as this scenario differs from Scenario C in the manner in which the input is provided to the system the suggestions are still not effective in identifying the need to connect the DataAccess component to both GARPPresampleLayers and GARPAlgorithm. Because the system has no connection history relating to these components it must rely on the static metadata in order to provide suggestions. As in the case of StringConstant the static metadata for the DataAccess component shows it to be from a different domain, project, and provider to the GARP components, and

similarly possesses an output port of a generic data type, as a result the system is not able to effectively highlight that GARPPresampleLayers and GARPAlgorithm are the ideal components with which to connect.

8.2.4.3.1 Limitations

This highlights how the use of a user's history of interactions with the system can be of benefit in improving the suggestions provided, however the effect is limited to situations where the user routinely makes use of the same set of components, connected in the same manner. In this way the current system is effectively working as an "autocomplete" facility, directing the user towards decisions which they already know they need to make. When a user introduces new components for which the system has no historical connection data the provision of suggestions must rely on the static metadata available about those components. Additionally if a user frequently makes mistakes, connecting components in a manner which is not beneficial then this will have a detrimental effect on the quality of the suggestions provided.

As briefly discussed in Chapter 4 an interesting extension of this approach which could help mitigate these limitations would be to enable the system to make use of connection history information from multiple users. In this way the situations where the current user has not interacted with components previously may not restrict the ability of the system to generate suggestions, providing these component have been used by another user within the connection history database. This potential development is discussed further in Chapter 10.

8.2.4.3.2 Summary

Static metadata is useful for generating suggestions when working with components which possess similar values for Domain, Project and Provider metadata. However there are instances when the ideal suggestion for connection is a component from a separate domain, in these cases the introduction of a use history of component connections can be of benefit in highlighting such potential component suggestions. Additionally when components produce a large number of suggestions the connection history is a potential factor that can assist in ensuring the ideal components are ranked above those which are less helpful.

However, as demonstrated in the composition of Scenario D, users from a single domain may introduce new components into their compositions after having worked exclusively with a different set of components previously, it is in situations such as these where the connection history is absent and therefore must be developed and can only become of use once the user has worked with the new components for a period of time.

To an extent the use of a user's connection history in this way can be seen as a mechanism to overcome deficiencies in the scope and accuracy of the existing static metadata. If a user from a domain is frequently using a component that is not listed as belonging to that domain, then this indicates that that the metadata for that component should be updated to reflect its common use within the user's domain. A future development for the system could be for components which are regularly used by users from domains that are not recorded in their Domain metadata to have that Domain updated to include the user's domain. Investigation would have to be made into the optimum point at which to decide a components usage in the other domain is common enough to begin altering the static metadata entries for that component.

8.3 Conclusion

This chapter has evaluated the approach to assisted workflow composition described within this thesis through a number of different approaches. Initially a "proof of concept" was performed, taking established example workflow scenarios from the Kepler SWS and illustrating the manner in which these could be successfully composed with the assisted composition approach.

Following these initial walkthroughs a comparison was performed using an existing SWS, Kepler. This process involved illustrating the manner in which the Kepler system could be utilised to compose one of the scenarios previously introduced. During this composition any instances where the user would be unable to readily identify the next step to take, or were required to utilise the assistance mechanisms provided by Kepler, were recorded, providing an overview of where the system presented challenges to the user during composition. This was compared against the assisted composition

approach, highlighting how each of the elements that presented a challenge within the Kepler composition process could be either eliminated or have their effect reduced by the facilities that the assisted composition approach provides.

Finally the benefit of each element of metadata to the provision of suggestions was investigated. This involved the definition of several metrics to establish the “quality” of a set of suggestions provided by the system, these were as follows:

- A - Percentage of “ideal” components suggested
- B - Percentage of incompatible components suggested
- C - Ranking of “ideal” components within the suggestions provided
- D - Total number of suggestions provided (and number of ideal suggestions possible)

Inspecting the potential for each element of metadata to provide suggestions in isolation enabled the value of combining metadata elements to be identified.

Overall this chapter illustrated how an assisted approach to scientific workflow composition can enable a user to successfully compose a variety of common workflow scenarios, how this approach can effectively reduce a number of challenges presented by existing manual composition approaches and finally how maintaining a structured, dynamic knowledge base of component metadata can provide users with high quality suggestions for successfully composing workflow scenarios.

9 Discussion

The previous chapter has provided an evaluation of the computer-assisted composition approach described in this thesis, illustrating the capabilities the system has in supporting the composition of a number of representative workflow scenarios, and demonstrating the benefits this has in comparison to existing SWSs. The present chapter will provide a wider discussion of the relative capabilities of this approach.

Specifically, we compare the approach to providing suggestion based assistance presented in this thesis against similar systems developed for the field of web service composition. This is achieved by comparing the suggestions which would be generated by each system during composition of the scenarios introduced in Chapter 8 and evaluating the benefit provided by each approach.

In addition this chapter will consider the scalability of the approach presented, seeking to predict its ability to provide useful support when an increasing number of components are available within the system, as well as how well it can assist in the composition of workflows featuring a more complex structure. Finally a brief synopsis of the feedback from user testing performed to date is provided, illustrating the views of an example end user from the field of Bioinformatics who has had experience interacting with this approach to workflow composition.

9.1 Capability of Approach

The evaluation in Chapter 8 has demonstrated that the approach to composition presented in this thesis is valid in that the process of providing the user with abstract components with which to outline their goals, and providing them with suggestions regarding which components to insert and connect in order to achieve these goals, can result in the creation of workflows which successfully meet their needs. In this section we will look in greater detail at the relative benefits and weaknesses of this approach that the evaluation in Chapter 8 has revealed.

9.1.1 Can the Scenarios be Composed Fully?

Section 8.2.2, along with the material presented in Appendix B, provided descriptions of the process a user would follow when using the suggestion based composition approach to compose two of the workflow scenarios described in 8.2.1. The first walk-through for Scenario B, reading an image file and performing basic manipulation upon it, illustrated how the basic steps involved in creating a workflow composition - identifying required components, correctly ordering those components, and specifying the connections between components - can be achieved using a suggestion based approach. Additionally the scenario illustrated how this approach can reduce the complexity of achieving these steps by providing the user with a targeted, reduced number of choices, selecting from a limited number of suggestions rather than exploring a complete list of available components. This also demonstrated how the use of suggestions as the means to progress the composition also affords a level of guidance beyond simply which component to include within the workflow, providing suggestions both on which components to use and on how to connect them.

Additionally this walkthrough illustrated how the use of various elements of metadata can assist in suggesting only those components which are of genuine benefit to the user. In this instance there were many other components which could have been suggested for inclusion within this scenario based on their use of ports that deal with image files. For example the GARPPrediction component encountered in Scenario C has an output port that produces image data, however as this component is not from the Image Processing domain it does not distract from those, more suitable, suggestions which are from this domain.

A further advantage is also illustrated when composition of the workflow is only partially complete. Upon the inclusion of the Rotate component the user is also prompted with the choice to introduce further Image Processing components. In this way the approach can assist in extending the reach of a user's initial composition, incorporating further functionality that they may not have initially identified.

However, the walk-through of Scenario B also illustrates a potential negative aspect of this approach; the user is required to manually identify situations where they must

specify parameters for components within the composition. Providing no mechanism to alert the user that components may require properties to be configured could result in the user being unaware of why their structurally complete workflow does not execute correctly. This situation could be improved by implementing an additional category of suggestion, Property Suggestions, which prompts the user with components whose properties are incomplete and potentially provide suggestions on how to satisfy these properties.

The second walk-through presented for Scenario C, modelling species distribution, illustrates how relating each component port to the PortDataObject metadata is of benefit in assisting this identification of components. As the GARP components required for this scenario each have ports with the common BasicType of “String”, this means that if the system were reliant on this factor to identify compatibility the user would be presented with a large number of suggestions, increasing the difficulty in identifying the correct step to take. As the PortDataObject metadata is more specific to the data involved in connections with these components’ ports the system is able to refine the list of suggestions more accurately to those of genuine interest to the user. As detailed in Section 8.2.4, each individual element of metadata retained by the system can be of benefit in improving the effectiveness of the suggestions provided by the system. However, a negative aspect, illustrated through Scenario C, is the reduced effectiveness of suggestions when a user has yet to develop a detailed history of past interactions with the system. If a user’s connection history is not present, or is not yet of sufficient detail the system can provide unhelpful lists of suggestions. In the case of this scenario the connection between StringConstant and GARPPresampleLayers and GARPAlgorithm would only be an obvious choice if the user has previously implemented this connection a number of times. If this connection had not been previously specified the system would only be able to use the static metadata regarding these components to provide suggestions, resulting in any components compatible with the BasicType “String” being identified as a possible connection.

9.2 Comparison with Existing Suggestion Approaches

As described in Chapter 2 there is an increasing desire for the functionality of multiple web services to be combined into “value-added” services, with several approaches to either automating or assisting this process being developed. In order to further establish

the benefit of our proposed techniques, this section makes comparison with two of these existing composition systems previously described in Chapter 2, the CAT system [12] and the approach described by Sirin *et al.* [34]. These systems have been chosen as they share similar mechanisms to those used in our proposed approach. Following the discussion of these approaches we provide a summary of the respective limitations and benefits which they present in relation to our own system.

9.2.1 Sirin *et al.*

An approach to assisted service composition proposed by Sirin *et al.* [34] was introduced in Chapter 2. Their solution involves the creation of semantic "service profiles" for each web service which is to be composable using their approach. These service profiles outline what the service does by providing details such as the input and output types of the service, and any preconditions that service has in order to execute. In addition their approach makes use of an ontology to impose a hierarchy on these service profiles, relating services to one another based on how closely related their service profiles are.

The composition of services is achieved using an inference engine which has the capability to inspect the ontology of service profiles to determine whether the output of any service is suitable for connecting to the input of any service which the user has already included in their composition. Similar to the approach described in this thesis the inference engine is able to impose some ranking on the suggestions it provides. By using its knowledge of input and output types the system decides whether two services input and output types are an exact match or a generic match, with exact matches being ranked highest. In addition non-functional attributes which are stored in the systems ontology can also be used to filter the list of provided suggestions; however this is a manual process which the user must perform.

9.2.2 CAT

The CAT [12] system provides a suggestive composition approach targeted at the composition of web services. The system works on a similar principle to the assisted composition approach described in this thesis, both systems enable users to achieve

their composition goals by providing a series of suggestions for progress based on the current state of the composition.

The CAT suggestion system uses a concept of error identification and correction, the current state of a composition is inspected against a set of pre-defined ideals for a complete composition. These describe that each element within a composition should be executable, that each element should have all of its input and output requirements satisfied, each element should be compatible with the elements it is connected to, and that the composition must have a clear beginning and an end – it must not be an endless cycle. Each of these ideals has one or more related “actions” designed to satisfy the requirements of that ideal. An incomplete workflow is scanned by the system for any occurrences where these ideals are not satisfied – when these are identified the system suggests one or more of its available actions in order to rectify the problem.

In this way the CAT approach seeks to allow the user to compose what is initially a generic web service composition, containing some of the tasks they wish to perform, and subsequently refine this composition through implementing the actions that CAT suggests based on the errors the system is able to locate within the current composition. In order to correctly identify errors within the web service composition, and to provide effective actions to resolve these, the CAT system maintains a task ontology which contains knowledge relating to the type of information produced and consumed by each web service available. Based on a composition containing services with incompatible types, or services with unsatisfied inputs or outputs, CAT is able to refer to the task ontology to locate services which can either translate the output type of one service into a compatible type to input into another service, or those which are compatible with those services currently unsatisfied.

9.2.3 Limitations of Existing Approaches

As described previously the approach to assisted composition detailed in this thesis utilises an ontology to record metadata relating to components, relating each component to both static factors such as the domain in which it is used, the type of data shared by its ports, and the high level abstract concept that it implements, as well as recording dynamic information such as the number of times a user has connected components within their history of compositions.

In contrast both the CAT system and the approach taken by Sirin *et al.* maintain only a limited ontology which relates each component to both the task it performs and the data type(s) of its input and output ports. As discussed in Section 8.2.4 relying solely on port types as the means to provide suggestions for suitable components to introduce into a workflow results in a reduction in the quality and effectiveness of those suggestions provided, particularly when composing scenarios that involve components with common or generic port data types. Utilising data types to determine compatibility ensures that no compatible choice will be overlooked, but also results in situations where the number of suggestions becomes overly large, increasing the difficulty the user encounters when inspecting the available suggestions for the ideal step to implement. Whilst the approach taken by Sirin *et al.* provides a filtering mechanism to attempt to reduce the size of suggestions provided, this is a manual process which must be undertaken by the user themselves and therefore returns to the problem of requiring that the user has sufficient knowledge of both the task they wish to perform, and the properties of the available components.

Secondly the knowledge maintained within these ontologies is entirely static, the continued interactions a user has with the system will have no effect on the suggestions that they are provided with. By not allowing the provision of suggestions to adapt to the manner in which a user interacts with the system the quality of suggestions cannot improve over time. For individual users this means that in situations where they repeatedly implement steps that are low in the list of suggestions provided by the system they must continually search through that list for the implementation they know to be correct. For a domain of related users this means that one user's interactions cannot be utilised to improve the suggestions provided to others – where the dynamic approach presented in this thesis enables experienced users interactions to improve the assistance provided to new users, a static knowledge base means that those with limited experience must overcome the same challenges as those who have already identified the correct steps to take.

9.2.4 Benefits of Existing Approaches

Whilst the knowledge that is represented within the ontologies presented by CAT and SIrin *et al.* has a number of limitations in relation to the suggestions it can provide, an

advantage the CAT approach provides comes from the mechanism which is used to generate those suggestions. By taking an approach that compares an on-going composition against a set of ideals such as each component's "Uniqueness" and "Consistency" the CAT approach is able to identify any element of a composition that does not match these ideals. Whilst in a SWS context this would have the disadvantage of interpreting unconnected ports as a problem that means the composition is incomplete, when there are numerous components that do not require all of their ports to be satisfied in order to function successfully, this approach has the advantage of always making the user aware of areas in which their on-going composition may be deficient.

In Section 8.2.2 we saw a number of situations where the approach to workflow composition described in this thesis was able to deliver a structurally complete composition for a scenario, but failed to inform the user of component properties that must be defined before the workflow could be successfully executed. By implementing an approach similar to that of the CAT system, which considered this definition of component properties as a requirement for a complete composition, it would be possible to highlight areas where component properties were in need of customisation before the composition could be considered complete.

Similarly whilst the filtering system of Sirin *et al.* requires the user to provide additional information about the type of services they wish to use in their composition, and therefore is of limited benefit to a user who is engaged in "discovering" the services they require for their composition, this is a powerful tool for more advanced users who already have knowledge of the type of services they require. As the numbers of components available within a system grows such a filtering mechanism may become a necessity in order to ensure that the user has some capability to quickly reduce the list of suggestions provided to only those they may be interested in.

9.2.5 Summary

This section has provided an overview of a number of existing approaches to supporting the composition of web services. The relative benefits and limitations of each system were discussed in relation to the approach proposed in this thesis, with the conclusion that whilst each of these systems provide mechanisms to support users in

creating their workflows or composite services, the limited knowledge that they retain about both the available services and the users performing the composition results in the quality of the assistance they provide being limited. By incorporating metadata relating to components' domains of use, provider, past history of connections, and further information relating to the data produced or consumed by their ports, the approach described in this thesis is able to identify where components are not just compatible but also desirable to include within a composition, and is able to reduce the number of suggestions that a user is provided with, thus reducing the challenge in identifying which suggestions to implement.

The CAT approach was shown to have benefits in the ability to further identify areas where a developing composition was in need of attention, beyond those identified by the approach to assistance discussed in this thesis. In the context of the scenarios utilised for composition in this chapter this manifested itself in the approach being unable to prompt the user when workflow components possessed internal parameters that needed to be defined before the composition would execute correctly. By employing the CAT approach of comparing these compositions against requirements for a complete, executable workflow such deficiencies could be identified and addressed. The approach presented by Sirin *et al.* was also shown to have benefits in its ability to allow users to filter the list of suggestions provided based on the non-functional properties of the available services.

Furthermore the approaches described in this section lack the dynamic aspects included within the approach presented in this thesis. By incorporating knowledge relating to a user's past interactions with the system the quality of suggestions can improve over time, as well as enabling the connection history of an expert user to improve the quality of suggestions provided to a novice.

9.3 Scalability of Approach

Scalability of the computer-assisted composition approach presented is an important consideration as there are a number of factors which could potentially increase in complexity after continued use. The main areas where scalability must be examined are: the effect of including an increasing number of workflow components and metadata elements within the framework, the support the approach provides for creating

increasingly complex workflow compositions, both in terms of numerical complexity (including many components) and structural complexity (including components connected in a complex manner such as loops, branches and split connections), and finally the effect an increasingly populated connection history has on the suggestions the approach can provide.

9.3.1 Increasing Components within Framework

Taking the initial mathematical example scenario from Section 8.2.2, calculating and displaying the remainder of a division, the proof of concept established this as a relatively straight-forward and simple scenario to compose using our method.

However re-calculating the suggestions which would be provided by the system during composition illustrates how the ease of completing this scenario could be reduced, if incorporating a metadata framework including far more components than are currently supported. As an example Table 9-1 lists the first addition suggestions the user is presented with following inclusion of the Remainder component.

Addition Suggestions
add a Constant component (matches with input to Remainder)
add a Display component (matches with output from Remainder)

Table 9-1 Initial Addition Suggestions for Scenario A

These suggestions are provided based on their compatibility with the Remainder component included within the composition. The decision for suggesting these components is based primarily on the compatibility of their port types with those of the Remainder component, in this instance each of the components includes ports which involve the sending or receiving of a Double numerical data type.

If the number of components within the framework which included this data type within their input or output ports were greatly increased then there would be a corresponding increase in the number of components which the system would identify as being compatible with the Remainder component.

In cases such as this scenario a previously trivial step of identifying which suggestion to implement from only a few choices, could become an arduous task of inspecting a long list of suggestions for the right one to implement. This is effectively making the suggestion approach equivalent to the existing manual composition approach where the user is left to navigate through a large list of available components with limited information to identify which is required.

This negative impact of an increased number of components defined within the framework is however mitigated by a number of other elements of the approach. Primarily the inclusion of elements within the reasoning to assess the desirability of connecting or inserting components, alongside simply identifying whether components are “type compatible”, is designed to reduce the impact of a large number of available components. By promoting components within the list of suggestions based on the similarity of their Provider, Project, and Domain metadata elements the system attempts to highlight those suggestions which are more desirable for the user to implement.

This metadata is designed to support the provision of suggestions in relatively well defined user-domains, where a user is utilising a set of components which are specific to their project or domain. In cases such as this the system can identify those compatible components which do not belong to the user’s domain or project, and rank these below components which do belong to the user’s domain and project, reducing the challenge for the user in locating the components they require.

However, returning to Scenario A, the trivial nature of the components utilised within this composition means there is limited benefit of using this desirability metadata in ranking suggestions. Components such as Constant and Display which are used within this scenario have no defined Domain or Project metadata as they are used across a number of different areas. As a result the system would be unable to utilise this information to remove undesirable components from the list of available suggestions, resulting in any compatible components being suggested.

Furthermore instances where one component does have a defined Domain, but the components which should be connected to it do not, or belong to another Domain,

would result in the required suggestions being ranked lower than desired. For example Scenario A included the components Remainder, Constant and Display. Remainder is of the Mathematics domain whilst both Constant and Display are from different domains, this would mean that whilst a user would desire these components to be suggested for connection with Remainder, this would not be promoted by the system. However, the GUI includes the option to customise the filters which affect the ranking of suggestions so the user does have the option to remove ranking by domain from the list of suggestions provided.

Beyond utilising information relating to a component's domain, project or provider to mitigate the effect of a large number of compatible components being suggested, an individual user's history of interaction with the system is also incorporated within the ranking process to improve the resulting suggestions.

In our example once a user has inserted the Constant component into the workflow the system identifies all components which are compatible with its output type of double. If all the components available within the base Kepler system are represented within the component ontology then this would result in 144 suggestions. As stated previously, Constant has no defined project or domain, as it can be used for many purposes, and so this information does not help reduce this list of suggestions. The Provider metadata for the Constant component, "Ptolemy", will be used to reduce this list of suggestions but still leaves a large number for the user to inspect manually. However, if a user has previously utilised the assisted approach to composition the system can inspect their history of interactions and identify those components within the suggestions which the user has previously connected. By promoting those suggestions which the user has previously implemented over those which have never been utilised the system can potentially highlight those components a user is more interested in for their composition. In this way a user, working with the restricted number of components that are relevant to their individual domain, user's will develop a history of connections which can increasingly be utilised to improve the quality of their future suggestions, even when a large number of "logically compatible" components are available.

There are limitations in utilising a history of interactions with the system to improve suggestions. Firstly it takes time to develop this history, before the system is able to

have recorded the user implementing a useful number of suggestions there is no way in which this dynamic information can be used to restrict the number of redundant components within list of available suggestions. Until such a history has been defined the system must rely on utilising the static metadata in order to attempt to highlight the best suggestions for a user to implement. Additionally using the history of connections made by a user to improve the suggestions may have a detrimental effect when a user is working with a set of components they have used previously but wishes to approach a problem in a different manner.

9.3.2 Composing Complex Scenarios

The scope for complexity within the workflows that can be composed using existing SWSs is considerable, as there are many ways that a user can choose to sequence and connect the large number of components, ports and data types available. These systems also provide the user with the option to introduce functionality such as loops and branching into their composition to further increase this complexity.

A limitation of the assisted composition approach presented in this thesis is the manner in which suggestions are identified for inclusion. As described in Chapter 5 the system generates suggestions based on the knowledge about each component in isolation, and is only designed to accept a single connection into or out of each component's ports. Given a composition with components A, B and C, each with one input and output, the system would be able to identify where the user could connect these sequentially, A-B-C, but would be unable to suggest more complex structures such as introducing a loop, or passing the data from component A's single output port to the input of both components B and C. Figure 9-1 represents examples of such composition structures.

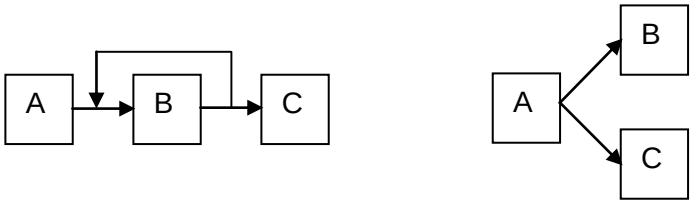


Figure 9-1 Advanced Workflow Structures

This limitation means that structurally complex compositions are difficult to compose within the confines of the assisted approach, requiring the user to implement structures such as loops and branches by returning to the underlying SWS itself. This limitation could potentially be mitigated by introducing new “structural” abstract components which the user could configure in order to achieve functionality such as looping and branching within their compositions. For example, a “loop” component could contain a “sub-workflow” representing the tasks the user wishes to repeat.

9.3.3 Summary

Whilst the approach to providing composition assistance described in this thesis has a number of advantages over existing, manual composition approaches, and similarly is able to provide higher quality suggestions than other suggestions based composition approaches, there are circumstances where it is not so effective.

As the suggestions provided by the system are based on knowledge it maintains about available components increasing the number of available components results in the system having a larger knowledge base to inspect for suggestions. In situations containing components which only have limited metadata available, or which have very “generic” metadata, this can result in a larger number of suggestions being presented to the user, reducing the benefit that this approach is designed to provide. However, the approach of including further metadata such as components' past connections history, and providing facilities within the UI for users to determine which elements of metadata are included in the provision of suggestions can reduce the impact of a large number of available components.

An important deficiency in this approach is its inability to assist users in the composition of structurally complex workflows. The suggestions the system provides only enable the user to connect components sequentially, limiting port connections to 1:1 relationships and preventing the inclusion of structures such as loops within compositions. Whilst some complex compositions such as connecting one component output to multiple component inputs can be circumvented by duplicating the component providing the output, this increases the number of components included in a workflow and requires the user to be aware of how to achieve this workaround.

The approach of providing the suggestion system as an extension to an existing SWS means that in circumstances such as this a user is able to save a workflow partially completed in the extension and load this back into the existing SWS, enabling the definition of more complex structures to be completed.

9.4 User Feedback

As mentioned in Chapter 1 the work on this thesis was undertaken with the support of a Microsoft Research Europe studentship; as part of this some user feedback was obtained from the Microsoft contact Dr Rich Williams. This involved Dr Williams using the computer assisted composition system to attempt to compose Scenario C as introduced in Chapter 8 and providing responses to a questionnaire based on this experience with the system. The completed questionnaire is provided in Appendix G.

The primary conclusions drawn from the user feedback were that the assisted approach is capable of correctly composing workflow scenarios and that the use of suggestions to guide users through the space of possible components and their sequencing was the most successful aspect. However the feedback also identified that the approach may need to be made more sophisticated in order to remain successful when dealing with more components and more complex composition scenarios.

9.5 Conclusion

This chapter has provided a discussion surrounding the benefits and limitations presented by the approach to assisted composition described in this thesis. The manner in which the scenarios presented in Chapter 8 demonstrate this approach's ability to overcome some of the identified deficiencies in existing SWSs has been discussed in greater detail, as well as further illustrating the areas where this approach is still lacking.

In addition this chapter has discussed the merits of this approach in comparison to existing suggestive composition systems available in the field of web service composition, such as CAT and the system proposed by Sirin *et al.*

This chapter also provided an overview of the scalability of this approach, illustrating situations where extending the scope of the components supported by the approach and the complexity of the scenarios to compose caused the approach to perform less effectively. Finally a brief overview of the response from an end user of the system was provided, establishing the extent to which they understood the assisted composition approach to be of benefit to the type of SWS composition they routinely perform.

10 Conclusions and Future Work

This chapter provides a summary of the results presented in this thesis, evaluating the extent to which the aims and objectives outlined in Chapter 1 have been achieved, and therefore how the truth of the hypothesis has been demonstrated. In addition we propose a number of areas where future work could be undertaken to improve on what has been achieved.

10.1 Overview

As mentioned in Chapter 1, current scientific workflow composition systems place large demands on the user in terms of the amount of knowledge that they must possess before they are able to successfully compose workflows. This includes knowing the exact tasks they wish to perform within the SWS environment, being aware of which specific components are required to achieve these tasks, and having knowledge of how to configure and sequence these components in order to provide the correct output.

This thesis has presented a new suggestion-based approach to workflow composition to test the hypothesis that knowledge about users and the workflow components which are available to them can be used to provide a computer-assisted approach that can reduce the challenges presented by existing SWSs. This approach makes use of a number of elements of metadata about both the components available and the users themselves in order to assist the composition process.

Chapter 1 introduced a number of aims and objectives for this thesis. The overall aims were to:

- Investigate how resource metadata can be used to generate suggestions to assist users in creating workflows.
- Explore how a user interface could be developed to present this assistance to users.
- Determine how such assistance could be provided across multiple existing SWSs.

In addition the following objectives were identified in order to achieve these aims, and subsequent chapters in the thesis relate to the achievement of these objectives:

1. Develop a framework for representing knowledge about available resources.
2. Populate the framework with knowledge relating to resources to demonstrate how such information can be of benefit when composing workflows.
3. Create algorithms to generate workflow composition suggestions from metadata
4. Develop an API layer to enable the interface to sit on top of multiple existing workflow systems.
5. Provide a user interface to enable users to utilise assistance during workflow composition
6. Evaluate the proposed framework with respect to the hypothesis and in relation to other published work

The hypothesis stated in Chapter 1 has been tested by (i) constructing a metadata framework that could capture suitably rich semantic information about workflow components, (ii) developing algorithms which can inspect the current state of a users composition and then utilise this metadata to generate suggestions for how to progress the composition, (iii) developing an API framework to enable the metadata ontology and suggestion algorithms to be used in conjunction with existing SWSs, and (iv) implementing a prototype system which has been successfully used to complete a number of composition scenarios from the Kepler SWS.

10.2 Objective Completion

The following sections draw together the main contributions of this thesis with respect to the objectives they have addressed.

10.2.1 Component Metadata Framework (Objectives 1 & 2)

In order to support the provision of suggestions which can assist users during workflow composition a framework is required which can store sufficiently detailed metadata relating to available workflow components. The metadata ontology presented in this thesis stores relevant information about both the properties of available components and the high level goals that those components perform. By standardising the

information which is stored about each component it becomes possible for the system to use this information to reliably inspect this data to assess component compatibility. Furthermore by representing the tasks performed by each component within a hierarchy it is possible to present the user with high level abstract components which can be directly mapped back to the available components which implement this activity. This ontology has also been designed to record information regarding the user's interactions with the system; making use of this information to improve the assistance which can be provided as the system becomes more familiar with a user's working habits.

10.2.2 Suggestion Algorithms (Objective 3)

A number of algorithms have been developed which make use of the information retained in the component ontology in order to present users with suggestions for how to complete their workflow composition. The information within the ontology is used to identify components which are compatible with the user's composition, but also to identify which suggestions are most desirable for the user to implement. In addition by including greater detail in the component ontology than is present in either the existing SWS or in other assisted composition approaches evaluated it is possible for these algorithms to rank the suggestions available to the user. In this way the system presents the user with a smaller number of suggestions, with the most useful suggestions being highlighted.

10.2.3 Intermediate API (Objective 4)

As there are a number of existing SWSs which are commonly used within the scientific community and which already provide suitably rich environments for sequencing and executing sets of components it is sensible to ensure that an approach to providing composition assistance can be used in conjunction with these existing systems. This thesis has presented an API which can sit between an existing SWS and additional software that provides extensions to the underlying SWS, providing the extension with access to the required functionality of the SWS to allow for workflows to be composed and executed. The generic nature of this API means that it can support a number of possible extensions, not just the approach to computer-assisted composition presented in this thesis.

10.2.4 SWS Extension and Performance Study (Objectives 5 & 6)

The proposed metadata ontology has been implemented using the OWL ontology language. This allows the relevant properties and relationships to be defined to record the required component metadata and to support the queries required by the suggestions algorithms. A basic UI has been developed to enable a user to compose workflows using abstract components and the suggestions provided by the system. A number of means have been used to determine the effectiveness of the prototype system when implementing a number of composition scenarios. These include rating the quality of suggestions provided by the system, illustrating how the approach can be used to avoid difficulties that are encountered during composition within the existing Kepler SWS, and also evaluating how the approach compares with a number of approaches used to assist in the composition of web services.

10.2.5 Summary

Overall the work undertaken has been able to achieve the aims and objectives which were established in Chapter 1. An approach to recording useful metadata about components was defined, implemented, and demonstrated by populating it with data for components from existing scientific workflow systems (Objectives 1 & 2). Mechanisms were then developed through which this metadata could be inspected in order to generate suggestions which could be presented to users to assist during the workflow composition process (Objective 3). By completing these objectives the primary aim of this thesis has been achieved; to identify whether metadata about workflow components could be used to assist users during the workflow composition process.

In order to demonstrate this practically a prototype user interface has been implemented (Objective 5), and an intermediate translation layer and API have been developed to enable this interface to be utilised with a number of existing scientific workflow systems (Objective 4). A suitable approach to evaluating this system has then been established through analysis of the methods used to evaluate both similar assisting systems and existing workflow systems, and this approach has been exercised against the systems developed (Objective 6). These steps have demonstrated the remaining aims of the work; to explore how the assistance provided

by knowledge based suggestions could be used in presented to a user and utilised during composition with a number of existing SWSs.

In conclusion these achievements and the work in this thesis in total have demonstrated that the original hypothesis was a sound statement: a detailed review of the state of the art within scientific workflows and related areas has established that genuine challenges exist for users attempting to create successful workflow compositions, and that these challenges have yet to be fully overcome by the work conducted in the field to date. By developing the approach to recording component metadata and utilising this to provide composition suggestions to users within a prototype user interface, it has further been demonstrated that knowledge about workflow components and their usage can be utilised to assist users in successfully completing the composition of genuine workflow scenarios, and that this approach has benefits over those approaches which currently exist.

10.3 Future Work

There are a number of areas where the work presented in this thesis could be extended, these include expanding the recording of a user's composition history beyond simply recording when two components are connected, incorporating a concept of defining the "ideal" input to satisfy a component, maintaining a hierarchy of user domains to enable suggestions from similar domains to benefit one another, a mechanism through which users could correct errors in, or otherwise improve, the information stored in the metadata ontology, and a change in the approach to representing component metadata by storing the information about components from each SWS within a single ontology.

10.3.1 Expanded User History

The current system keeps a record of each connection which a user makes between two components, storing information regarding the components and ports involved, as well as a count of the number of times which this connection has been made. As has been demonstrated in Chapter 8 this approach can assist in identifying when two components are a good option for the user to include or connect in their composition

but it is limited as it is only concerned with interactions between two individual components.

As demonstrated in the two bioinformatics scenarios described in Chapter 8 it is to be expected that users may routinely connect sequences of more than two components across multiple workflow compositions, in this case the three “GARP” components. If the system were to be able to identify and record such patterns then more assistance could be afforded to the user, suggesting the inclusion of the whole sequence in one step, rather than expecting the user to follow each individual step at a time.

Additionally the system could be expanded further to monitor the wider usage of components within a workflow composition, for example trends in terms of which components are present in a workflow - it may be the case that for a particular user their compositions which include components A and B, always also include component C, or trends in the connections between those components – if X, Y and Z are included in a composition then Y and Z are always connected, if X is not present then this is not the case. Such information could assist the system in providing the user with more accurate suggestions for steps to implement.

10.3.2 Ideal Inputs

At present the metadata stored regarding components is static; the information which identifies whether two components can be logically connected is reliant on the PortType and PortDataObject elements of metadata. A further refinement could be made to define the “ideal” input that satisfies a component, this would essentially expand upon the existing PortDataObject hierarchy to introduce even more specific refinements.

For example the current PortDataObject hierarchy includes the following child elements:

- PortDataObject
    - FileDataObject
        - ImageDataObject
            - JPEGImageDataObject
            - GIFImageDataObject

This could be extended to include very specific detail about the image which a component produces or consumes, for example the image size, colour depth or even file size. Recording this more specific information may help to ensure that suggestions provided to the user are more likely to provide a positive outcome when executed.

10.3.3 User Domain Hierarchy

At present the history or interaction which a user builds up as they make use of the system is only able to provide assistance to that individual user. If the system were able to record the domain(s) within which a user is working, as well as a hierarchy describing how closely related two domains were then there is potential for the history which one user is developing to be of assistance to another user in a related domain. This would be of assistance in the situation where a number of components are used across multiple domains, or more specifically are used in the same manner across multiple domains.

Care would need to be taken to ensure that the domain hierarchy were carefully constructed to ensure that this did not result in a decrease in the accuracy or helpfulness of the suggestions provided by the system. It may be sensibly for such a mechanism to be optional within the user interface to enable users to avoid this possibility. A similar approach could be taken with regards to defining relationships between specific projects as well as domains, where interactions made by users from one project could be used to help those from a similar or related project.

10.3.4 User Curated Metadata

As described in Section 4.5 the current implementation of the component metadata ontology is populated by extracting information about components from the various sources within existing SWSs such as component names, locations within the component listing and component documentation.

Given the large and growing number of components and the possibility that the specific usage or features of any one component could be misinterpreted from these existing sources of information, it is not feasible that the same approach could be taken long

term to populate the metadata ontology, or if it were then it would result in inaccurate information being recorded.

A more useful approach may be to implement a mechanism within the system to enable users to extend and improve the accuracy of the information within the ontology. This could be achieved by providing a direct interface through which users can manipulate the ontology, or by enabling users to mark certain suggestions provided by the system as unhelpful or inaccurate, allowing the system to learn which suggestions are most beneficial. By allowing the ontology to evolve in this way the quality of suggestions would improve over time as well as enabling the system to remain relevant as newer components are made available.

This could be improved further still by taking advantage of the growing repositories of completed workflow compositions which are being made available on the internet. Projects such as myExperiment [10] and Workflow4ever [81] are making available a vast selection of completed workflow compositions across a number of domains. By taking advantage of this information it could be possible to "pre-prime" aspects of the metadata ontology, such as the history of usage of common components, thus offering benefit in situations where a component is commonly used within a domain but a particular user has yet to interact with it.

10.3.5 Unified Component Metadata Ontology

At present the implementation of the prototype system made use of an individual ontology per SWS; this approach was taken to simplify the implementation and to enable the ontology to be populated more quickly to enable testing of the approach. However it could be possible to represent the components from each system within a single ontology, grouping components which achieved the same task within each at the same level within the task hierarchy. This approach would have several benefits, potentially enabling for compositions from one SWS to be easily translated to another by using the task hierarchy to identify the required components, as well as enabling the history of user interaction within one SWS to have benefit when composing workflows with another.

10.3.6 Improved Component Parameter Metadata

As discussed in Chapter 8 a limitation with the current system is the inability to identify when the user may need to perform some manual configuration of a components parameters in order for their workflow composition to execute correctly, for example configuring a file reader component with the location of the file to be read. This situation could be improved if the metadata ontology were extended to categorise each component's parameters based on whether they are *optional* or *mandatory*. If the composition includes a component with mandatory parameters then the suggestions section of the UI (or some new mechanism) could highlight this fact to the user.

Identification of which parameters are optional and which are mandatory may prove a difficult task for all components, but could be information that is provided by the component provider themselves when producing their components. This idea could potentially be expanded further by offering similar metadata regarding component parameters as is currently available for ports, describing the type of data which is required to satisfy a parameter, thus enabling the system to not just highlight when a parameter needs to be satisfied but also to potentially suggest what it needs to be satisfied with.

10.3.7 Workflow System Interoperability

An area of research which is gaining interest is that of workflow interoperability, allowing workflows and components from one SWS to interact with those of another. It is conceivable that with further development the API which has been developed in this work could be expanded to facilitate such interoperability. The API at present provides a mechanism to provide a single interface through which common functionality can be achieved within a number of existing SWSs, by introducing a translation element into this API it could be possible to enable the computer-assisted composition system which has been developed on top of this API to interact with multiple SWSs simultaneously, creating workflows which makes use of the resources available across each system.

Publications

Elements of the work described in this thesis have been previously contained in the following publications:

- McIver, R. P., Jones, A. C., & White, R. J. (2009, August). A Framework for Supporting the Composition of Biodiversity Informatics Resources. In 20th International Workshop on Database and Expert Systems Applications (DEXA'09), pp. 350-354, IEEE.

- McIver, R. P., Jones, A. C., and White, R. J. (2008). Workflow Systems for Biodiversity Researchers: Existing Problems and Potential Solutions. *Proceedings of Biodiversity Informatics: challenges in modelling and managing biodiversity knowledge*. http://biodiversity.cs.cf.ac.uk/bncod/proceedings2008.html

References

1. Lane, M. A., Edwards, J. L., & Nielsen, E. (2000, September). Biodiversity informatics: the challenge of rapid development, large databases, and complex data (keynote). In Proceedings of the 26th International Conference on Very Large Data Bases (pp. 729-732). Morgan Kaufmann Publishers Inc..

2. Bisby, F. A. (2000). The quiet revolution: biodiversity informatics and the internet. Science, 289(5488), 2309-2312.

3. Bowers, S., Ludascher, B., Ngu, A. H., & Critchlow, T. (2006). Enabling scientificworkflow reuse through structured composition of dataflow and control-flow. In Data Engineering Workshops, 2006. Proceedings. 22nd International Conference on (pp. 70-70). IEEE.

4. Oinn, T., Greenwood, M., Addis, M., Alpdemir, M. N., Ferris, J., Glover, K., ... & Wroe, C. (2006). Taverna: lessons in creating a workflow environment for the life sciences. Concurrency and Computation: Practice and Experience, 18(10), 1067-1100.

5. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., ... & Li, P. (2004). Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics, 20(17), 3045-3054.

6. Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., & Mock, S. (2004, June). Kepler: an extensible system for design and execution of scientific workflows. In Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on (pp. 423-424). IEEE.

7. Gil, Y., Deelman, E., Ellisman, M., Fahringer, T., Fox, G., Gannon, D., ... & Myers, J. (2007). Examining the challenges of scientific workflows. Ieee computer, 40(12), 26-34.

8. Churches, D., Gombas, G., Harrison, A., Maassen, J., Robinson, C., Shields, M., ... & Wang, I. (2006). Programming scientific and distributed workflow with Triana services. Concurrency and Computation: Practice and Experience, 18(10), 1021-1037.

9. Eker, J., Janneck, J. W., Lee, E. A., Liu, J., Liu, X., Ludvig, J., ... & Xiong, Y. (2003). Taming heterogeneity-the Ptolemy approach. Proceedings of the IEEE, 91(1), 127-144.

10. Goble, C. A., & De Roure, D. C. (2007, June). myExperiment: social networking for workflow-using e-scientists. In Proceedings of the 2nd workshop on Workflows in support of large-scale science (pp. 1-2). ACM.

11. Gaaloul, K., Charoy, F., & Godart, C. (2006). Cooperative processes for scientific workflows. In Computational Science–ICCS 2006 (pp. 976-979). Springer Berlin Heidelberg.

12. Kim, J., & Gil, Y. (2004, March). Towards interactive composition of semantic web services. In in Proceedings of the AAAI Spring Symposium on Semantic Web Services, 22nd-24th March.

13. Shawn, C. B., Bowers, S., Jones, M. B., Ludäscher, B., Schildhauer, M., & Tao, J. (2005). Incorporating semantics in scientific workflow authoring. In In Proceedings of the 17th International Conference on Scientific and Statistical Database Management (SSDBM'05.

14. Kepler: An Extensible System for Scientific Workflows, http://kepler.ecoinformatics.org

15. Ptolemy II, http://ptolemy.eecs.berkeley.edu/ptolemyII/

16. Triana, http://www.trianacode.org/

17. Taylor, I., & Schutz, B. (1998). Triana-A quicklook data analysis system for gravitational wave detectors. In Second Workshop on Gravitational Wave Data Analysis (pp. 229-237).

18. Taverna Workflow Management System, http://www.taverna.org.uk/

19. myGrid, http://www.mygrid.org.uk/

20. W3C Web Services Glossary, http://www.w3.org/TR/ws-gloss/

21. Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V., & Shan, M. C. (2000, January). Adaptive and dynamic service composition in eFlow. In Advanced Information Systems Engineering (pp. 13-31). Springer Berlin Heidelberg.

22. Rao, J., & Su, X. (2005). A survey of automated web service composition methods. In Semantic Web Services and Web Process Composition (pp. 43-54). Springer Berlin Heidelberg.

23. Benatallah, B., Dumas, M., Sheng, Q. Z., & Ngu, A. H. (2002). Declarative composition and peer-to-peer provisioning of dynamic web services. In Data Engineering, 2002. Proceedings. 18th International Conference on (pp. 297-308). IEEE.

24. Medjahed, B., Bouguettaya, A., & Elmagarmid, A. K. (2003). Composing web services on the semantic web. The VLDB Journal, 12(4), 333-351.

25. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., ... & Sycara, K. (2004). OWL-S: Semantic markup for web services. W3C member submission, 22, 2007-04.

26. Beco, S., Cantalupo, B., Matskanis, N., & Surridge, M. (2006). Putting semantics in grid workflow management: the OWL-WS approach. DATAMAT SPA, University of Southampton IT Innovation Centre. Retrieved September, 15, 2012.

27. Casati, F., Ilnicki, S., Jin, L. J., Krishnamoorthy, V., & Shan, M. C. (2000). eFlow: a platform for developing and managing composite e-services. In Research Challenges, 2000. Proceedings. Academia/Industry Working Conference on (pp. 341-348). IEEE.

28. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., & Sheng, Q. Z. (2003, May). Quality driven web services composition. In Proceedings of the 12th international conference on World Wide Web (pp. 411-421). ACM.

29. Blythe, J., Deelman, E., & Gil, Y. (2004). Automatically composed workflows for grid environments. Intelligent Systems, IEEE, 19(4), 16-23.

30. Wu, D., Parsia, B., Sirin, E., Hendler, J., & Nau, D. (2003). Automating DAML-S web services composition using SHOP2 (pp. 195-210). Springer Berlin Heidelberg.

31. Nau, D., Munoz-Avila, H., Cao, Y., Lotem, A., & Mitchell, S. (2001, August). Total-order planning with partially ordered subtasks. In IJCAI (Vol. 1, pp. 425-430).

32. Wu, D., Sirin, E., Hendler, J., Nau, D., & Parsia, B. (2006). Automatic web services composition using shop2. Maryland univ college park dept of computer science.

33. McIlraith, S. A., Son, T. C., & Zeng, H. (2001). Semantic web services. IEEE intelligent systems, 16(2), 46-53.

34. Sirin, E., Hendler, J., & Parsia, B. (2003, April). Semi-automatic composition of web services using semantic descriptions. In 1st Workshop on Web Services: Modeling, Architecture and Infrastructure (pp. 17-24).

35. Harrison, A., Kelley, I., Mueller, K., Shields, M., & Taylor, I. (2007). Workflows hosted in portals. In Proceedings of the UK e-Science All Hands Meeting (pp. 32-39).

36. Zhao, Y., Raicu, I., & Foster, I. (2008, July). Scientific workflow systems for 21st century, new bottle or new wine?. In Services-Part I, 2008. IEEE Congress on (pp. 467-471). IEEE.

37. Dryer, D. C. (1997, January). Wizards, guides, and beyond: Rational and empirical methods for selecting optimal intelligent user interface agents. In Proceedings of the 2nd international conference on Intelligent user interfaces (pp. 265-268). ACM.

38. Horvitz, E. (1999, May). Principles of mixed-initiative user interfaces. In Proceedings of the SIGCHI conference on Human Factors in Computing Systems (pp. 159-166). ACM.

39. Kim, J., Gil, Y., & Spraragen, M. (2004). A knowledge-based approach to interactive workflow composition. In 14th International Conference on Automatic Planning and Scheduling (ICAPS 04).

40. Medjahed, B. (2004). Semantic web enabled composition of web services (Doctoral dissertation, Virginia Polytechnic Institute and State University).

41. Plock, C. (2008). Synthesizing executable programs from requirements (Doctoral dissertation, New York University).

42. Gallopoulos, E., Houstis, E., & Rice, J. R. (1994). Computer as thinker/doer: Problem-solving environments for computational science. Computational Science & Engineering, IEEE, 1(2), 11-23.

43. Beco, S., Cantalupo, B., Giammarino, L., Matskanis, N., & Surridge, M. (2005, July). OWL-WS: a workflow ontology for dynamic grid service composition. In e-Science and Grid Computing, 2005. First International Conference on (pp. 8-pp). IEEE.

44. Bubak, M., Gubała, T., Kapałka, M., Malawski, M., & Rycerz, K. (2005). Workflow composer and service registry for grid applications. Future Generation Computer Systems, 21(1), 79-86.

45. McIlraith, S., & Son, T. C. (2002). Adapting golog for composition of semantic web services. KR, 2, 482-493.

46. Yu, J., & Buyya, R. (2005). A taxonomy of scientific workflow systems for grid computing. ACM Sigmod Record, 34(3), 44-49.

47. van Nimwegen, C., van Oostendorp, H., & Schijf, H. (2004, June). Can more help be worse?: the over-assisting interface. In Proceedings of the conference on Dutch directions in HCI (p. 4). ACM.

48. Birnbaum, L., Horvitz, E., Kurlander, D., Lieberman, H., Marks, J., & Roth, S. (1997, January). Compelling intelligent user interfaces-how much AI?. In International Conference on Intelligent User Interfaces: Proceedings of the 2 nd international conference on Intelligent user interfaces (Vol. 6, No. 09, pp. 173-175).

49. McGuinness, D. L., & Van Harmelen, F. (2004). OWL web ontology language overview. W3C recommendation, 10(10), 2004.

50. Horridge, M., Knublauch, H., Rector, A., Stevens, R., & Wroe, C. (2004). A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools Edition 1.0. University of Manchester.

51. McBride, B. (2002). Jena: A semantic web toolkit. Internet Computing, IEEE, 6(6), 55-59.

52. Pérez, J., Arenas, M., & Gutierrez, C. (2006). Semantics and Complexity of SPARQL. In The Semantic Web-ISWC 2006 (pp. 30-43). Springer Berlin Heidelberg.

53. Weske, M. (2001, January). Formal foundation and conceptual design of dynamic adaptations in a workflow management system. In System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on (pp. 10-pp). IEEE.

54. Lämmermann, S. (2002). Runtime service composition via logic-based program synthesis (Doctoral dissertation, KTH).

55. Paventhan, A. (2007). Grid approaches to data-driven scientific and engineering workflows (Doctoral dissertation, University of Southampton).

56. Warr, W. A. (2012). Scientific workflow systems: Pipeline Pilot and KNIME.*Journal of computer-aided molecular design*, 1-4.

57. Barseghian, D., Altintas, I., Jones, M. B., Crawl, D., Potter, N., Gallagher, J., ... & Hosseini, P. R. (2010). Workflows and extensions to the Kepler scientific workflow system to support environmental sensor data access and analysis.*Ecological Informatics*, *5*(1), 42-50.

58. Bachmann, A., Kunde, M., Litz, M., & Schreiber, A. (2009, May). A dynamic data integration approach to build scientific workflow systems. In *Grid and Pervasive Computing Conference, 2009. GPC'09. Workshops at the* (pp. 27-33). IEEE.

59. McPhillips, T., Bowers, S., Zinn, D., & Ludäscher, B. (2009). Scientific workflow design for mere mortals. *Future Generation Computer Systems*, *25*(5), 541-551.

60. De Roure, D., Goble, C., & Stevens, R. (2009). The design and realisation of the myExperiment virtual research environment for social sharing of workflows.*Future Generation Computer Systems*, *25*, 561-567.

61. Astakhov, V., Bandrowski, A., Gupta, A., Kulungowski, A. W., Grethe, J. S., Bouwer, J., ... & Ellisman, M. (2012). Prototype of Kepler processing workflows for Microscopy and Neuroinformatics. *Procedia Computer Science*, *9*, 1595-1603.

62. Beisiegel, M., Blohm, H., Booz, D., Dubray, J., Colyer, A., Edwards, M., ... & Trieloff, C. (2005). Service component architecture. Building systems using a service oriented architecture. Whitepaper [online], 1-31.

63. Beisiegel, M., Karmarkar, A., Patil, S., Rowley, M. (2011). Service Component Architecture Assembly Specification Version 1.1. *Open Service Oriented Architecture* http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec.pdf

64. Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., ... & Zhao, Y. (2006). Scientific workflow management and the Kepler system. Concurrency and Computation: Practice and Experience, 18(10), 1039-1065.

65. Barker, A., & Van Hemert, J. (2008). Scientific workflow: a survey and research directions. In Parallel Processing and Applied Mathematics (pp. 746-753). Springer Berlin Heidelberg.

66. Callahan, S. P., Freire, J., Santos, E., Scheidegger, C. E., Silva, C. T., & Vo, H. T. (2006). Managing the evolution of dataflows with vistrails. In Data Engineering Workshops, 2006. Proceedings. 22nd International Conference on (pp. 71-71). IEEE.

67. Goecks, J., Nekrutenko, A., Taylor, J., & Team, T. G. (2010). Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome Biol, 11(8), R86.

68. Deelman, E., Singh, G., Su, M. H., Blythe, J., Gil, Y., Kesselman, C., ... & Katz, D. S. (2005). Pegasus: A framework for mapping complex scientific workflows onto distributed systems. Scientific Programming, 13(3), 219-237.

69. Ghanem, M., Curcin, V., Wendel, P., & Guo, Y. (2008). Building and using analytical workflows in discovery net. Data Mining Techniques in Grid Computing Environments, 119.

70. Deelman, E., Gannon, D., Shields, M., & Taylor, I. (2009). Workflows and e-Science: An overview of workflow system features and capabilities. Future Generation Computer Systems, 25(5), 528-540.

71. Howe, B., Lawson, P., Bellinger, R., Anderson, E., Santos, E., Freire, J., ... & Silva, C. (2008, December). End-to-end escience: Integrating workflow, query, visualization, and provenance at an ocean observatory. In eScience, 2008. eScience'08. IEEE Fourth International Conference on (pp. 127-134). IEEE.

72. McPhillips, T., Bowers, S., Zinn, D., & Ludäscher, B. (2009). Scientific workflow design for mere mortals. Future Generation Computer Systems, 25(5), 541-551.

73. Tan, W., Madduri, R., Nenadic, A., Soiland-Reyes, S., Sulakhe, D., Foster, I., & Goble, C. (2010). CaGrid Workflow Toolkit: A taverna based workflow tool for cancer grid. BMC bioinformatics, 11(1), 542.

74. Fisher, P., Hedeler, C., Wolstencroft, K., Hulme, H., Noyes, H., Kemp, S., ... & Brass, A. (2007). A systematic strategy for the discovery of candidate genes responsible for phenotypic variation. BMC Bioinformatics, 8(Suppl 8), P7.

75. Goble, C. A., Bhagat, J., Aleksejevs, S., Cruickshank, D., Michaelides, D., Newman, D., ... & De Roure, D. (2010). myExperiment: a repository and social network for the sharing of bioinformatics workflows. Nucleic acids research, 38(suppl 2), W677-W682.

76. De Roure, D., Goble, C., & Stevens, R. (2009). The design and realisation of the myExperiment virtual research environment for social sharing of workflows. Future Generation Computer Systems, 25, 561-567.

77. Bowers, S. (2012). Scientific workflow, provenance, and data modeling challenges and approaches. Journal on Data Semantics, 1(1), 19-30.

78. SHIWA: SHaring Interoperable Workflows for large-scale scientific simulation on Available DCIs. http://www.shiwa-workflow.eu, 2011

79. Plankensteiner, K., Prodan, R., Janetschek, M., Fahringer, T., Montagnat, J., Rogers, D., ... & Kacsuk, P. (2013). Fine-Grain Interoperability of Scientific Workflows in Distributed Computing Infrastructures. Journal of Grid Computing, 1-27.

80. Plankensteiner, K., Montagnat, J., & Prodan, R. (2011, November). IWIR: a language enabling portability across Grid workflow systems. In Proceedings of the 6th workshop on Workflows in support of large-scale science (pp. 97-106). ACM.

81. Belhajjame, K., Corcho, O., Garijo, D., Zhao, J., Missier, P., Newman, D., ... & Goble, C. (2012). Workflow-centric research objects: First class citizens in scholarly discourse. In Proceedings of the ESWC2012 Workshop on the Future of Scholarly Communication in the Semantic Web.

82. Bechhofer, S., Buchan, I., De Roure, D., Missier, P., Ainsworth, J., Bhagat, J., ... & Goble, C. (2013). Why linked data is not enough for scientists. Future Generation Computer Systems, 29(2), 599-611.

83. De Roure, D., Goble, C., & Stevens, R. (2009). The design and realisation of the myExperiment virtual research environment for social sharing of workflows. Future Generation Computer Systems, 25, 561-567.

84. Roure, D. D., Goble, C., Aleksejevs, S., Bechhofer, S., Bhagat, J., Cruickshank, D., ... & Zhao, J. (2010, December). The evolution of myexperiment. In e-Science (e-Science), 2010 IEEE Sixth International Conference on (pp. 153-160). IEEE.

85. Bechhofer, S., De Roure, D., Gamble, M., Goble, C., & Buchan, I. (2010). Research objects: Towards exchange and reuse of digital knowledge. The Future of the Web for Collaborative Science.

86. Green, C. (1969). Application of theorem proving to problem solving (No. SRI-TR-4). Sri International Menlo Park CA Artificial Intelligence Center.

87. Manna, Z., & Waldinger, R. (1980). A deductive approach to program synthesis. ACM Transactions on Programming Languages and Systems (TOPLAS), 2(1), 90-121.

88. Gulwani, S. (2012, September). Synthesis from examples: Interaction models and algorithms. In Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2012 14th International Symposium on (pp. 8-14). IEEE.

89. Qin, J., & Fahringer, T. (2012). Semantic-Based Scientific Workflow Composition. In Scientific Workflows (pp. 115-134). Springer Berlin Heidelberg.

90. Deelman, E., & Gil, Y. (2006, December). Managing large-scale scientific workflows in distributed environments: Experiences and challenges. In e-Science and Grid Computing, 2006. e-Science'06. Second IEEE International Conference on (pp. 144-144). IEEE.

91. Altintas, I., Wang, J., Crawl, D., & Li, W. (2012, March). Challenges and approaches for distributed workflow-driven analysis of large-scale biological data: vision paper. In Proceedings of the 2012 Joint EDBT/ICDT Workshops (pp. 73-78). ACM.

92. Aupperle, B., Haney, D., Robbins, P. (2010). Service Component Architecture Client and Implementation Model for C++ Specification Version 1.1. *Open Service*

Oriented Architecture http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-cppcni-1.1-spec.pdf

93. Booz, D., Edwards, M., Karmarkar, A. (2011). Service Component Architecture SCA-J Common Annotations and APIs Specification Version 1.1 *Open Service Oriented Architecture* http://docs.oasis-open.org/opencsa/sca-j/sca-javacaa-1.1-spec.pdf

94. Holdsworth, S., Karmarkar, A. (2013). Service Component Architecture Web Service Binding Specification Version 1.1 *Open Service Oriented Architecture* http://docs.oasis-open.org/opencsa/sca-bindings/sca-wsbinding-1.1-spec.pdf

95. Holdsworth, S., Karmarkar, A. (2011). Service Component Architecture JMS Binding Specification Version 1.1 *Open Service Oriented Architecture* http://docs.oasis-open.org/opencsa/sca-j/sca-javacaa-1.1-spec.pdf

96. Booz, D., Edwards, M, J,. Malhotra, A. (2011). SCA Policy Framework Version 1.1 *Open Service Oriented Architecture* http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1.pdf

97. Goecks, J., Nekrutenko, A., Taylor, J., & Team, T. G. (2010). Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome Biol, 11(8), R86.

98. Blankenberg, D., Kuster, G. V., Coraor, N., Ananda, G., Lazarus, R., Mangan, M., ... & Taylor, J. (2010). Galaxy: A Web- Based Genome Analysis Tool for Experimentalists. Current protocols in molecular biology, 19-10.

99. Abouelhoda, M., Issa, S. A., & Ghanem, M. (2012). Tavaxy: Integrating Taverna and Galaxy workflows with cloud computing support. BMC bioinformatics, 13(1), 77.

100. Callahan, S. P., Freire, J., Santos, E., Scheidegger, C. E., Silva, C. T., & Vo, H. T. (2006, June). VisTrails: visualization meets data management. In Proceedings of the 2006 ACM SIGMOD international conference on Management of data (pp. 745-747). ACM.

101. Callahan, S. P., Freire, J., Santos, E., Scheidegger, C. E., Silva, C. T., & Vo, H. T. (2006). Managing the evolution of dataflows with vistrails. In Data Engineering Workshops, 2006. Proceedings. 22nd International Conference on (pp. 71-71). IEEE.

102. Taverna Player, http://www.taverna.org.uk/developers/work-in-progress/taverna-player/

103. Taverna Workflow Components, http://www.taverna.org.uk/developers/work-in-progress/components/

104. Erl, T. (2005). *Service-oriented architecture* (Vol. 8). New York: Prentice Hall.

105. Záková, M., Kremen, P., Zelezny, F., & Lavrac, N. (2011). Automating knowledge discovery workflow composition through ontology-based planning. Automation Science and Engineering, IEEE Transactions on, 8(2), 253-264.

106. Gulwani, S., Jha, S., Tiwari, A., & Venkatesan, R. Component-based Synthesis Applied to Bitvector Programs.

107. Stickel, M., Waldinger, R., Lowry, M., Pressburger, T., & Underwood, I. (1994). Deductive composition of astronomical software from subroutine libraries. In Automated Deduction—CADE-12 (pp. 341-355). Springer Berlin Heidelberg.

108. Johnson, T. A., & Eigenmann, R. (2006, June). Context-sensitive domain-independent algorithm composition and selection. In ACM SIGPLAN Notices (Vol. 41, No. 6, pp. 181-192). ACM.

109. Jha, S., Gulwani, S., Seshia, S. A., & Tiwari, A. (2010, May). Oracle-guided component-based program synthesis. In Software Engineering, 2010 ACM/IEEE 32nd International Conference on (Vol. 1, pp. 215-224). IEEE.

110. Lieberman, H. (2001). Your wish is my command: Programming by example. Morgan Kaufmann.

111. Bartalos, P., & Bieliková, M. (2012). Automatic dynamic web service composition: A survey and problem formalization. Computing and Informatics, 30(4), 793-827.

112. Foster, I., & Kesselman, C. (Eds.). (2003). The Grid 2: Blueprint for a new computing infrastructure. Elsevier.

113. OSGi Alliance, http://www.osgi.org/

114. Lara, R., Polleres, A., Lausen, H., Roman, D., de Bruijn, J., & Fensel, D. (2005). A conceptual comparison between WSMO and OWL-S. WSMO Final Draft D, 4, 44.

115. Dean, T. L., & Kambhampati, S. (1997). Planning and Scheduling.

116. Deelman, E., Singh, G., Su, M. H., Blythe, J., Gil, Y., Kesselman, C., ... & Katz, D. S. (2005). Pegasus: A framework for mapping complex scientific workflows onto distributed systems. Scientific Programming, 13(3), 219-237.

117. Gil, Y., Ratnakar, V., Kim, J., González-Calero, P. A., Groth, P., Moody, J., & Deelman, E. (2011). Wings: Intelligent workflow-based design of computational experiments. IEEE Intelligent Systems, 26(1), 62-72.

118. Alrifai, M., Risse, T., Dolog, P., & Nejdl, W. (2009, January). A scalable approach for qos-based web service selection. In Service-Oriented Computing–ICSOC 2008 Workshops (pp. 190-199). Springer Berlin Heidelberg.

119. Jiang, W., Zhang, C., Huang, Z., Chen, M., Hu, S., & Liu, Z. (2010, July). Qsynth: A tool for qos-aware automatic service composition. In Web Services (ICWS), 2010 IEEE International Conference on (pp. 42-49). IEEE.

120. Cerezo, N., Montagnat, J., & Blay-Fornarino, M. (2013). Computer-Assisted Scientific Workflow Design. Journal of grid computing, 11(3), 585-612.

121. Wolstencroft, K., Haines, R., Fellows, D., Williams, A., Withers, D., Owen, S., ... & Goble, C. (2013). The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. Nucleic acids research, gkt328.

122. Vicario, S., Balech, B., Donvito, G., Notarangelo, P., & Pesole, G. (2012). The BioVel Project: Robust phylogenetic workflows running on the GRID. EMBnet. journal, 18(B), pp-77.

123. BiodiversityCatalogue, https://www.biodiversitycatalogue.org/

124. Web Services Description Language (WSDL) Version 2.0, http://www.w3.org/TR/wsdl20/

125. SOAP Version 1.2, http://www.w3.org/TR/soap12-part1/

126. Fielding, R. T., & Taylor, R. N. (2002). Principled design of the modern Web architecture. ACM Transactions on Internet Technology (TOIT), 2(2), 115-150.

127. Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures (Doctoral dissertation, University of California, Irvine).

128. McCarthy, J. (1963). Situations, actions, and causal laws (No. AI-MEMO-2). STANFORD UNIV CA DEPT OF COMPUTER SCIENCE. Chicago

129. McCarthy, J., & Hayes, P. (1968). Some philosophical problems from the standpoint of artificial intelligence (pp. 463-502). USA: Stanford University.

Appendix A - Scientific Workflow System Implementation

This appendix provides an overview of the manner in which the SWS functionality which is to be exposed by the API is achieved within each of the existing SWS considered, namely Triana, Taverna and Kepler.

A.1 Starting a new workflow composition

Triana

In the Triana SWS each workflow is represented by an XML file, known as a "TaskGraph". This file represents the components, or Tasks as they are called within Triana, that are present within the workflow and the relationships that have been defined between them. In order to begin creating a new workflow within Triana the system must create a new TaskGraph file; this is achieved by calling the relevant operation of a class called the TaskGraphManager – createTaskGraph().

This operation creates a new empty TaskGraph object which can then be populated with the components required for a workflow composition. In order to create a visualisation of this new TaskGraph the operation AddParentTaskGraphPanel is utilised, when provided with a TaskGraph object as a parameter this creates a JPanel containing a visualisation of the current state of that TaskGraph.

Taverna

As with the Triana SWS each workflow in Taverna is represented by an XML file, but in this system each workflow is called a "ScuflModel". Creating a new workflow within Taverna involves creating a new instance of ScuflModel which will hold all of the information to be added to the workflow. A visualisation of this ScuflModel can be created using the ScuflSVGDiagram class, when given the ScuflModel as a variable.

Kepler

In the same manner as in both Triana and Taverna, Kepler workflows are represented through the use of an XML file. This file records the components, properties and

connections that make up the workflow. In Kepler the XML files to represent workflow models are called MoML (MOdelling Mark-up Language) files. The process of starting a workflow composition within Kepler is more complex than that of the other SWSs; the sequence diagram in Figure 10-1 outlines the various steps involved. Within the Kepler system creating a new workflow is achieved by first making a new instance of Workspace, a class which acts as a space to contain and refer to workflow objects, and then creating an instance of TypedCompositeActor using this Workspace.

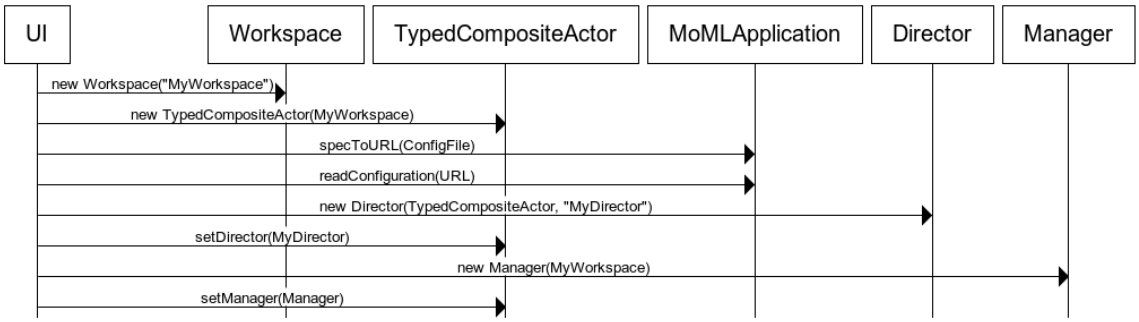


Figure 10-1 Sequence diagram showing Kepler startup activity

As well as creating both the Workspace and TypedCompositeActor, in order to successfully create and utilise a workflow within Kepler there are several further elements that must be specified. Firstly an instance of Configuration is required; this class is used to facilitate the interaction of the user with the workflow model. The Configuration refers to a XML config file which is provided by the Kepler system.

Secondly in order for a workflow model to be successfully executed once complete we must also create both a Manager and a Director for that workflow. The Manager class is used to initiate the execution of a workflow, and the Director is used to control the manner in which a workflow executes including the number of iterations performed or the length of time for which an execution should run.

Finally in order to create a visualisation of a Kepler workflow the classes Tableau and ActorGraphFrame are used to create a graph of the model which can then be displayed as a JGraph.

Summary

Through these functions from the Triana, Taverna and Kepler systems we can acquire both an object which represents our current workflow, and an object which we can use to display that workflow within a user interface. Whilst the Kepler system requires us to perform several extra operations in order to successfully create a new workflow, once this has been performed there are no significant differences that would complicate our API.

A.2 Locating available workflow components

Triana

In Triana the components available to a user are represented by a searchable tree; this lists components by their domain of operation such as ImagProc (Image Processing). Within the Triana system this list of tools is represented by an object called the ToolTable, generating a new instance of this object is achieved using the TaskGraphManager operation getToolTable. Utilising the class ToolTreeModel this ToolTable object can then be used to create a visualisation of the available workflow components within a standard JTree.

Taverna

Similar to the Triana approach, components within Taverna are made available through a tree structure, however in the case of Taverna the components are listed based on their provider rather than their function. This is due to Taverna relying primarily on distributed components. Taverna uses a ScavengerTreePanel to represent this list of available components, in order to populate the list a new DefaultScavengerTree object must also be created and the operation attachToModel is used to associate this with the current ScuflModel.

As the ScavengerTreePanel is a visual component once created it can be handled in the same manner as any standard component in order to display the list of available components within the user interface.

Kepler

In the same manner as with both Triana and Taverna, components in Kepler are presented to the user using a tree structure. The manner in which Kepler components

are organised within this tree is a combination of the approaches taken by Triana and Taverna. Kepler lists some components by their purpose, similar to Triana, whereas others are listed by either their provider or by the project for which they were developed, similar to Taverna.

In order to display this list of components Kepler uses two classes, LibraryIndex and EntityTreeModel. The LibraryIndex acts as the link to the set of available components and the EntityTreeModel is the means of visualising this list.

Summary

The previous sections describe the operations required to retrieve the list of components which is available within each of the three SWS. Using these operations it is possible to display this list of components within a user interface. The similarity between the mechanisms used to display the component list within each SWS helps to simplify this area of the API.

A.3 Adding components to the workflow

Triana

In order to insert a component into a workflow within the Triana system it is necessary to know the complete name of the component. This is represented by the component's position in the component tree. For example the component StringViewer is listed under the nodes Common and String, this gives it the complete name; `Common.String.StringViewer`. Using the complete name of a component it is then possible to create a new instance of the class Tool to represent that component by using the getTool() operation on the TaskGraph which represents our workflow.

Once the Tool object has been created it is possible to add this component to our workflow using another operation from our TaskGraph object, createTask().

Taverna

Adding a component to a workflow within Taverna is achieved by identifying the component to insert from the list available and creating a new instance of Object to represent that component. The class ProcessorFactory is then used in order to insert this Object into the current ScuflModel using the createProcessor operation.

Kepler

Whilst in both Triana and Taverna the object which represents the current workflow includes a function for adding new components to the workflow, the same functionality is not present within Kepler. Within Kepler each component is represented by a separate Java Class, the process of adding components to a workflow in the Kepler SWS is achieved through creating a new instance of the class that represents that component. Therefore, as Kepler does not provide a simple function for achieving the action of adding components, the approach for this SWS is to identify the name of the Class that represents the component to be added, create a constructor for that class, and use that constructor to create a new instance of the class. By including the TypedCompositeActor which represents our workflow as one of the parameters used in construction of the new class we are able to add the component to our existing workflow.

Summary

The functions described in the previous sections make it possible for the API to provide a call which will insert a new component into the current workflow when utilising any of the underlying SWSs. This API call is complicated somewhat by the differing approach presented by the implementation of the Kepler SWS, necessitating the manual creation of new component classes, however the results are consistent across each system; following these calls the selected component will be added to the current workflow.

A.4 Defining connections between components in the workflow

Triana

When using the Triana system, connections between components are specified by dragging links between components identified input and output nodes within the workflow visualisation. As a representation of what is occurring programmatically this is achieved through the use of the CableInterface class. Connections within the TaskGraph are defined using the CableInterface operation “connect”, this takes the selected output and input nodes and creates the connection between them.

Taverna

The manner in which connections are made between components in a workflow is one of the areas in which the Taverna SWS approach differs from that of both Kepler and Triana. Connections in both Kepler and Triana are created by physically dragging a link between the required components in the workflow visualisation. In Taverna such connections are created using an element of the user interface called the AdvancedModelExplorer. This lists all of the components that are present within the workflow along with various properties associated with those components, from here the user can select an individual component and from a drop down list identify which other component within the workflow they would like to connect it to.

Whilst the user experience of connecting components differs, within the Taverna system these connections between components are achieved in much the same way as within Triana and Kepler. The relevant components input and output ports are identified through their Processor objects and then a DataConstraint between these ports is created and added to the ScuflModel using the addDataConstraint operation.

Kepler

As stated previously from the users perspective Kepler takes a similar approach to Triana for the procedure of specifying connections between components; the workflow visualisation provides endpoints for each component that can be connected by dragging a line between them. The underlying method that supports these connections within Kepler is also similar to both Triana and Taverna. IOPort objects are created to represent the endpoints that are to be connected and the operation “connect” is used from the TypedCompositeActor object that represents our workflow in order to establish the connection.

Summary

Despite the difference in approach that is utilised from the user’s perspective when connecting components in the Taverna SWS, the operations that are performed from the code level are very similar across each of the systems – the ports of the components to be connected are identified, and the object representing the workflow is updated to reflect the connection.

A.5 Executing a composed workflow

Triana

When using Triana the workflow is executed by simply pressing the “run algorithm” button from the user interface. This causes each component within the workflow to execute in turn, with any components that provide a visualisation of their output displaying this on the screen. From a code perspective this is achieved using the LocalServer class, this is created using both the TaskGraph object which represents the users workflow, and the ToolTable object representing the components available within the Triana system. Once created the “run” method of LocalServer can be used to execute the workflow.

Taverna

The execution of workflows is another area in which the approach of Taverna differs from that of both Kepler and Triana. Taverna provides a separate user interface through which the execution of a workflow can be monitored; this allows the user to identify what occurs at run time, as well as providing access to intermediate results passed between components. In the underlying system the execution of a Taverna workflow is achieved through the creation of an EnactorProxy object which is then used to call the operation compileWorkflow. The outcome of performing this execution is then held in a WorkflowInstance object.

Kepler

Execution of workflows within the Kepler system functions in a similar manner to that of the Triana SWS, the components of the workflow are executed and the output from the endpoints is displayed. One key difference with the approach taken by Kepler is the introduction of the Director, an element of the workflow which controls the way in which the workflow executes. In order to execute a Kepler workflow we use the “getManager” operation on the TypedCompositeActor object that represents our workflow, this Manager is then used to call the “run” operation which executes the workflow.

Summary

Through these functions it is possible to execute the current state of a users workflow. Whilst from a users perspective the approach to achieving this outcome differs somewhat between each of the SWSs the API call to execute a workflow is relatively

simple, obtaining an instance of the appropriate "executor" class within each SWS and calling the relevant "run" function.

A.6 View the workflow results

Triana

In the Triana SWS when the user executes their workflow the system automatically opens windows to display the results of those components which produce output. In this way there are no specific steps which must be performed in order to view the workflow results beyond those already performed to execute the workflow.

Taverna

Within the Taverna system the results of a workflow execution are displayed within the same user interface that is utilised to execute the workflow, an example of this element of the UI is shown in Figure 10-2. This displays the results produced by any elements of the workflow that have been defined as “outputs” during composition. The class EnactorInvocation is used to create a JPanel containing the execution and result details relating to the WorkflowInstance created by executing the workflow.

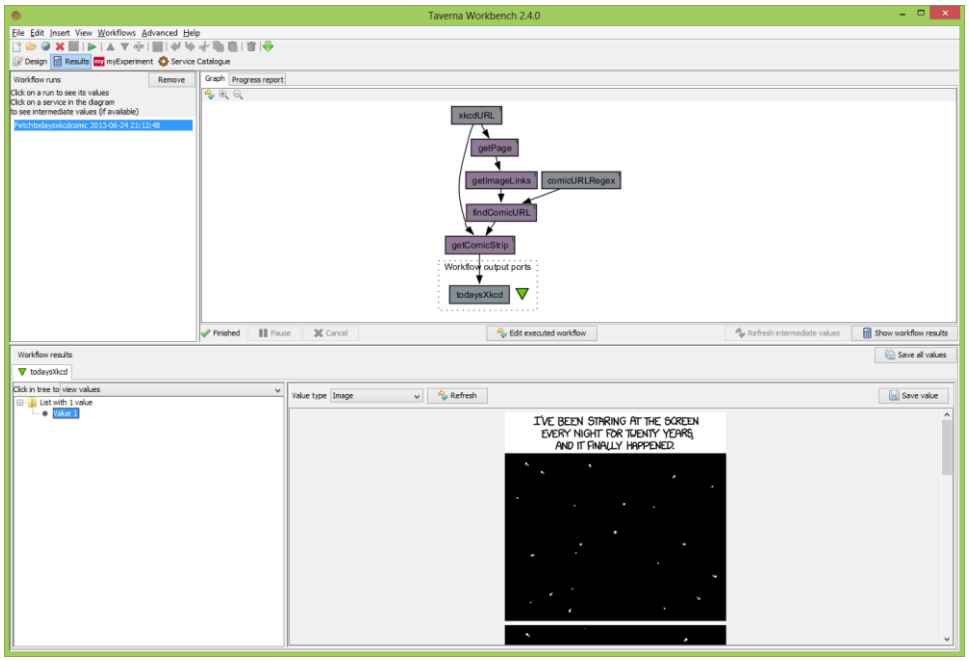


Figure 10-2 Taverna GUI showing the results of an execution

Kepler

Unlike the Taverna system, Kepler does not have any defined interface for inspecting the execution results of a workflow. Similar to the approach taken by Triana, following the execution of a workflow the Kepler user interface will display the results of any “output” components that are present in the workflow. For example a workflow containing an ImageDisplay component would display the result of invoking that component after execution. As with the Triana system this means that no specific functionality must be invoked in order for the results of a workflow execution to be displayed as all the necessary interactions will be completed by invoking the execution itself.

Summary

Both the Triana and Kepler SWSs will automatically display the results of any components which produce suitable output when these are executed, as such the only additions to the API which are required to view workflow output are to accommodate the Taverna SWS. Upon requesting the API call execute when using Taverna the API will automatically perform steps described previously in order to display the separate UI which is required to view output of Taverna workflows.

A.7 Scientific Workflow System Implementation Summary

This appendix has described the manner in which the key functionality to be presented by the API is achieved in each of the considered SWSs, Kepler, Triana, and Taverna. Despite differences which are present in these system from a user perspective, for example they have differing approaches to displaying the current state of a workflow and locating the components which a user can insert into a workflow within their own UIs, from an implementation perspective they are suitably similar. As such the API itself does not need to be overly complicated by a requirement to satisfy any particular SWS and the functions it defines can be implemented in a consistent manner across each of the SWSs considered.

Appendix B - Scenario Composition Walk-throughs

This appendix provides detailed walkthroughs of the steps taken by a user to compose each of the workflow scenarios identified in Section 8.2.1.

B.1 Scenario A

The first scenario is a basic mathematical operation, performing a remainder calculation on a user entered number and printing the result. As described in the main text the system adopts the overall composition approach of identifying abstract components to represent workflow tasks, specialising these to appropriate implementable components, and implementing the relevant suggestions to connect these components, with this process being repeated if further components and connections are required.

Step 1: Identify Initial Components

The user begins by identifying a number of abstract components to represent the processes that will be involved in the complete scenario. As discussed in Chapter 7 the user interface provides the user with a list of available abstract components which can be inserted into the composition.

The initial set of top level abstract components presented to the user contains the following options:

- Database Component
- I/O Component
- Integration Component
- Modelling Component
- Operation Component
- Image Component

From this list the user identifies the abstract "I/O Component" as a useful starting point, based on the knowledge that the scenario requires the user to provide input in the form of the number they wish to manipulate, as well as output in the form of the outcome of the remainder operation. Having inserted this component the user is now free to either

select another abstract component to insert, or to begin specialising "I/O Component" to a concrete executable component. Knowing that the scenario centres on performing a mathematical operation the user also decides to insert the abstract "Operation Component". At this stage, having identified abstract components to perform both their input and output requirements, as well as to complete the calculation the user proceeds to the specialisation stage of the composition process.

Step 2: Specialise Abstract Components

Once the abstract components have been added to the workflow composition the system begins the process of inspecting the metadata ontology in order to generate suggestions for how the user could proceed. The suggestions that the system provides for specialising "I/O Component" are provided in Table 10-1.

Specialise I/O Component to:	
• Constant	• Line Writer
• Display	• Sequence
• File Reader	• String Constant
• File Writer	• Token Reader
• Line Reader	• Zip Files

Table 10-1 Suggestions to Specialise I/O Component

Understanding that Scenario A requires input of a numerical value on which to operate the user identifies the input component "Constant" as a good candidate to achieve their goals. At this point the user may also determine that the output component "Display" would also be useful for completing this scenario, in which case they could either choose to insert a second abstract "I/O Component" and specialise this to "Display" or simply insert the component directly.

Following these steps the user has now included both the components required to perform the input and output required of Scenario A. However, as the composition still includes a second abstract "Operation Component" the system will provide the user with further suggestions for how to proceed. Table 10-2 lists the specialisation suggestions generated for this abstract component.

Specialise Operation Component to:	
• Absolute Value	• Remainder
• Add or Subtract	• Round
• Decimal Format Converter	• Scale
• Multiply or Divide	

Table 10-2 Suggestions to Specialise Operation Component

From this list the user identifies that the component "Remainder" is the desired specialisation for the abstract "Operation Component". At this point the user has now satisfied the main operational requirements of the scenario; providing numerical input, operating on the number, and displaying the output. As the user identifies that no further components should be required, from here the remaining activity is to correctly connect these components.

Step 3: Connecting Components

As before the user can choose to either follow further guidance from the system in order to connect their selected components, or if confident perform these connections manually. Assuming that the user is not confident with the manner in which these components should be connected they would be given a number of options for how to proceed, based on a composition containing the components "Constant", "Display" and "Remainder" the system would provide the connection suggestions as listed in Table 10-3.

Suggestions for components to connect within workflow:
• Connect Constant.output and Display.input as both have matches
• Connect Constant.output and Remainder.input as both have matches
• Connect Remainder.output and Display.input as both have matches

Table 10-3 Possible connections between Constant, Remainder and Display

Based on the suggestions provided the user identifies the first option, to connect "Constant" directly to "Display", as redundant; given that this would leave "Remainder" unconnected. Implementing the remaining suggestions results in the components being

successfully connected in sequence, and leaves each component satisfied as its input and output ports are now connected. The structure for scenario A is now completed.

Step 4: Defining Parameters

The remaining step is to configure both the input for the scenario and the divisor to be used in the remainder operation, this is performed by setting the relevant parameter of the "Constant" and "Remainder" components, the "value" and "divisor" parameters respectively. The UI does not present suggestions to the user to indicate that this configuration is required, and so the user must manually perform this step. The system utilises the implementation of the underlying SWS in order to define component parameters, to complete the workflow the user must double-click on each component to input the desired values.

B.2 Scenario B

This scenario represents a basic image processing exercise consisting of three primary activities; identifying an image to work with, performing a rotation on that image, and displaying the result.

Step 1: Identify Initial Components

As before the user begins the composition process by selecting a starting set of components, chosen from the list of top level abstract components which are presented via the user interface:

- Database Component
- I/O Component
- Integration Component
- Modelling Component
- Operation Component
- Image Component

As this scenario is primarily focussed on performing an image processing task the user identifies and inserts the “Image Component” abstract from the list of available abstract components. As with the previous scenario, at this stage the user can choose either to

insert further abstract components, or proceed to specialising the Image Component. Given that each step of the scenario involves working with images the user may decide this is the only relevant abstract component they could choose.

Step 2: Specialise Abstract Components

After selecting inserting the abstract "Image Component" the system provides the user with a number of suggestions for how to specialise this component based on information from the metadata ontology. At this stage the list of suggestions would simply be all of the components which are implementations of the Image Processing abstract as there is no other information to inform the system of further suggestions. Based on this selection the suggestions which the system would offer to the user are listed in Table 10-4.

Specialise Image Component to:	
• Brightness	• ImageReader
• Contrast	• ImagetoString
• Converter	• Rotate
• ImageDisplay	• StringtoImage
• ImageJ	• URLtoImage

Table 10-4 Suggestions to specialise Image Component

As the goal of Scenario B is to perform the rotation of an image the user would identify the “Rotate” component as a potential option to achieve their goal.

Step 3: Inserting Additional Components

After inserting the “Rotate” component into the workflow composition the user has a number of choices for how to proceed. A user with prior knowledge of the image processing components within Kepler may identify that the ImageDisplay and ImageReader components are required to perform the necessary tasks of selecting an image to work with and displaying the results, such a user could chose to add these components directly to the composition. Alternatively if the user is not aware of the specific components which perform these steps they can repeat the process from the previous steps, based on their knowledge that inserting an “Image Component”

provides suggestions that will specialise this to other components which could perform the tasks they require.

However if the user is still unsure of the next step to take at this stage they can inspect the list of suggestions which the system provides. Based on the fact that the user has now inserted the “Rotate” component into their composition the system will inspect the metadata ontology to identify additional components which are compatible with the input and output ports of this component and suggest these to the user for inclusion within their composition. A selection of the suggestions which the system would provide to the user at this stage is shown in Table 10-5.

Suggestions for components to add to workflow:
• add a Brightness component (matches with input and output for Rotate)
• add a Contrast component (matches with input and output for Rotate)
• add an ImageDisplay component (matches with output from Rotate)
• add an ImageJ component (matches with output from Rotate)
• add an ImageReader component (matches with input for Rotate)
• add an ImagetoString component (matches with output from Rotate)
• add a StringtoImage component (matches with input for Rotate)
• add an URLtoImage component (matches with input for Rotate)

Table 10-5 Suggestions compatible with Rotate

In addition to the suggestions shown in Table 10-5 further, potentially less helpful components would be suggested as possible connections to the Rotate component. For example Scenario C discussed later utilises a component “GARPPrediction” which has an output port that is compatible with the input for Rotate, as such this component will be suggested for addition to the composition. However, as the suggestion system also takes into account knowledge of the domain in which components are utilised the suggestion for GARPPrediction, a component from the BioInformatics domain, would have a lower ranking than the Image Processing components previously listed. In this way this approach ensures that whilst all components that are technically compatible with Rotate are listed, those which are unlikely to be used in connection with it do not distract the user from those components which are of potential use.

Based on the goals of the scenario, reading an image, rotating it and displaying the result, the user chooses to implement suggestions to add ImageReader and ImageDisplay to the composition.

Step 4: Connecting Components

Now that the user has included components to achieve the tasks identified in the scenario they can begin implementing suggestions on how to connect and sequence those components. Based on a composition including Rotate, ImageReader, and ImageDisplay components the suggestions which the system would provide for connections are listed in Table 10-6.

Suggestions for components to connect within workflow:
• Connect ImageDisplay.input and Rotate.output as both have matches
• Connect ImageReader.output and Rotate.input as both have matches
• Connect ImageReader.output and ImageDisplay.input as both have matches

Table 10-6 Possible connections between ImageDisplay, ImageReader and Rotate

Eliminating the option to connect ImageReader and ImageDisplay directly, as this would not achieve the goals of their scenario, the user chooses to implement connections between ImageReader and Rotate, and between Rotate and ImageDisplay. On the implementation of these connections the structure of Scenario B has now been satisfied.

Step 5: Defining Parameters

As with Scenario A the final step required is to configure the components within the workflow composition, in this case to provide the composition with the location of the image that the user wishes to process. As before this is performed by manually double-clicking the configurable component, in this case ImageReader, and setting the required parameter.

B.3 Scenario C

Scenario C is an example from the domain of biodiversity informatics, based around creating a workflow to achieve the modelling of species distributions. This involves obtaining relevant environmental and species presence data, and feeding these into a series of bioclimatic modelling components. The result of this modelling is then fed into a visualisation component in order to provide a graphical representation of the species distribution investigated.

Step 1: Identify Modelling Component

Composition begins with the user identifying the abstract component required to achieve the goal of modelling species distribution. As before the system presents the user with a selection of abstract components such as:

- Visualisation Component
- Database Component
- Operation Component
- Modelling Component

In this scenario the goal of composition is to perform modelling of species' distribution, as a result we are taking the starting point of the user identifying that an instance of "Modelling Component" is potentially a beneficial component to include. Other possible starting points include identifying that a Database or Visualisation Component may be required; however this would have limited effect on the eventual outcomes of the walk-through, with the user effectively performing the same steps but in a different order.

After including the "Modelling Component" abstract the user could choose to introduce further abstract components from those available, but for the purpose of this example we will assume they have no further knowledge of which to include. Upon inserting the "Modelling Component" into the workflow the system begins to provide the user with suggestions for progressing the workflow.

Step 2: Specialise Modelling Component

As there are no other components present in the workflow the system will provide the user with suggestions for how to specialise "Modelling Component". Whilst the system is therefore unable to narrow down the list of suggestions based on metadata about

other components present in the composition, information such as the user's past interaction with components which are implementations of "Modelling Component" as well as information about the "domain" to which those components belong can still be used to improve the list of suggestions provided to the user. Based on this information the specialisations listed in Table 10-7 are suggested.

Specialise Modelling Component to:
• GARPPresampleLayers
• GARPPrediction
• GARPAlgorithm

Table 10-7 Suggestions to specialise Modelling Component

Assuming the user is from the domain of biodiversity informatics it is likely that they will be aware of the GARP algorithm and its use in the modelling of species distributions. In such a case the user would be confident in adding any of the suggested components to the workflow confident that they would be required. However a user who was not aware of the purpose of the GARP algorithm would not know whether these suggestions were of benefit. In such a case the assisted composition approach means there are only a limited number of options to explore. As GARPPresampleLayers is the top suggestion provided by the system the user chooses to implement this specialisation.

This change results in a workflow containing one component, GARPPresampleLayers. Again the user has the choice to return to inserting further abstract components, based on their knowledge of what they want to achieve, or to continue inspecting the suggestions the system provides in order to identify the changes to implement.

Step 3: Inserting Additional Component 1 - GARPAlgorithm

With the GARPPresampleLayers component included in the workflow the next step is to augment the workflow by providing the user with suggestions for suitable additional components. The system inspects the knowledge stored in the metadata ontology regarding this component to identify other components which are compatible and desirable to include in the workflow. Figure 10-3 shows the knowledge which is stored regarding the GARPPresampleLayers component.

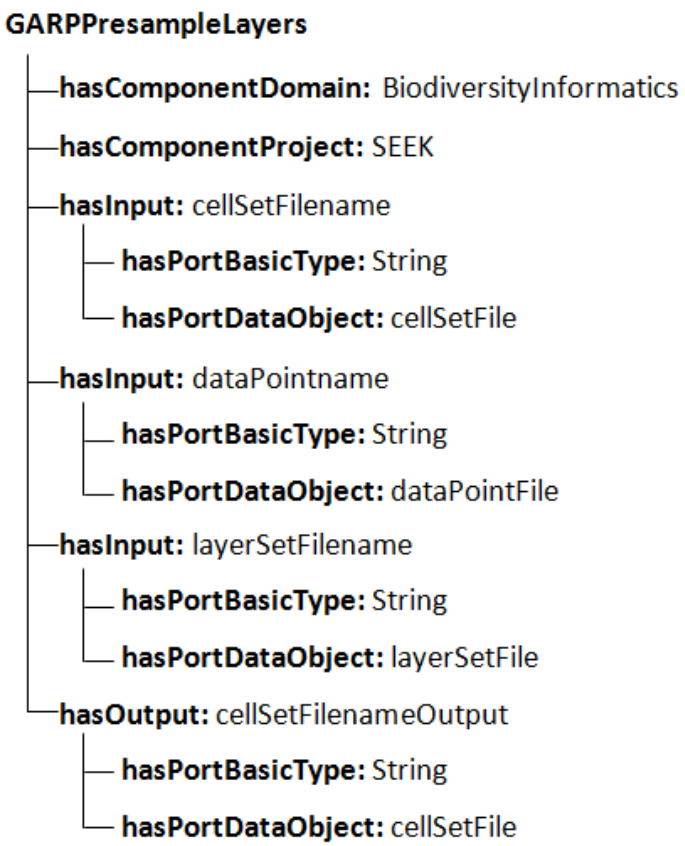


Figure 10-3 GARPPresampleLayers Component Metadata

Using this information the system inspects the rest of the ontology to identify those components which possess matches with the GARPPresampleLayers component. The input and output port metadata is used to identify those components whose ports are logically compatible, the domain and project metadata is used to narrow this selection to those components from the same working domain or project as the user and finally the user's personal history of interactions is utilised to identify those suggestions most commonly implemented by the user. As a result the system provides the user with GARPAlgorithm as the primary component to add to the workflow as shown in Table 10-8.

Suggestions for components to add to workflow:
• GARPAlgorithm(possible connection to cellSetFilenameOutput)

Table 10-8 Components to add to GARPPresampleLayers

This component is provided as the most ideal suggestion as it has an input port, cellSetFilename, which shares the same BasicPortType, String, and PortDataObject, cellSetFile, as the output port, cellSetFilenameOutput from GARPPresampleLayers. Additionally the system identifies further components which may be candidates for including in the workflow, however these are all identified as weak connections, having only a single match with GARPPresampleLayers.

As the system has identified GARPAlgorithm as the prime candidate for including within the workflow the user implements this suggestion; this results in a workflow containing both GARPPresampleLayers and GARPAlgorithm.

Step 4: Connecting Components – GARPPresampleLayers + GARPAlgorithm

At this stage the system now has two categories of suggestion to provide the user – further components to add to the workflow based on the presence of these two components, and connections that could be made between these two components.

As identified previously the metadata matches between component ports cellSetFilename and cellSetFilenameOutput are the reason GARPAlgorithm was suggested as a possible addition to the workflow, therefore creating a connection between these two ports is also now identified as a possible connection step for the user to take. Similar to the addition suggestions provided previously there are also further suggestions for connections to create between the two components now present in the workflow, however these again are not identified as ideal due to only containing matches on one element of metadata, BasicPortType.

The user has three choices at this stage – implement the suggested connection between GARPPresampleLayers and GARPAlgorithm, explore the list of suggestions for further components to add to the workflow, or add an additional abstract component to further specialise. By choosing to implement the connection between GARPAlgorithm and GARPPresampleLayers the user is reducing the set of ideal suggestions remaining, and therefore taking themselves closer to a completed composition; once this suggestion has been implemented the system's suggestions update to indicate that there are no further ideal connections to make, therefore the user is presented with

suggestions for additional components which could be included in the composition, or they can choose to insert a new abstract component.

Step 5: Inserting Additional Component - GARPPrediction

Based on the new content of the workflow, GARPPresampleLayers and GARPAlgorithm components connected by the ports cellSetFilenameOutput and cellSetFilename, the system attempts to identify further components which possess matches with the knowledge stored about these components. GARPPresampleLayers still possesses the same metadata as detailed previously and so the system will inspect the framework for components suited for connection, although since its output port is already connected, matches to this element are discounted. Additionally the knowledge of GARPAlgorithm stored in the framework is also utilised in the process of identifying matches, the elements of metadata the system stores regarding this component are shown in Figure 10-4.

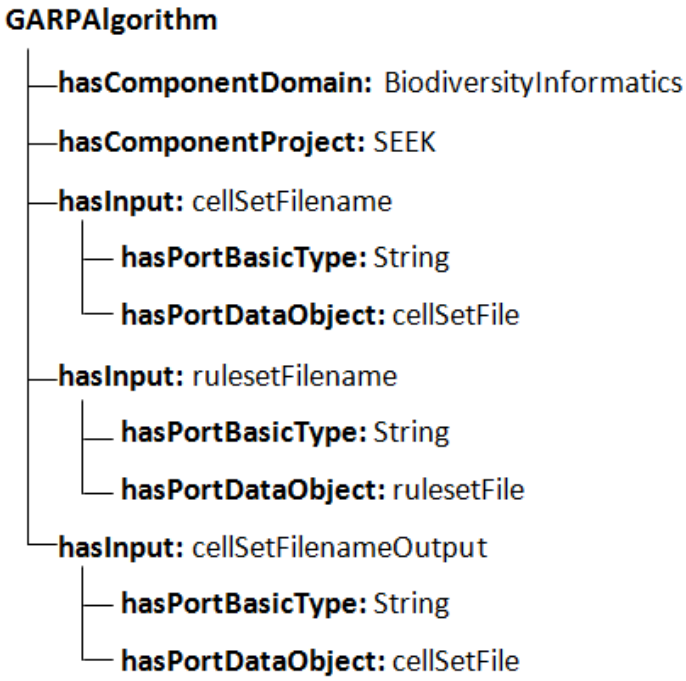


Figure 10-4 GARPAlgorithm Component Metadata

Again as the port cellSetFilename is already utilised in the connection to GARPPresampleLayers this is discounted from the suggestion process. Utilising this

knowledge the system is able to identify the GARPPrediction component as an ideal candidate for addition to the workflow as shown in Table 10-9.

Suggestions for components to add to workflow:
• GARPPrediction (possible connection to GARPAlgorithm)

Table 10-9 Components to add to GARPAlgorithm and GARPPresampleLayers

GARPPrediction is identified as an ideal component to add to the workflow as it has an input port, ruleSetFilename, which shares both the BasicPortType, String, and PortDataObject, ruleSetFile, with the output port ruleSetFilename output provided by GARPAlgorithm. As before further lower ranked suggestions not listed here would be provided by the system, however due to possessing only matches with the current content of the workflow based on the BasicPortType metadata it is reasonable to accept that the user would dismiss these options.

At this stage the only ideal suggestion available to the user is to add the GARPPrediction component, there are no connection suggestions as the connection between GARPPresampleLayers and GARPAlgorithm has satisfied both of those components. Implementing this addition results in a workflow containing the three GARP components.

Step 6: Connecting Components – GARPAlgorithm + GARPPrediction

Once again the user has three options available; to implement further addition suggestions, to explore the available connection suggestions, or to introduce further abstract components to the workflow. As before the user can choose to implement the reason why GARPPrediction was suggested as an addition to the workflow, and connect this with the GARPAlgortihm component. Implementing this connection results in a workflow with GARPPresampleLayers, GARPAlgorithm, and GARPPrediction components connected in sequence.

Step 7: Inserting Additional Component - ImageJ

The list of available connections and additions is once again updated to reflect the new state of the workflow, the connections present between each of the GARP components

result in no further ideal connections being suggested at this stage, leaving only further additions as a route forward for the user. Given the current state of the workflow the system identifies the component ImageJ as an ideal candidate for addition to the workflow as shown in Table 10-10.

Suggestions for components to add to workflow:
• ImageJ (possible connection to GARPPrediction)

Table 10-10 Components to add to GARPPrediction, GARPAlgorithm, and GARPPresampleLayers

ImageJ is identified as an ideal component to add to the composition as its input port has a PortDataObject of genericImageFile matching the PortDataObject of GARPPredicitons output port outputBMPFilename. Further lower ranking suggestions are once again provided.

Step 8: Connecting Components – GARPPrediction + ImageJ

Inserting the ImageJ component into the workflow updates the connection suggestions to indicate the connection between this component and GARPPrediction as a possible further step. As this is the only ideal suggestion remaining the user can choose to implement this or to add further abstract components.

Implementing this latest connection results in workflow with GARPPresampleLayers, GARPAlgorithm, GARPPrediction and ImageJ components connected in sequence. These components form the basis of the overall goal of species distribution desired by this scenario.

Step 9: Inserting Additional Components – StringConstant

At this stage the available suggestions update to indicate that only non-ideal connections and additions are available (those suggestions where only the basic port type matches), the remaining step requires the user to identify the need for further input to the GARP components in order to achieve success. The next requirement is for a StringConstant component to be connected to both the GARPPresampleLayers and GARPPrediction components in order to provide pointers to the required input files. Whilst initially this step requires the user to identify this need and introduce the StringConstant components manually, over time the history of interactions built up between the system and user, or the system and a group/domain of users, will enable

the easier identification of this step as the component will be promoted to the top of the addition suggestions.

Step 10: Configure Component - StringConstant

Introducing the StringConstant components completes the structure of the workflow for this scenario, as show in Figure 10-5. The final requirement is to configure the StringConstant components to point to the relevant input files for the species distribution to function correctly. As described previously this functionality is achieved as it would be within the underlying Kepler SWS, double-clicking the component brings up a dialog into which parameter such as file locations can be specified.

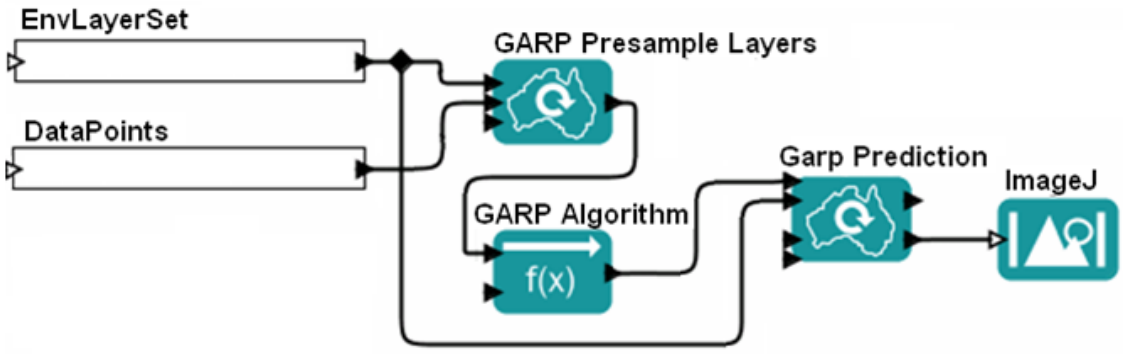


Figure 10-5 Completed Scenario C Composition

B.4 Scenario D

As discussed in the main text Scenario D provides an extension to the functionality of Scenario C, performing the same GARP analysis task but using a remote database connection to retrieve the required species distribution data. This involves essentially the same workflow components and sequencing as Scenario C, but with one of the StringConstant comonents replaced with an instance of the database component DarwinCoreDataSource.

Step 1: Inserting Base Components

As stated previously the analysis and output portions of the workflow composition required for this scenario are identical to Scenario C. As such the user can either choose to follow the same approach as was taken with Scenario C; adding and specialising a sequence of abstract "Modelling Component" instances to obtain the

three required GARP analysis components, connecting these modelling components in the suggested sequence, then following the provided suggestions to include the "ImageJ" component as a means of displaying the produced output. Or, based on their gained knowledge that this is the required configuration they can manually add and connect each of these components without the assistance of the system.

Step 2: Inserting Input Components

As with the walk-through for Scenario C at this point, with the modelling components and output components inserted and connected, the user will only be presented with a set on non-ideal suggestions for connections and additions. The difference in this instance is that, having connected the components previously, the system would now highlight more clearly that introducing "StringConstant" components for connection with "GARPPresampleLayers" and "GARPPrediction" would be a sensible step to perform.

However, knowing that the scenario calls for input to be provided from a remote source a sensible step for the user to perform at this stage is to return to the list of abstract components provided by the system:

- Visualisation Component
- Database Component
- Operation Component
- Modelling Component
- ...

From this selection the user opts to include an instance of Database Component. Based on introducing this abstract component into the composition the user would now be presented with suggestion for how to specialise this component. The first suggestion provided by the system is to specialise "Database Component" to " DarwinCoreDataSource", this option is presented above others as the Domain metadata for "DarwinCoreDataSource" matches that of the existing GARP components within the workflow, making this a more desired choice than other database components.

After introducing the "DarwinCoreDataSource" component the user would be provided with suggestions as to how this could be connected to the rest of the composition. However, as the output from this component "DataTable" is fairly generic, in that it takes the form of whatever output is produced by the database consulted, the system will only produce weak suggestions based on the BasicPortType of "string". Table X lists the suggestions for connections involving DarwinCoreDataSource that would be provided at this stage.

Suggestions for components to connect within workflow:
• DarwinCoreDataSource.DataTable and GarpPresampleLayers.layerSetFilename
• DarwinCoreDataSource.DataTable and GarpPresampleLayers.dataPointFilename
• DarwinCoreDataSource.DataTable and GarpAlgorithm.ruleSetFilename
• DarwinCoreDataSource.DataTable and GarpPrediction.layerSetFilename
• DarwinCoreDataSource.DataTable and GarpPrediction.outputASCII
• DarwinCoreDataSource.DataTable and GarpPrediction.outputBMP

Table 10-11 Suggestions for components to connect with DarwinCoreDataSource

Given the knowledge gained from composing Scenario C the user would be able to discard several of these options, for example the "ruleSetFilename" port of GarpAlgorithm, and the "outputBMP" and "outputASCII" ports of GarpPrediction were never connected during scenario C so it would be reasonable to assume that they are also not required here. Similarly the user could recall that the StringConstant component from Scenario C which was responsible for the same activity as DarwinCoreDataSource is intended to be used here was connected to the GarpPresampleLayers port "dataPointFilename", therefore they would identify that it is to this port that the database component should also be connected.

At this point the user is required to begin making deductions of their own regarding the steps to take, as the system does not have enough static metadata to identify and highlight the steps to take, nor sufficient knowledge about how these components have been used previously to assist this process. Whilst it is reasonable to assume that the user would be able to overcome this difficulty between the system suggesting that

StringConstant should be used to provide input to GarpPrediction and GarpPresampleLayers as with Scenario C, the users own experience from having successfully composed that Scenario previously, and the limited options which remain for how to connect the available components, it is still situation where the current approach would fail to offer much assistance.

Again, as with Scenario C, having successfully completed this scenario the system would gain knowledge of how these components interact and how the user has deployed them previously, therefore during any future compositions utilising these components it would be able to offer greater benefit.

Appendix C - Scenario C Kepler Composition Walk-through

This appendix provides a walkthrough of the steps which a user will go through to compose Scenario C from Section 8.2.1 using the Kepler SWS. The purpose of this walk-through is to illustrate the difficulties and problems which the user might encounter as a result of the manual composition approach provided by the Kepler.

C.1 Scenario C Kepler SWS Composition

Kepler Manual Composition – Scenario C Species Distribution Modelling

As described previously this scenario from the domain of bioinformatics involves the modelling of species distribution based on an input of relevant data relating to species locations and environmental factors. The result of this modelling is then displayed graphically.

Step 1: Identifying Components

The Kepler GUI initially presents the user with two main elements – a blank canvas on which to compose the workflow scenario, and a tree containing the components available for inclusion within the workflow. The first process a user must perform is to identify which components are required for this scenario from this tree. From examining the overall goals of Scenario C the user is able to identify a variety of tasks that will need to be performed in order to successfully complete this scenario. These tasks include:

- Accessing data files
- Performing calculations with the relevant modelling algorithms
- Producing a graphical output of the result

Problem A: At this stage the user may not be aware of which components within the Kepler component tree are suitable for achieving these tasks. An experienced user who has composed similar scenarios previously may know which components provide the functionality required for these tasks, however the system cannot assume that every user possesses this knowledge.

Kepler Solution - Searching Facility: To assist users in identifying the necessary components for a composition Kepler provides a search facility, this provides users with the option to search the list of components based on their given names. As identified in the walkthrough for Scenario B this approach to locating components can be successful, providing the name of a component is a useful description of the task it achieves. Unfortunately this is of limited benefit in locating components to perform the above tasks identified for this scenario.

Searching the list of components for terms from our task list such as “Distribution” and “Calculation”, in order to locate a component to perform the required distribution modelling, returns no helpful results. Searching for the term “Modelling” is able to return a selection of components which includes a number of those required for this scenario, however as Figure 10-6 shows there are also other components listed which are not required for this situation, so the user is still required to know which specific component or components from those provided is the correct choice.

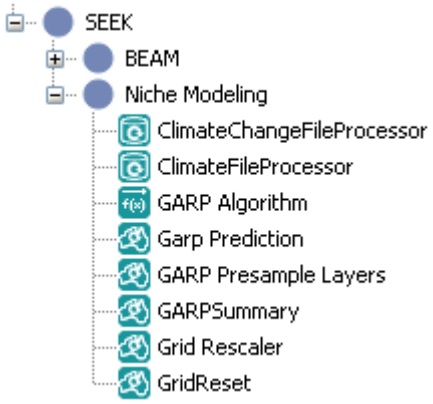


Figure 10-6 Results of the search term "Modelling" in the Kepler SWS.

Problem B: The user may not know the specific name within Kepler for each of the components required for the composition, and searching for generic task related terms may not return the desired results

Kepler Solution – Structured Component Listing: If the user is unaware of the names of the components required for their composition, and is unable to locate these

components through searching, Kepler provides a further mechanism to help locate those components required. The tree of available components is structured in such a way as to group components by a selection of categories. The categories provided are as follows:

- Components
- Projects
- Disciplines
- Statistics

The Kepler system utilises this categorisation in order to direct users to the appropriate section of components within the tree in an attempt to make it possible to identify the components a user requires. This approach again relies on the user having a degree of knowledge about where to start looking within these categories. For example if a user knows that the distribution modelling components required are provided as part of the SEEK project, then by expanding the projects category they can quickly identify those components required.

Problem C: However if the user is unaware of the project or discipline to which their desired components belong then this mechanism is of limited benefit in assisting such a user to identify their desired components. Furthermore the manner in which the components are divided into these categories is somewhat unhelpful – whilst a user may not know the project to which the distribution modelling components belong they may be aware of the possible disciplines in which they are used, bioinformatics, ecology etc. However exploring the Disciplines category shows no entries for the required components under the branch “Ecology”. This illustrates how the components are only listed once within the categorisation, overlooking the possibility that a component may be used in more than one domain or project, and potentially leading to a situation where, having explored the relevant branch of the component tree the user wrongly assumes that their desired component does not exist because it has only been listed under another branch they were unaware of.

Kepler Solution – Component Documentation: Failing to possess the level of knowledge to identify the required components a user must fall back on a process of manually exploring the complete list of components available. Locating the correct

component in this instance relies on an element of the component's location within the tree providing some link to the task that the user wishes to perform, or failing this Kepler provides documentation for each component which the user can inspect. This documentation comprises an in-depth description of what each component is used for and provides a list of the various properties and input or output ports associated with each component. This documentation could help in identifying the correct components, although requiring users to inspect the documentation of each component until they locate a potential candidate for inclusion within the workflow is a cumbersome process.

Problem D – Documentation Consistency

Identifying which components to utilise for composing a scenario when using the Kepler SWS relies heavily on inspecting the documentation available to identify the tasks performed by each component. However as described in Chapter 4 the benefit of this process is reduced by the lack of consistency in the availability, detail and diversity of the information provided within each components documentation. Some components provide clear and concise descriptions of their function, others contain only limited detail. Some components contain in-depth information regarding the information sent or received through their input and output ports, others contain no information regarding ports. There are also a number of components which lack any documentation whatsoever.

Assuming that a user is able to overcome the problems in identifying the components to perform the distribution modelling required for Scenario C, Figure 10-7 shows a composition that could be achieved containing the three GARP components that will perform this task.

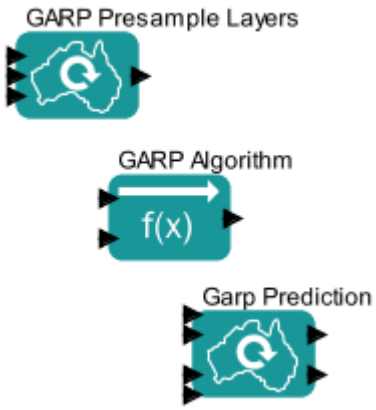


Figure 10-7 Modelling Components Identified for Scenario C

Now that the user has identified the modelling components required for Scenario C they can utilise the available facilities to identify other components required to complete this scenario. As described in the overview of the scenario the user must specify some input data on which to perform the modelling operation, along with providing some graphical means of displaying the result. Attempting to locate the correct component to provide input for the distribution modelling introduces another potential difficulty with the existing approach provided by the Kepler SWS.

Problem E: As described previously the user's first option in locating components from those provided by the system is to search using keywords representing the task they wish to perform. In this instance searching for terms such as "File" or "Input" will present the user with a large list of possible components. Furthermore this list includes a number of components whose name suggests they are capable of providing the functionality required when they are actually incompatible with the input the modelling components require. For example a search for "File" will provide the results:

- FileReader
- FileFetcher
- SimpleFileReader

Whilst each of these component's names and descriptions state they are used for the process of including files within a workflow composition they each provide output of a type that is incompatible with this scenario. In this way using the search facility can

deliver misleading results and make it more difficult for the user to locate components to provide their required functionality.

Kepler Solution – Inspect Documentation: In order to locate the correct input for the modelling components the user's only option is to return to the documentation for those components already included in the workflow, inspecting the descriptions for their input ports and noting the type and format of input that they accept. This knowledge of the input which an existing component accepts can then be used to locate a suitable component to provide this input. In order to achieve this, the user must again manually search the list of available components, inspecting their output types listed in the documentation and trying them within the composition, until they either locate the components required or simply give up.

Assuming the user is able to overcome this difficulty in locating the correct component to provide input for the modelling operation, in this instance the component required is the StringConstant component, the process of identifying the component to produce a graphical display of the results of the modelling operation is more straightforward. A search for "Display" produces, amongst other results, the component "ImageDisplay" the description of which states that it "reads an image token and displays the image on the screen". Inspecting this component amongst the results produced it is relatively easy for the user to deduce that ImageDisplay is a suitable component to produce output for the scenario.

Identifying Components Summary

In this instance only a user who already knows which specific components are required for completing Scenario C will be able to readily locate those components within the Kepler SWS. If a user has some knowledge of the components they require, such as the domain or project in which they are used, then they *may* be able to locate the components they require, however this is still a limited process that can result in users failing to discover the components they require.

Step 2: Connecting Components

After including the ImageDisplay component the user has achieved a workflow composition which contains components to provide the majority of the functionality required by Scenario C, Figure 10-8 displays the composition at this state.

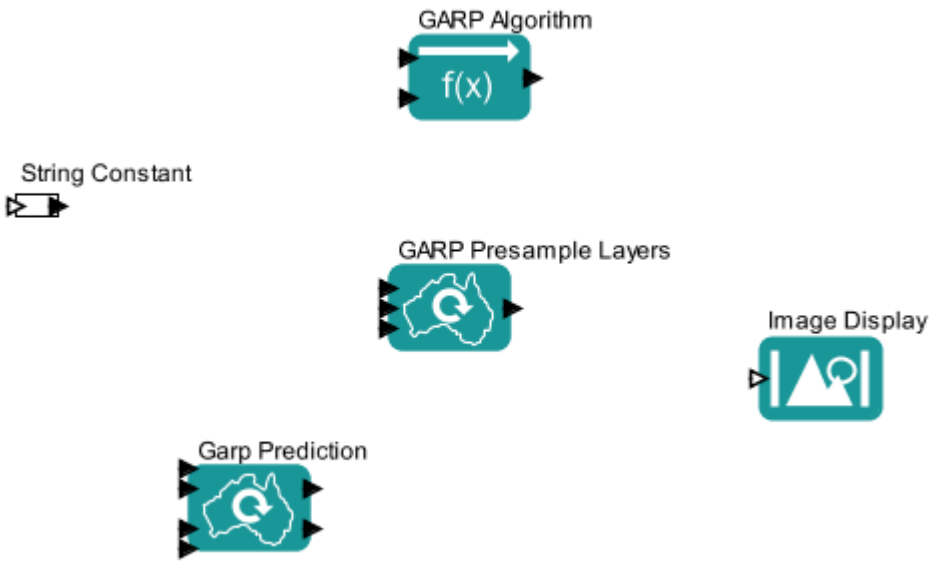


Figure 10-8 Components Identified for Scenario C

The next task the user needs to perform in order to complete this composition is to correctly sequence and connect the components now present in the workflow. Knowing that the StringConstant component acts as input for the scenario and Image Display as the output the user can assume that these represent the beginning and end points of the composition, however the challenge is in how to sequence the GARP components which represent the body of the scenario. Again if the user has experience in composing workflows using these components they may already know the manner in which the three interact and so will have little difficulty in achieving this step. However a user with little experience must discover this sequencing manually.

Problem F: Users with limited knowledge, or those working with new components, are unable to identify how to sequence and connect components in order to achieve their desired functionality.

Kepler Solution – Port Types and Documentation: In order for the user to overcome this challenge of sequencing the three GARP components Kepler provides two forms of assistance; the GUI displays the number and type of input and output ports provided by each of the components, and further information about these ports and the purposes of each component is provided within the available documentation. Figure 10-9 shows the port types of each of the three components.

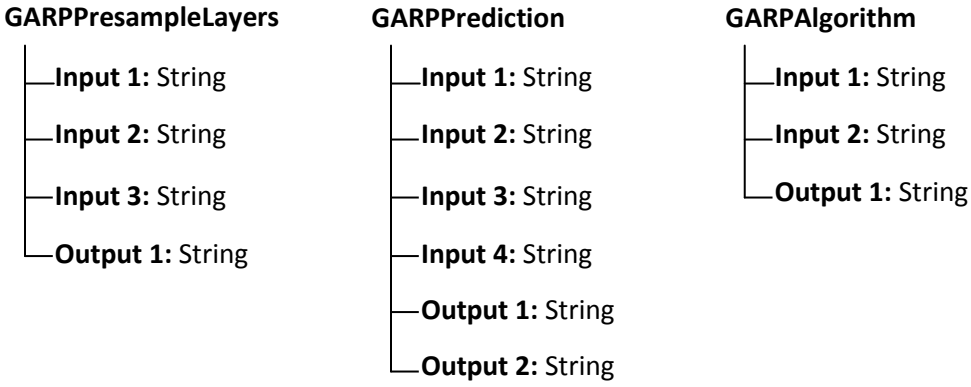


Figure 10-9 GARP Components Port Types

Each of the input and output ports of the three components accepts the type "String". This provides no assistance in identifying the sequence in which these components must be connected in order to successfully achieve the goals of the scenario. From this basic information the user could conclude that these components could be connected in any sequence. However there is only one sequence which will provide the correct output. Beyond specifying the port types each component port within Kepler has a given name which further describes the purpose of that port. Inspecting the names of the ports provided by the three components reveals the following:

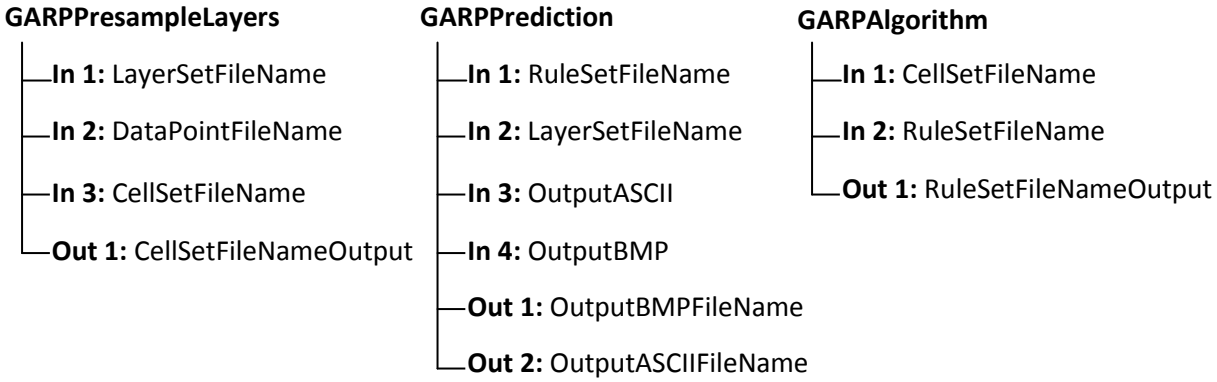


Figure 10-10 GARP Components Port Names

Using this information regarding the names of the ports provided by the three GARP components the user can begin to infer connections that could potentially be made between the components. For example the output from GARPPresampleLayers, “CellSetFileNameOutput”, could potentially be provided as input for the port “CellSetFileName” provided by the GARPAlgorithm component. Similarly the connection between the “RuleSetFileNameOuput” and “RuleSetFileName” ports of GARPAlgorithm and GARPPredicition respectively could be identified by the user. However following this logic there is a similar potential connection that could be identified between the two similarly named input and output ports of GARPAlgorithm itself.

The inclusion of the filetype “BMP” within the name of ports provided by the GARPPrediction component could also allow the user to identify this as the component which provides output for the ImageDisplay component to visualise, and therefore that this component is the last of the three in this sequence. Inspecting the documentation provided for each of these components would enable the user to confirm GARPPrediction as the output component of this sequence, however the challenge in inspecting and correctly interpreting the knowledge provided by Kepler, both within the descriptions of components and in the types and names of their ports, is not inconsiderable and does not lend itself to supporting users in easily identifying the correct sequencing of these components.

Step 3: Completing the Composition

Having utilised the available information to identify the sequencing of components, along with a number of port connections to make between those components, the user can develop the composition to the state as shown in Figure 10-11.

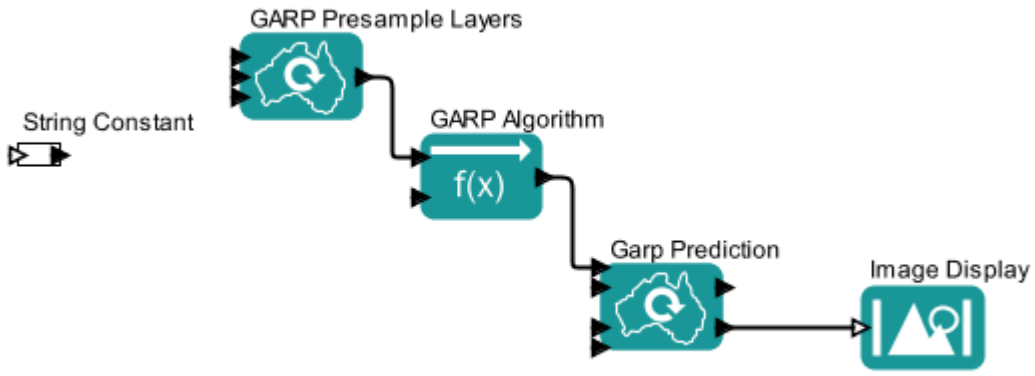


Figure 10-11 Scenario C with String Constant disconnected

At this point each of the components still has a number of unsatisfied input and output ports and the user has yet to identify how to provide the required input files for the modelling components. Again for a user who has not previously interacted with these components the only mechanism available to discover how to complete the scenario is to return to the documentation and port descriptions provided for each of the components. The port type information provided for these components is unable to assist the user as each remaining port has the type “String” and so the user could deduce that StringConstant could connect to any of the GARP components ports. Returning to the documentation for these components, the descriptions provided for the GARP components and their remaining ports indicate that the user needs to supply the locations of the species presence and environmental data files required for the distribution modelling. In order to achieve this, further StringConstant components must be inserted into the composition and correctly configured to provide “Strings” to represent the location of these files.

Summary

The walk-through presented above has highlighted a number of problems with the manual approach to workflow composition as provided by the Kepler SWS. In summary these problems are:

- **Problem A** - Identifying required components to achieve goals

- **Problem B -** Difficulty using search facility to assist
- **Problem C** - Misleading component listing hierarchy
- **Problem D** - Inconsistency of documentation
- **Problem E -** Difficulty identifying source of input
- **Problem F** - Difficulty correctly sequencing components

This walk-through has also demonstrated that whilst the Kepler SWS provides a number of facilities with which to assist the user (documentation for each component, a facility to search through the list of available components etc.) these are insufficient to overcome the problems identified. As described in Chapter 4 each of the SWSs considered win this thesis (Taverna, Triana, and Kepler) offers similar help to the user during composition, but each has comparable limitations and so is susceptible to the same problems as identified in this Appendix.

Appendix D - Calculating Quality Scores

As an illustration of the process involved in calculating suggestion quality scores the following describes how the scores are generated for Scenario A at the first key point, the inclusion of the Remainder component.

If restricted to using the Port Type element of metadata to identify suggestions the system would inspect the ports provided by Remainder to discover their type. Remainder has two ports, Input and Output, each with the basic type “Double”. The system would then search the ontology for other components possessing ports compatible with this type. This would produce the following results:

- Add
- Constant
- Display
- Divide
- Multiply
- Remainder
- Subtract

Constant and Display would be identified as compatible as their port types, “unknown” and “general” respectively, are treated as being compatible with any other type. Each of the mathematical operations would be identified as they all have either input or output ports of type Double, matching those of the Remainder component.

Based on the ideal composition of this scenario already identified there are two ideal components that should be identified by the suggestions – Display and Constant. As the use of basic port type for generating suggestions has identified both of these components this suggestions would achieve a score of 100% for mechanism A; percentage of ideal suggestions.

Scoring mechanism B looks at the percentage of suggestions provided which are incompatible with the current workflow. If using the Provider element of metadata to

generate suggestions the system would look at the developer or vendor who has provided this component for use in the SWS and provide suggestions to include other components that they provide within the workflow. As Remainder is a component originally provided by the Ptolemy system, of which Kepler is a descendant, it has a Provider of value “Ptolemy”, thus the system suggests any other “Ptolemy” components. As expected this includes a large number of the generic components provided by the system, the following are the first ten components the system suggests:

- Average
- Array Average
- Array Length
- Array Minimum
- Array Maximum
- Array Plotter
- Timed Plotter
- Maximum
- Minimum
- Bar Graph

Overall 143 components are associated with the “Ptolemy” Provider, of these there are 67 which are compatible with the Remainder components input or output ports. Thus for scoring mechanism B, percentage of incompatible suggestions, this instance receives a score of 47%

Finally scoring mechanism C evaluates the ranking of ideal components included in a set of suggestions, the goal is for the system to have the most ideal components ranked highest within any set of suggestions. Again utilising the Port Type element of metadata the system would generate the following suggestions for inclusion in the composition with the Remainder component:

- Add
- Constant
- Display

- Divide
- Multiply
- Remainder
- Subtract

With Port Type as the only information used the suggestions are simply ranked alphabetically and in this case the two ideal components for the scenario, Constant and Display, have been given high rankings of 2nd and 3rd position within the list. Unfortunately the Add component has been ranked above the two ideal components meaning the components required to complete the scenario are not the first suggestion a user is provided with for completing this scenario. Scoring here is achieved by assigning a number of points to each of the items in the list of suggestions, 1st position achieving the most points and last position the least. Points awarded are related to the number of suggestions, here there are 7 suggestions and therefore 1st position is awarded 7 points, each position down receives one less point with last position, 7th, receiving only 1 point.

The overall score is calculated as a percentage of the maximum number of points that the ideal suggestions would have gained if they were the top suggestions. Here there were 2 ideal suggestions, Constant and Display, therefore the maximum number of points they could have achieved was 13, 7 points for first position and 6 points for second. However as Constant and Display only achieved 2nd and 3rd positions respectively their actual score was 11, 6 points for first position and 5 points for 3rd. This results in an overall score for the suggestions of 85%.

Whilst this has a limited impact in this instance, where it may be assumed a user desiring to find and display the remainder of a division would be able to identify the need for both the Constant and Display components, in other situations the low ranking of ideal components may result in making the completion of a composition more challenging than it otherwise could have been.

Appendix E - Scenario Suggestion Scores

E.1 Scores for Individual Metadata Elements

Scenario A

As introduced in Section 8.2.1.1 Scenario A represents a workflow for performing a simple mathematical operation, calculating and displaying the remainder of a division operation. An ideal composition to achieve Scenario A involves the components Constant, Remainder and Display. In order to assess the quality of suggestions provided during composition of this scenario the value for each of the quality scores defined in Section 8.2.4.1 (A - Percentage of ideal suggestions, B - Percentage of incorrect suggestions, C - Ranking of ideal suggestions, D - Total number of suggestions) is calculated after the inclusion of each of the ideal components involved. The mechanism through which these scores is calculated is described in Appendix D.

Table 10-12 lists the scores across each of the three quality rankings for suggestions provided by the system in composing scenario A, when each element of metadata is used in isolation, note that no scores are provided for the Project or Domain elements of metadata as these components as they exist within the Kepler SWS do not have this metadata defined.

	Remainder				**Constant**				**Display**			
	A	B	C	D	A	B	C	D	A	B	C	D
Provider	100	47	79	141 (2)	100	55	53	141 (2)	100	100	56	141 (2)
Project												
Domain												
Port Type	100	100	85	7 (2)	100	100	53	7 (2)	100	100	62	7 (2)
Port Data Object	100	100	85	7 (2)	100	100	53	7 (2)	100	100	62	7 (2)

Table 10-12 Suggestion Scores for Scenario A, the columns A,B,C,D represent the four quality criteria defined in 8.2.4.1. The (2) entries in the D columns represents the total number of "ideal" suggestions possible at that stage, contrasted against the number of suggestions the system is providing.

These scores show that for relatively simple compositions, working with a system where relatively few components are made available, such as scenario A, where only a few components are required, the use of individual elements of metadata to generate suggestions is quite effective. In the case of the Port Type and Port Data Object metadata elements the system is able to identify both of the ideal components desired at each stage, and is relatively effective in highlighting these above other, less useful suggestions available. Other metadata elements such as Provider are less successful in providing suggestions of a high quality when used in isolation. For example although in this instance the Provider metadata was able identify both the ideal components, Constant and Display, within the suggestions it did not rank these highly within those suggestions, suggested a large proportion of components which are incompatible with the composition, and generated a large total number of suggestions thus increasing the challenge for the user to locate the ideal suggestions to implement.

Scenario B

Scenario B involves the manipulation of an image file by rotating it and displaying the result, the ideal components for this composition are Image Reader, Image Rotate and Image Display. As before the suggestions scores are calculated after the inclusion of each of these components within the composition. These scores are shown in Table 10-13.

	Image Reader				**Image Rotate**				**Image Display**			
	A	B	C	D	A	B	C	D	A	B	C	D
Provider	100	100	58	141 (2)	100	100	58	141 (2)	100	100	58	141 (2)
Project												
Domain	100	100	42	9 (2)	100	78	47	9 (2)	100	56	37	9 (2)
Port Type	100	100	58	141 (2)	100	100	58	141 (2)	100	100	58	141 (2)
Port Data Object	100	100	71	4 (2)	100	100	71	4 (2)	100	100	71	4 (2)

Table 10-13 Suggestion Scores for Scenario B

Similar to Scenario A these scores illustrate that for a relatively simple operation such as this Image Processing example the use of individual elements of metadata can

provide helpful suggestions. In this case as the components required are all provided directly by the Kepler platform and so the Provider element of metadata can be used to suggest the inclusion of the other ideal components at each stage of the composition, however as before this does also result in a large number of suggestions for unhelpful components. The scores across each of the components in this scenario are the same as each is from the same provider, contain Port Type and Port Data object metadata which is identical, and are similarly named so are ranked equally when listed alphabetically within the provided suggestions.

One difficulty identified by this scenario is that used in isolation the Port Type metadata can be misleading when identifying useful suggestions. In this each of the components ports has a Port Type of "null", essentially indicating that it accepts any type of data, resulting in all other components being identified as compatible. Situations such as this can demonstrate the value of utilising a variety of metadata in identifying suggestions.

Scenario C

Table 10-14 shows the scores for suggestions provided by each individual metadata element during composition of Scenario C, GARP niche modelling. The scores shown are what would be provided by the system after inclusion of each of the first three ideal components from this scenario - String Constant, GARPPresampleLayers, and GARPPrediction. A complete table with scores for the suggestions provided after inserting each of the components for this scenario is provided in Appendix H.

	String Constant				**GARPPresampleLayers**				**GARPAlgorithm**			
	A	B	C	D	A	B	C	D	A	B	C	D
Provider	0	0	0	143 (2)	50	100	35	9(2)	100	100	53	9(2)
Project					50	100	35	9(2)	100	100	53	9(2)
Domain					100	100	100	5(2)	100	100	78	5(2)
Port Type	100	100	21	358(2)	100	100	69	168(2)	100	100	67	168(2)
Port Data Object	100	100	66	124(2)	100	100	41	125(2)	100	100	100	2(2)

Table 10-14 Suggestion Scores for Scenario C

Whilst suggestions generated by individual elements of metadata can be sufficient for composing simple workflow scenarios, with more complex situations involving a wider array of components and connections the quality of suggestions can become reduced. Scenario C represents a composition involving a greater range of components and component interactions than those present in Scenarios A and B, as a result the quality of suggestions provided by each individual element of metadata is reduced.

For example after the inclusion of the GARPPresampleLayers component the desired outcome is for the system to suggest the inclusion of the components GARPAlgorithm and StringConstant. The Provider, Project and Domain elements of metadata are all able to identify GARPAlgorithm as a suggestion as it shares these elements with GARPPresampleLayers, additionally both Project and Domain based suggestions are able to identify this component within a very small number of suggestions, as the project “SEEK” and domain “BioInformatics” have only a limited number of components associated with them.

None of the metadata elements; Provider, Project, and Domain, is able to identify StringConstant as a component to include within the composition, the first element which identifies this component is the PortType metadata. However in this case the PortType of both components is defined as “String”, a generic type shared by many components. As a result StringConstant is suggested low down in the list of suggestions. Despite this drawback PortType is the only element of metadata which successfully identifies both StringConstant and GARPAlgorithm as components to be connected to GARPPresampleLayers.

Finally the PortDataObject metadata successfully identifies GARPAlgorithm as a component to connect as it shares the value “cellSetFile” with GARPPresampleLayers, additionally as this is a very specific value only two suggestions are provided using this element of metadata, reducing the complexity of identifying which suggestion to implement.

This scenario illustrates that there are several limitations of using only a single element of metadata to generate suggestions - it is difficult to identify all of the ideal components to include within a composition, a large number of redundant or unhelpful suggestions

can be generated, and finally even though the system may be able to identify the correct components to include within a scenario, it is unable to effectively highlight those above other less useful suggestions.

Scenario D

As discussed previously scenario D represents a similar workflow to scenario C, replacing the use of hard coded species occurrence data with input data retrieved from a remote database. As these two scenarios share many of the same components and these are connected in the same manner suggestion scores for Scenario D have not been provided as these would not be sufficiently distinct from those of Scenario C.

E.2 Scores for Combined Metadata Elements

As the previous sections have demonstrated, using a single element of metadata to generate suggestions is of limited benefit. For simple scenarios some metadata elements can produce effective results, for example the PortType and PortDataObject elements of metadata scored highly for scenario A, however this benefit was reduced significantly when the complexity of the scenario increased.

By utilising several elements of metadata in conjunction the benefit of each can be brought together to improve the quality of the final selection and ranking of components provided within suggestions.

Scenario A

Table 10-15 lists the suggestion scores that would be generated for scenario A at each of the same points as used previously when using all of the metadata elements together. These scores are compared against the average of the scores generated by each of the individual elements of metadata in isolation at the same stages.

	Remainder				**Constant**				**Display**			
	A	B	C	D	A	B	C	D	A	B	C	D
Individual Score	60	50	50	51	60	51	32	51	60	60	36	51
Combined Score	100	100	85	7	100	100	53	7	100	100	62	7
Difference	+40	+50	+35	-44	+40	+49	+21	-44	+40	+40	+26	-44

Table 10-15 Comparison of Suggestion Scores for Scenario A

As described previously the relative simplicity of this scenario means that even with a single element of metadata useful suggestions can be generated, however the improvement provided when using all elements together is still tangible. Again, as this scenario is relatively simplistic the scores at each stage are now in line with those provided by the PortType and PortDataObject metadata previously, this is due to the generic nature of the components meaning that combining these with the Provider metadata is not able to improve the score further.

Scenario B

This scenario is slightly more involved than the previous example, and in this case more metadata is available for use in generating suggestions, with the Domain metadata now being taken into account. Table 10-16 provides the comparison of scores for Scenario B

	Image Reader				**Image Rotate**				**Image Display**			
	A	B	C	D	A	B	C	D	A	B	C	D
Individual Score	80	80	46	74	80	76	47	74	80	71	45	74
Combined Score	100	100	71	4	100	100	71	4	100	100	71	4
Difference	+20	+20	+25	-70	+20	+24	+24	-70	+20	+29	+26	-70

Table 10-16 Comparison of Suggestion Scores for Scenario B

As before we see an increase in the quality of the suggestions being provided. A large portion of the benefit comes from using the metadata elements related to component *compatibility* (PortType and PortDataObject) in connection with the metadata more related to a components *suitability*.

Scenario C

This scenario introduces further complexity in comparison to the previous two, including more components and multiple connections between those components.

	String Constant				**GarpPresampleLayers**				**GarpPrediction**			
	A	B	C	D	A	B	C	D	A	B	C	D
Individual Score	40	40	17	208	80	100	56	63	100	100	70	39
Combined Score	100	100	72	124	100	100	100	5	100	100	100	5
Difference	+60	+60	+55	-84	+20	-	+44	-58	-	-	+30	-34

Table 10-17 Comparison of Suggestion Scores for Scenario C

Due to the increased complexity of this scenario we see that the quality of suggestions which can be provided by the individual elements of metadatq in isolation has been reduced, illustrating how whilst potentially useful in simple scenarios for more complex workflows it is necessary to generate suggestions using multiple metadata elements in order to ensure that useful suggestions continue to be provided.

Scenario D

As discussed in the previous section Scenario D represents a composition which is largely a repetition of Scenario C and as such suggestions scores have not been calculated for this scenario.

Appendix G - Feedback Questionnaire

Cambridge Meeting 27/07/09 Feedback Questionnaire

Current Prototype
These questions relate to the prototype in its current state.

1. Were you able to successfully compose a workflow with the prototype?

Partially. The system didn't have many "pieces" in it, and my knowledge of how to use those pieces was limited. This meant I needed some help to do much.

a. If not what was it that prevented you from achieving this?

b. And can you suggest a solution to the problem?

More documentation on how to use the available pieces would help. They were quite specialized – to do with the GARP ecological niche model – and it'd been a long time since I'd looked at them. The other thing that needs to happen is the hard work of creating more components...

2. What did you think was the best element of the current prototype and why?

The suggestion scheme, though obviously limited because the number of components in the system is small, looks very promising.

3. What did you think was the worst element of the current prototype and why?

I find the basic workflow UI pretty clumsy. Overall, I'm not convinced that visual programming of workflows is easier than simple scripting. But that's probably a matter of taste – after all, I am a programmer.

4. Do you think a suggestion based approach to workflow composition is beneficial and why?

I think the suggestion-based approach as great potential, whether the underlying workflow language is implemented visually or as a scripting language. Of course it would benefit from richer semantic description of the components and links than currently exist, but it is an interesting start.

5. Do you think the ranking of suggestions was effective?

Yes

a. Was it easy to identify which suggestions were the most suitable?

Yes – but the number of choices was fairly small. The current approach might need to be made smarter if it is to scale to systems containing large numbers of components

b. Was it easy to find the suggestion which you required to continue composing your workflow?

Yes, but again the number of choices was small.

6. Were there situations where a step you desired to make was not available in the suggestions?

No – but again the number of components was small (same comment applies to 7 and 8 below)

7. Were there situations where you believed such a suggestion should have been ranked higher / more obviously identifiable?

No

8. Did you find the filters useful in assisting to identify required steps within the suggestions?

Yes

a. Are there any other filters or mechanisms that could have made this easier?

9. How well do you think this approach to composition will scale?

a. If used with a wider array of components?

The ranking might need to be more sophisticated

b. If used to compose more complicated workflows?

Seems like the goal of the suggestions is “local” – just what to link up next. This approach should scale fine, and anything more, a global analysis of what the workflow is trying to do, would probably be prohibitively complex.

10. How beneficial do you think that utilising data regarding previous component usage to influence future suggestions can be?

Seems like a good idea if enough users are willing to contribute.

a. Do you think such information would be of more benefit if various "levels" were stored; component usage by individual users, within a given domain or within a specific project?

All good ideas, but probably mainly relevant as "second order" approaches after an initial, non-hierarchical system is implemented

i. Do you think that given sufficient data about the relationships between given users or domains that usage data taken from one user or domain could be used to assist another user or domain?

Yes – after all, I think most people learn to write code by example – looking at other's code, modifying existing code etc. This would be a way to extract useful information from a lot of examples. Once you start sharing info between users, you'll need to deal with quality and trust issues – after all, I wouldn't want suggestions to be highly influenced by information from an inexperienced user building workflows in odd and not very useful ways.

11. How beneficial do you think that identifying both the compatibility and desirability of composing workflow components is to a suggestion based composition approach?

Probably beneficial, but also probably one of those things that'll have to be tried out to find out which approaches are most useful

12. How helpful do you think that utilising an ontology to store further information about the data passed between components is in identifying their suitability to be composed?

For some components, the amount of semantic description needed is small – for example many statistical techniques can be applied to data from a wide range of domains. Other analyses are very specific to the data semantics and so in those cases, semantic description would be very important.

Future Developments

These questions relate to possible future directions to be explored within this approach.

1. **Would storing information about ideal input parameters for a component, those which produce useful results, be of use in identifying other components to provide this input?**

Hmmm, not sure what "ideal input parameters" means. But I can imagine models where there are ranges of parameter values, or relationships between different parameters, that need to be met.

   a. **Do you think that enabling the system to "dry run" components or sections of workflow, identifying properties of the output produced, combined with the further information about properties of ideal component inputs would enable the system to better identify which connections would be desirable?**

      Seems useful. And propagating semantic information through the workflow during a dry run will probably be important

2. **How effective would allowing users to edit information within the ontology about component inputs and outputs be in improving suggestions provided?**

I don't know enough about the ontology structure, but I can see that allowing users to enter information about components could be useful. In my experience, tools for interacting with ontologies and instance data are a mess, so as with most of this stuff, its success will likely hinge on producing a well designed UI. Probably one where users don't even realize they're interacting with an ontology.

3. **Do you think that allowing users to assign ratings to the suggestions provided by the system, and incorporating those ratings into the ranking of future suggestions, would improve suggestions?**

Seems like a useful way to provide feedback. Of course you'd then need a way to rank the quality of the user information.

   a. **Again would it be useful if such information was recorded relative to a users domain, so as to only affect future suggestions for other members of that domain?**

      Seems like a good filtering option.

4. **Would the prototype have benefitted from a mechanism to explain the reasoning behind the suggestions provided?**

Hard for me to say – you'd already described a lot of the methodology to me. I generally like to know how things work, but I'm an engineer...

5. **Would your interactions with the prototype have been improved if you had been able to inspect the ontology manually and why would this have been helpful?**

Not sure – but for the average user, I hope this isn't necessary!

Appendix H - Complete Scenario C Suggestion Score Tables

Scenario C Suggestion Scores

	String Constant				GARPPresampleLayers				GARPAlgorithm				GARPPrediction				ImageJ			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
Provider				143 (2)	50	100	35	9(2)	100	100	53	9(2)	33	100	25	9(3)				(1)
Project					50	100	35	9(2)	100	100	53	9(2)	33	100	25	9(3)				(1)
Domain					100	100	100	5(2)	100	100	78	5(2)	66	100	47	5(3)				(1)
Port Type	100	100	21	358 (2)	100	100	69	168(2)	100	100	67	168(2)	100	100	44	168(3)	100	100	63	285 (1)
Port Data Object	100	100	66	124(2)	100	100	41	125(2)	100	100	100	2(2)	100	100	39	152(3)	100	100	63	285 (1)

Table 10-18 Complete Suggestions Scores for Scenario C

Scenario C Suggestion Scores Single vs. Multiple Metadata

	String Constant				GARPPresampleLayers				GARPPrediction				GARPAlgorithm				ImageJ			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
Individual Average	40	40	17	208	80	100	56	63	100	100	70	39	66	100	36	69	40	40	63	285
Combined	100	100	72	124	100	100	100	5	100	100	100	5	100	100	100	5	100	100	63	285
Difference	+60	+60	+55	-84	+20	-	+44	-58	-	-	+30	-34	+34	-	+64	-64	+60	+60	-	-

Table 10-19 Complete Comparison of Suggestion Scores for Scenario C