Scientific Workflow Patterns

A dissertation submitted to the University of Manchester for the degree of Master of Advanced Computer Science

September 2012

Bamdad Dashtban

School of Computer Science
Contents

List of Figures and Tables .................................................................................................................. 7

Tables: ................................................................................................................................................. 7

Abstract ................................................................................................................................................. 9

DECLARATION ....................................................................................................................................... 10

INTELLECTUAL PROPERTY STATEMENT ....................................................................................... 11

Acknowledgement ............................................................................................................................... 12

Chapter 1 : Introduction ...................................................................................................................... 13

1.1 Workflow ....................................................................................................................................... 13

1.2 Business vs. Scientific Workflows ............................................................................................... 15

1.3 Aims and Objectives .................................................................................................................... 19

1.4 Report Overview .......................................................................................................................... 19

Chapter 2 : Background Research .................................................................................................... 21

2.1 Scientific Workflows .................................................................................................................... 21

2.2 Characteristics of scientific workflows ....................................................................................... 22

2.2.1 Abstract vs. concrete workflows ............................................................................................ 25

2.2.2 Scientific Workflows Reuse .................................................................................................. 26

2.2.3 Scientific Workflow Repository ............................................................................................. 28

2.2.4 Dataflow-based vs. Control flow-based Workflows .............................................................. 32

2.2.5 Scientific Workflow Management Systems ............................................................................ 34

2.2.6 The Taverna Workflow Management System ....................................................................... 36

2.2.7 Other Workflow Management Systems .................................................................................. 42
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3</td>
<td>Scientific Workflow Patterns</td>
<td>44</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Abstract vs. Concrete Workflow Patterns</td>
<td>46</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Scientific Workflows Control Patterns</td>
<td>46</td>
</tr>
<tr>
<td>2.3.3</td>
<td>Scientific Workflows Data Patterns</td>
<td>49</td>
</tr>
<tr>
<td>2.3.4</td>
<td>Workflow Templates</td>
<td>51</td>
</tr>
<tr>
<td>2.4</td>
<td>Related Works</td>
<td>55</td>
</tr>
<tr>
<td>3.1</td>
<td>An approach for investigating patterns</td>
<td>57</td>
</tr>
<tr>
<td>3.2</td>
<td>Pattern Acquisition</td>
<td>58</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Catalogue of dataflow-based patterns</td>
<td>58</td>
</tr>
<tr>
<td>3.2.2</td>
<td>An example of workflow data pattern</td>
<td>58</td>
</tr>
<tr>
<td>3.2.3</td>
<td>myExperiment repository as a resource of study</td>
<td>59</td>
</tr>
<tr>
<td>3.2.4</td>
<td>SPARQL Queries</td>
<td>63</td>
</tr>
<tr>
<td>3.3</td>
<td>myExperiment Scraper</td>
<td>65</td>
</tr>
<tr>
<td>3.4</td>
<td>Summary</td>
<td>68</td>
</tr>
<tr>
<td>4.1</td>
<td>Catalogue of data patterns analysis</td>
<td>70</td>
</tr>
<tr>
<td>4.2</td>
<td>Data cleaning</td>
<td>72</td>
</tr>
<tr>
<td>4.3</td>
<td>Result of the analysis</td>
<td>74</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Lack of templates and components in Taverna</td>
<td>79</td>
</tr>
<tr>
<td>5.1</td>
<td>Extension of Taverna Workbench</td>
<td>80</td>
</tr>
</tbody>
</table>
5.2 Shared Variable Manager Plugin for Taverna Workbench.................................80
  5.2.1 Scoping Variables ...........................................................................................82
  5.2.2 Dataflow Unique ID.........................................................................................82
  5.2.3 Defining a variable.........................................................................................83
  5.2.4 Variable Manager Services ...........................................................................85
5.3 Development of the tools..................................................................................87
  5.3.1 Development of the Shared Variable Manager .............................................87
  5.3.2 Development of Workflow Components ....................................................88
  5.3.3 Development of Workflow Templates .........................................................89
5.4 Development of Patterns ..................................................................................93
  5.4.1 Implementing the pattern using a subworkflow ...........................................94
  5.4.2 Implementing the pattern using the Shared Variable Manager ....................96
  5.4.3 Implementing the pattern as a workflow components .................................98
  5.4.4 Implementing the pattern as a workflow templates .....................................101

Chapter 6 Evaluation of the approaches.................................................................103
  6.1 Evaluations of the first 3 methods.................................................................103
  6.2 Evaluation of the template method ................................................................105
  6.3 Comparison of the methods...........................................................................106
  6.4 Summary .......................................................................................................107

Chapter 7 Conclusion and Future Works .................................................................108
  Future Works ....................................................................................................109
    1. Using objects instead of strings for Shared Variable Manager .......................109
2. Template Designer .................................................................109
3. Better component replacement solution .....................................110
4. Ability to define colours and icons for the components ..............110
5. Usage of Templates ..................................................................110
6. User Evaluation ........................................................................111
7. Refactoring feature in Taverna Workflow Management System .........111

Appendix ......................................................................................112

7.1 SPARQL Queries ......................................................................112

7.1.1 Workflow versions with their URL ..........................................112
7.1.2 Workflows added after September 2009 .................................112
7.1.3 List of all data-flows .............................................................113
7.1.4 List of dataflow processors ....................................................113
7.1.5 Latest versions of workflows ................................................113
7.1.6 List of current versions workflows with their data flows ..........114
7.1.7 List of all processors with their types ......................................114
7.1.8 List of WSDL processors .......................................................115
7.1.9 List of subworkflows ............................................................115
7.1.10 Service information of a Dataflow .......................................115
7.1.11 Dataflow processors and theirs types ...................................116
7.1.12 Processors of a defined workflow ........................................117
7.1.13 List of a component inputs ..................................................117
7.1.14 List of a defined workflow’s data links .................................118
7.1.15 Processors and their type with links for a defined workflow

7.1.16 Processors and their type with links for a defined workflow with the WSDL info

7.2 Example myExperiment RDF Data

The final word Count: 24,642 words
List of Figures and Tables

Tables:

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Workflow Processors Types</td>
<td>40</td>
</tr>
<tr>
<td>Table 2</td>
<td>Execution time of the Copy In/Out Pattern with different Implementations</td>
<td>104</td>
</tr>
<tr>
<td>Table 3</td>
<td>Comparison of Pattern Implementation Methods</td>
<td>107</td>
</tr>
</tbody>
</table>

Figures:

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>A workflow in Taverna Workbench</td>
<td>17</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Task-to-Task Data Pattern (From: <a href="http://www.workflowpatterns.com">www.workflowpatterns.com</a>)</td>
<td>18</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Executable workflow (left) and abstract workflow (right) for the Protein Structure comparison portion of the Dragomen.</td>
<td>26</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Workflows on myExperiment</td>
<td>29</td>
</tr>
<tr>
<td>Figure 5</td>
<td>CrowdLabs workflow repository</td>
<td>30</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Chemistry component collection on Pipeline Pilot</td>
<td>31</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Shared variables (left) and data tokens (right) workflows</td>
<td>33</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Example of a scientific workflow which search for the mouse gene pathways in a gene database</td>
<td>38</td>
</tr>
<tr>
<td>Figure 9</td>
<td>A simple workflow with a Beanshell in different views</td>
<td>41</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Example of an abstract workflow from myGrid wiki website</td>
<td>54</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Structure of investigation</td>
<td>57</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Scope data pattern</td>
<td>59</td>
</tr>
<tr>
<td>Figure 13</td>
<td>myExperiment contribution and components ontology with their classes</td>
<td>64</td>
</tr>
<tr>
<td>Figure 14</td>
<td>An SPARQL Query for myExperiment which lists all the workflows, workflow versions and dataflows</td>
<td>65</td>
</tr>
<tr>
<td>Figure 15</td>
<td>An SPARQL Query for myExperiment which information about the first dataflow of the workflow number 16 in myExperiment.</td>
<td>67</td>
</tr>
<tr>
<td>Figure 16</td>
<td>How myExperiment Scraper Processes</td>
<td>68</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Comparison of defining and using variables in Patterns Catalogue and Taverna</td>
<td>71</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Google refine user interface with the data on the right and text facets on the left panel</td>
<td>73</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Google refine RDF extension</td>
<td>73</td>
</tr>
</tbody>
</table>
Abstract

A Scientific Workflow represents a multi-step experimental process, protocol, or methodology. They are used to encode and run repetitively executed scientific data and analytical software pipelines. Scientific Workflows are created from chaining together private, in house or public, third party services to satisfy the scientists’ requirements for process automation. Many of these requirements occur frequently, and can be modeled using patterns. The lack of information about the patterns in the scientific workflows reduces the chances of reusing scientific workflow elements. In this dissertation, we examine methods for the acquisition of patterns, analyze the acquired patterns, and investigate with the possibility of implementation the patterns in Taverna Workflow Management System.
DECLARATION

No portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.
INTELLECTUAL PROPERTY STATEMENT

i. The author of this dissertation (including any appendices and/or schedules to this dissertation) owns certain copyright or related rights in it (the “Copyright”) and he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this dissertation, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has entered into. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trademarks and other intellectual property (the “Intellectual Property”) and any reproductions of copyright works in the dissertation, for example graphs and tables (“Reproductions”), which may be described in this dissertation, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialization of this dissertation, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see http://documents.manchester.ac.uk/display.aspx?DocID=487), in any relevant Dissertation restriction declarations deposited in the University Library, The University Library’s regulations(see http://www.manchester.ac.uk/library/aboutus/regulations) and in The University’s Guidance for the Presentation of Dissertations.
Acknowledgement

I would like to thank my supervisor Prof. Carole Goble for her great supervision, guidance and wonderful inspiration. I’m thankful of Dr. Khalid Belhajjame for his generous help. Thanks to myGrid team for producing open source tools and supporting me in this project. I’m thankful of my family for their continuous supports in my studies.
Chapter 1: Introduction

A wide range of changes has happened in the last two decades in the field of science and engineering. Specially, computation has changed a great deal in theory and experiment. Computer applications have been used to simulate large and complex systems and they have embraced sophisticated data study and data representation. Moving information to computational infrastructure with the aim of analysis or simulation and gathering the result in a human understandable way become common place. In addition, business companies have been faced difficulties with automating their tasks and the computer industry has started to develop software to help them.

These software solutions have included multiple steps, which each step has been a computational unit or a business process helping people to solve their problems by moving from one processed step to the next one automatically. The outcome of this solution has been a group of automation software, which has produced a sequence of connected computational units, and they have been called workflows.

Three fundamental terms in the field of workflows according to [1] have been defined.

- *Workflow orchestration* refers to the process of specifying a series of tasks required to administer a business or experience in a scientific way or engineering.
- *Workflow* is defined as an environment for that orchestration.
- *Workflow instance* is referred to a sample of a workflow for a special problem in business or science.

1.1 Workflow

Initial workflows in business and science were designed and developed by complicated languages and scripts that were employed to bring the input information into the computer, schedule and execute the processing tasks, and then pass the generated results to
somewhere to show or store. Commonly those programs included a considerable pre-
processing step to change the data into an appropriate format for the computational tools
and a post processing step to gather them into a human understanding form. Using scripts
in analysis becomes more difficult as the problem size grows and the number of problems
increases. Furthermore, experiments and development of software become sophisticated
when computational resources are distributed over a network. Managing computers over a
network to coordinate with each other, run different tasks, gather them and finally generate
a result is not a trivial process, which can be handled by simple scripts. This problem
requires knowledge and technology of computing using distributed systems to eliminate
obstacles which often happens in those systems such as fault tolerance and recovery,
synchronization between remote concurrent tasks, advanced logging, and managing remote
information. To deal with these problems, workflow systems have been created to help
evolving simple scripting programs into systems built around remote procedure calls,
distributed object technology, distributed file systems and databases. These approaches to
distributed systems have now evolved into Grid [2] technology and Web-service-oriented
architectures [3], which have been, used in some workflow management systems as an
infrastructure of network and computation.

Workflows are usually represented as graphs specially directed graphs, with nodes, which
represent computational units and connections, which represent the dependency between
them. Workflow designers can use graphical programs to design and develop workflow
graphs to minimize the labor of programing. Those programs provide visual design
facilities for a workflow designer and with the mixture of some other features software
packages called Workflow Management Systems (WfMS) are developed. Because of the
differences between science and business, WfMS with special features and capabilities are
developed only to be used in scientific areas. They are called Scientific Workflow
Management Systems (SWfMS).
Experiences of using SWfMS in [4] reveals two fundamental issues. First, designing scientific workflows using a SWfMS is complex enough to stop non computer-savvy researchers using this software. Secondly, a designed and developed scientific workflow is a valuable piece of knowledge. Therefore, the rational next step after designing workflows is sharing with others, which introduces collaboration in research and reuse in workflows that might have an enormous effect on the scientific research. The consequence of these requirements is innovation of online scientific workflow repositories discussed later in this thesis. Recurring design tasks have been observed [4-8] in those repositories, which suggested us a pattern-based approach in the reuse of scientific workflows.

In this dissertation we have focused on Taverna Workflow Management System [9] as a SWfMS and myExperiment [10] as a repository of workflows.

1.2 Business vs. Scientific Workflows

There are several differences between scientific and business workflows. Business workflows are designed with the aim of automating the whole or part of business process [1]. According to the Workflow Management Coalition (WfMC) definition of workflow, the important concepts in the business workflows are task, how tasks are assigned to a participant and what the procedural rules are over tasks. The primary goal in the business workflows is to execute the process in an efficient way. This automation is designed to increase the benefit and income of a business. Enterprises and corporates deal with the business workflows. We usually cannot call business workflows fully automated because they are dependent on the actions of people. Once a business workflow is designed, many workflow users use it regularly. Controlling when, how and where activities or tasks are taking place is the crucial role of business workflows. Business workflows tend to be control-flow oriented; this means that task enablement is determined by the relative
temporal ordering imposed on tasks. Business workflow instances are usually executed independently and usually there are large numbers of instances executing at a same time working on different cases.

Scientific Workflows are developed to answer research questions and enable scientists to design computerized scientific experiments using computers. Usually they are designed by a scientist or a group of scientists to compute a research question answer, and they are often more automated. However, still there might be some interaction with the user in scientific workflows. In contrast with business workflows, scientific workflows are less regularly used after creation, they are designed for a particular experiment and when scientists get their answer they may not use it again, and they may switch to another problem. According to [11] the modeling aspect of scientific workflows is getting more attention rather than implementation of a workflow in business. Also, in the business workflows we might be able to predict the outcome of workflow before execute them but predicting the result of scientific workflows is hard due to the experimental nature of the domain. Scientific workflows are not as independent as business workflows, and there are number dependent instances of workflows, which are invoked, and there are connections between them. The way that the workflows executed is also different. Scientific workflows are usually dataflow oriented; i.e. the enablement of workflow tasks is determined by data availability. Because scientific workflows are more data oriented, they can be executed concurrently.
Figure 1: A workflow in Taverna Workbench

Figure 1 shows Taverna Workbench with an example workflow in it. Taverna Workbench in this Figure is in the design perspective. A workflow designer can compose a workflow using the services provided in the services panel on the top left side of the Taverna Workbench. The bottom left side of the Workbench is showing the details of the opened workflow such as the inputs, outputs, service and data links. The right side of the Taverna Workbench in the design perspective is the place where the graphical presentation of a workflow being presented.

The workflow in Figure 1 has one input, three services and two outputs. It can be executed by clicking the play button in the top toolbar or by clicking on the file menu and selecting the “Run workflow…” item which make the Workbench to switch to the “Results” perspective. If the workflow has an input a dialog window will be opened, asking about the input values. Otherwise Taverna executes the workflow immediately.

Workflows are created using the composition of different services. The definition of the data visibility, data interaction, data transfer and data routing in the workflows can be different depending on the aim of the workflow and the design of the workflow designer.
However all the above information can be captured in the way services composed in a workflow. The common way of defining data visibility, interaction, transfer and routing in the workflow is called **workflow patterns**.

For instance, Figure 1 shows the “Task to Task with distinct data channels” data interaction pattern from [12]. In this pattern, data is passed between services via explicit data links, which are presented as arrows in the Taverna workflows. The service A in Figure 1 is connected to the services B and C with distinct data links and this is one the common data patterns which are occurring in workflows.

In Figure 2 we can see the *Task-to-Task* data patterns with three different methods of implementation. The workflow in Figure 1 used the “Distinct Control and Data Channels” method of implementation and passed each variable via a separate data link.
In summary, two topics are covered in this section: First, an introduction to the workflows and fundamental terminologies used in that area are presented. Second, the differences between the scientific and business workflows, which necessitate the specialization of workflow management systems only for scientific tasks, are described.

1.3 **Aims and Objectives**

The project aims to identify the recurring patterns in the scientific workflows, and to propose means for facilitating the design of workflow using patterns, as well to promote the reuse of workflows. We have intended to address the above aims by tackling the following objectives:

- To identify scientific workflow patterns in the relevant literature.
- To survey and investigate the methods have been used to identify scientific workflow patterns.
- Design a method for the identification of scientific workflow pattern.
- Create a list of scientific workflow patterns.
- Encode scientific workflow patterns using Taverna scientific workflow management system.
- Examine ways of scientific workflow reuse through encoded workflow patterns.
- Evaluate the patterns reusability.

1.4 **Report Overview**

This rest of this thesis has been organized as follows:

- **Chapter 2: Background Research** gives the fundamental terminology and surveys the previous works in the field of workflows and scientific workflows,
discusses the differences between scientific and business workflows; and reviews the scientific workflow patterns in the literature. In this chapter we present Taverna, which will be our implementation platform.

- **Chapter 3: An approach for investigating and acquiring patterns** discusses an approach we have designed for investigating data-flow patterns, and presents the resources, techniques and tools we have used in this thesis for acquiring the list of patterns.

- **Chapter 4: Pattern Analysis**: We have analysed our results in the pattern acquisition part in two different ways. This chapter has covered how we have analysed the patterns and what the results have been.

- **Chapter 5: Pattern Implementation** presents how we have extended Taverna Workbench to produce the tools, which has been required for the development of patterns. Secondary we present four different methods of implementing patterns in Taverna Workflow Management System;

- **Chapter 6: Evaluation of the methods** presents two methods of evaluation for our solutions and compares and evaluates those methods.

- **Chapter 7: Conclusion and Future works** concludes the work and suggests future works to improve the results of this dissertation.

- **Appendix**
Chapter 2: Background Research

2.1 Scientific Workflows

Scientific research and business processes are the place which workflows have been utilized for a while. The workflows have been categorized differently from various approaches.

A great number of workflows have been used in business and enterprise applications for improving efficiency in companies. The fact that Scientific Workflows are becoming more popular can be attributed to several reasons. According to McPhilips (2009) in [13] the large amount of data and complex computation tasks which have been used by scientists have made scientists look for more automated solutions to help them reduce the time and the effort of experiment design. Different programs, methodologies and concepts have been tried to design SWfMS which satisfies the scientists’ demand of task automation. These attempts include several subjects such as user interface design [14] and the model of computation in scientific workflows [15, 16]. Some of this work has been acquired from Petri Net models [17], and some others are inspired from control-flow and dataflow ideas [1].

Ian J Taylor (2007) in [1] and Goderis (2005) in [14] show that when availability of scientific materials on the internet especially on the Web grows, in silico experiments designed by scientists starts to rely more on the Web based technologies. Publishing scientific resources in the Web format and Grid services is a new way of experiment design which is making scientists switch from the traditional processes to use distributed resources and perform their data analysis and knowledge discovery using those technologies.
Service oriented architecture (SOA) is also assisting a lot in the acceleration of scientific research and experimentation. [18] asserts that SOA paradigm has been applied in the orchestration of various services which have been used in the scientific workflows. Software architecture and paradigm are not the only aspect which have been investigated and in the field of scientific workflows. The need for high-performance computing and the use of multiple computers for performing an experiment presented us with new enormous variety of complex issues. Therefore, a lot of new concepts and technologies have been studied and discussed such as: deploying workflows in high performance systems and Grids, fault tolerance and fault recovery mechanisms, workflow modeling languages, data and workflow provenance, management of workflow specification [13].

There are several groups of workflow constructs, which recur frequently in, scientific workflows and they aim to handle specific tasks. Each of these recurring groups of workflow construct can be referred as a Scientific Workflow Pattern.

Scientific workflows are a composition of a series of computation or data manipulation steps. They are designed with the aim of scientific experimentation and can be executed over and over with different data or different parameters.

Several methods to categorize scientific workflows are available. We can classify them by the model of computation they adopt, characteristics they have, technology they use and the level of abstraction they support.

2.2 Characteristics of scientific workflows

In [13] four key properties of scientific workflows are presented: reusability, performance, design and collaboration. These are handled differently in each scientific workflow management systems. Here in this dissertation, we are describing some of the
characteristics. These properties in [13] are highlighted as the authors claimed that a WfMs should:

- Assist scientists in order to design and implement workflows
- Support for modeling and managing scientific data, not just analytical processes.
- Optimizing performance should be handled by the workflow management systems automatically.

**Reusability:** Workflow management systems have to make it easier for workflow designer to reuse their previously developed workflows in their under development workflows. Therefore, designers do not need to start developing a new workflow from scratch. It is the responsibility of Workflow Management System to let the designers reuse and minimize the process needed in order to reuse an existing workflow as well as preventing the errors that might happen to do so. Also, it should log and show the errors when they are occurred. SWfMS should always note that the process of reusing a workflow should be easier than designing from scratch. For this aim, some mechanisms have to be implemented in the workflow such as letting users define a sub-workflows, design and use templates for workflows or define a workflow as a component.

The nature of scientific research is exploratory and predicting the outcome of each experiment developed by SWfMSs is not easy. Workflows should let users easily modify them (i.e., Change the input data, manipulating parameters), connect and compose them into a chain of processes and track them.

Identified workflow patterns can be mixed with the mechanisms mentioned above in order to improve the reusability for the scientific workflows.

**Performance:** Workflow optimization is not a trivial task, there are different methods can be applied on a workflow to reduce the execution time. SWfMSs are responsible for
applying optimization steps over a workflow to improve the performance. Workflows executed on only one computer to those large and complex workflows executed on a high-performance grid environment have to have the ability for optimization and they do not have to expect that the scientist understand complicated software concurrency concepts such as synchronization, dead lock and race condition. The SWfMSs should automatically optimize the performance and take advantage of all the available hardware resources.

Due to the fact that optimization has to happen automatically by computers, workflow patterns can be used to identify the places which the computer can apply the optimization.

**Design:** Almost all the modern SWfMSs provides a rich graphical user interface for creating workflows. For example, Taverna[9], Kepler [19] , Pipeline Pilot [20] , Triana [21] and Vistrail [22] have advanced graphical user interface which lets the designers compose workflow elements and create a graph, where nodes in the graph indicate tasks [23]. The aim of providing graphical composition mechanism is to ease the step of describing workflows for the workflow developers.

According to [24] and [25] having the ability to represent patterns as a graphical element can improve the design process.

**Collaboration** Due to the increase in the number of workflows and collaborative nature of scientific research projects developing share and collaboration mechanisms through the network and internet for workflows is a must. Some projects such as myExperiment specially focused on this approach [26].

These four main characteristics of scientific workflows are mostly common between scientific workflows and business workflows.
2.2.1 Abstract vs. concrete workflows

Workflows have been implemented for a while, and some shared functional elements between them have been identified in workflow repositories. In [27] workflows with no shared functional elements are called concrete workflows and those workflows (or sub-workflows) which are extracted from the shared functional elements of workflows are called abstract workflows [4]. According to [28] “An abstract scientific workflow is a definition of a scientific process with emphasis on the analytical operations or function to be performed rather than the mechanisms for performing these operations” and “A concrete workflow is a definition of a number of tasks represented as actual executable services. A concrete workflow can be converted to specific workflow language and sent to a workflow engine to be executed”. Translating an abstract workflow into a concrete workflow is a process of discovering suitable services that implement these functions and solving the connectivity between services. These two types of classification in workflows help users with different level of understanding of domain to create and use workflows in a fine grained level [28]. In [14] deciding on type of abstraction to be used is described as a modeling decision and it is said that the decision is mainly related to the application.

Patterns are also come from the abstractions. Later on this thesis you can read about the granularity of scientific workflow patterns. Figure 3 depict an example of abstract workflow in the Wings workflow system along with the an executable instance of the same abstract workflow acquired from [29].
Figure 3 Executable workflow (left) and abstract workflow (right) for the protein structure comparison portion of the dragomen.

The right workflow on Figure 3 shows an abstract workflow with the abstract processors coloured in the grey. The workflow on the right shows the concrete workflow with real processors, which is executable. We can see that for example the abstract “GetSignificatResults” processor is replaced with concrete “GetFATCATSignificantResults” processor or the concrete “FATCATProteinIDChecker” is replaced with abstract “ProteinIDChecker” processor.

### 2.2.2 Scientific Workflows Reuse

Reusing scientific workflows is important because it can decrease the time and effort of workflow design and development. Describing workflows as a graph is one approach that we can look at the abstract or concrete workflow. When we describe them as a graph, different methods related to graphs can be applied for finding similar workflows and shared elements in them such as graph matching techniques \[30\]. Finding similarities guide...
us to the way that we can reuse workflows. Since we are focused on scientific workflows, looking at scientific workflow repositories and applying those techniques can be really useful.

In [28] three ways of scientific workflows reuse is described.

- Reuse by third parties: This is a kind of reuse, which is based on the scientific collaboration where third parties can access to the workflow and reuse it in their own experiments.

- Reuse by collaborators: Scientists in a research group who know each other and working together usually exchange knowledge with each other by sharing fragments of workflows.

- Personal reuse: Sometime building large workflows can take a lot of time. Designers can reuse their own fragment of workflows in the other projects to boost the process of design. A scientist usually focuses on one domain so the services or functional elements he or she use usually shared between the workflows the scientist designs. Sometimes it happens that due to the exploratory nature of scientific workflows, scientists keep changing the workflows and lose the previously designed workflows. In that situation the scientists are responsible for backing up different version of designed workflows. Having a versioning system can reduce the effort of personal reuse by keep tracking of changes and letting scientist to go back to the previous stages of workflows.

Eeach of this reusing mechanism can somehow improve the speed of whole workflow development process. Workflow patterns can be utilized in all the methods described as a mechanism of sharing information.
2.2.3 Scientific Workflow Repository

All kinds of reuse described previously require an environment which lets scientists publish and store developed workflows. Those are named as scientific workflow repository which some of them are introduced as follows.

myExperiment [10] is a scientific workflow repository which lets users upload, store and share workflows created by different Workflow Management Systems. Unlike others it is not a repository for a special kind of workflow management system. It provides advanced socializing concepts and lets scientists to form groups and build relationships to collaborate with other scientists. It lets users search over the repository and download the public and permissioned workflows. The ability to version workflows is provided in this repository, so that users can keep track of changes in a special workflow. myExperiment helps workflow designers to avoid the reinvention of previously developed workflows. It supports the Linked Data and workflow preservation [26]. You can access the workflow meta-data through the downloadable linked data or through the SPARQL endpoint [31]. At the time of writing myExperiment has over 5000 members, 250 groups and 2000 workflows from 21 different systems. In [32] a full lifecycle of workflow development is mentioned.

Due to the large number of workflows uploaded into this repository, an analysis can show us the repetitive patterns in the workflow. In [4] an analysis with the aim of detecting recurring functional elements over the myExperiment was performed and some scientific workflow pattern are identified.
Figure 4 workflows on myExperiment

Figure 4 shows the workflow page on myExperiment. Their types, tags, authors, licenses, groups, Web Service addresses and curators can be used to filter workflow list. A lot of details about workflows, their authors and their users can be found in the myExperiment.

myExperiment is a social website which provides the social features such as making groups, the ability to comment and ranking workflows.

crowdLabs [33] is another repository the only accepts workflows from one specialised workflow managements system called VisTrails. There are less than 200 workflows uploaded to this repository. Beside uploading workflows, crowdLabs let users to create
projects, groups and blog posts. Also some basic socialization features are provided in the crowdLabs.

Figure 5 shows the crowLabs workflow page. Not a lot of information about the uploaded workflows can be seen by looking at the website and in order to get the details about the workflows users have to download the workflow.

Accelrys Component Collections [34] is another special repository for the pipeline pilot workflow management system or Accelrys enterprise platform which are commercial pieces of software. The Component Collections has plenty of components, which let scientists perform scientific, or general-purpose functions. Components are categorized
into the chemistry, biology, material modeling and simulation, reporting and visualization, database and application integration, imaging, analysis and statistics, document search and analysis and laboratory categories. The software developers in that company develop component collections and there is no mechanism available for the workflow developers to share and contribute their own workflows by using it.

Figure 6 presents the chemistry component collection provided by Accelrys for the Pipeline Pilot workflow management system. The components in this library can be reused in different workflows by dragging them from the components tab of the explorer to the workflow composition interface.
2.2.4 Dataflow-based vs. Control flow-based Workflows

Previously in this report we have written about the differences of scientific workflows and the business workflows. One of the key points that distinguish scientific workflows from business workflows is in the execution paradigm. Scientific workflows are inclined to be data-driven because; they are dealing with a huge amount of data [5].

Understanding the execution of a workflow is simpler in the paradigm adopted by business WfMSs since the control-flow is defined explicitly and does not need to be derived from data availability. In Scientific Workflows the control-flow can be hard to detect, as it is defined in terms of fine-grained data exchanges. Modelling control-flow oriented tasks by only using data-flow elements guides us to sophisticated models. For this purpose most of the SWfMSs provide some specialized modelling constructs. In [5] these are called routing constructs. Routing constructs are used to explicitly define the control-flow dependencies such as loops or conditional branches. The adoption of a data-flow computational model in place of a control-flow one, determines three fundamental differences between scientific and business WfMSs:

- **Data tokens vs. shared variables**

In business WfMSs data are usually stored in the variables, which are shared among tasks inside a predefined scope. In contrast, SWfMSs do not use shared variables. In SWfMSs tokens shows both the availability of data and the data values themselves. In this way each task gets its own copy of the data and during execution mainly there is not a situation when two processes affect each other.
For example, imagine a task called A that is waiting for an input value which can be generated by both a task B and a task C. Suppose that this input value is stored into a shared variable x and B writes a value b into x. If C writes another value c before A has read the value b, this value will be lost. Conversely, in scientific WfMSs the data produced by B and C are queued into a channel and retained until A consumes both of them [5].

Figure 7 shows two workflows. The workflow on the left is using x as a shared variable and the workflow on the right is passing data tokens to the x. We can see that the second write on the workflow which uses shared variable the value of variable x is over written by the new value and in the other workflow both variables has been stored in a queue.

- **Self-concurrent vs. self-sequential behaviour**

In scientific workflows usually each task can safely execute concurrently with itself. If there is a data-flow dependency between two tasks A and B, as soon as an instance of A produces the necessary data, a new instance of B can start executing concurrently with other instances of B. This characteristic of SWfMSs reflects the need in a scientific domain to run the same process multiple times with different data sets. In business WfMSs the concurrent execution of the same task needs to be modelled explicitly, e.g. through a
multiple instance task. Such constructs also require taking into account synchronization issues that may arise from the execution of multiple instances of a given task. For instance, specific rules need be defined to merge the values produced by all instances of a task in a single shared output variable (e.g. an array).

- **Individual semantics vs. collective semantics**

In scientific workflows tokens usually contain data and are consumed in the order of their arrival: they are not mutually interchangeable, unless explicitly stated by some special constructs. Conversely, in business workflows tokens represent threads of control that do not carry data information and usually are not distinguished from each other.

The aim of scientific and business WfMSs are different. Business WfMSs are usually relying on human interactions and their main goal is the coordination of the work performed by various human resources within or across organizations. However, SWfMSs focus on the automation and optimization of computations performed by one or just a few users [5]. Almost all workflows have manual intervention but in business the focus is on people orchestration.

In conclusion, execution of workflows are generally related to the way they processes proceed. If a workflow waits for enablement of data it can be considered as data-flow workflow. The control-flow workflows on the other hands follow a designed flow of actions.

### 2.2.5 Scientific Workflow Management Systems

Scientific Workflow Management Systems are software systems developed for automating scientific experiments that need to deal with large amounts of data. The main goal of these systems is to facilitate the reuse and integration of functions and tools through a graphical environment. According to [5] SWfMSs can be used to automate repetitive error prone
activities, such as accessing data, analysis, transforming data, and visualization, which lets scientists focus on a specific domain problem. Optimizing the workflow in execution and resource management are two other important roles of the SWfMSs and they are designed in a way to handle the optimization step with minimum intervention of the workflow developer.

Deploying and scheduling tasks are complex tasks and some SWfMSs have the built-in ability to handle them, such as deploying workflows in a cluster of computers or a Grid [35] environment. There are number of shared goals and characteristics in the SWfMSs which differentiate them from the scripting languages and tools. In [13] some of these differences are investigated, e.g., SWMSs are usually based on the data-flow languages whereas, scripting approaches are based on imperative languages. Scientific workflows are represented as graphs specially directed graphs, with nodes, which represent computational units and connections, which represent the dependency between them. Almost all modern WfMSs enable workflow developers to design and modify workflows through a graphical user interface. The fact that SWfMSs are using data-flow paradigm is makes them suitable for creating modular workflows that are amenable for reuse.

Some WfMSs like Taverna [3] and Kepler [13] let a fraction of a workflow be re-used in other workflows or in other place in the same workflow. This way of reuse creates an abstraction layer for hiding the details from the author of workflow. Usually they let workflows to be defined as processors in other workflows.

There are differences between the SWfMSs themselves. One of the main differences between them is the Model of Computation (MoC) adopted to orchestrate the distributed resources [15]. According to [15] MoC is a formal abstraction of execution in a computer. In most of the SWfMSs a single model of computation is adopted. But there are a few SWfMSs adopting multiple MoC. Another difference between SWfMSs is the workflow
abstraction mechanism they provide. Encapsulating computational units in components is one way of abstraction. It provides a reusable mechanism of computation in a black-boxed element. Template and frames are other ways of abstraction. They are abstraction of control-flow issues surrounding a set of concrete workflows, components, sub-workflows or processors [36]. Figure 7 shows an example of abstract workflow, which is used as a template to create a concrete workflow. Using them allow scientists to focus on the data-flow problems and leave the control-flow issues to be implemented in wrappers around existing actors [36]. Both component and MoC can be simple (standardised) or complicated (non-standardised). Complications in MoC can be overcome by simplifications in components also complications in components can be overcome by simplifications in MoC. Workflow Templates are methods to overcome complications in both.

2.2.6 The Taverna Workflow Management System

Taverna [9] is an open source SWfMS. A language called SCUFL and a MoC are designed to overcome the complexity of data intensive and service-based processes automation. It has the flexibility of being used in different domains and sciences to support experimental investigation. The first release of Taverna was in 2004 and a number of new features have been introduced and improved during the time.

Taverna platform is based on 4 parts:

- **Taverna Workbench** enables the workflow designer to find and compose services and link them together to make the workflow graph.
- **Workflow Execution Engine** executes the composed workflows.
- **SCUFL Workflow Language** is a data-flow language, which is used to define a graph of data interaction and data links between different services.
- **Taverna Service** enables the execution of the workflows on a server.
Scientific Workflow Patterns

Taverna supports executing scripts written in languages such as Java or R. It also provides execution of command line applications, which lets the user to run any type of applications and use the results inside Taverna. It has adopted the dataflow MoC which a workflow that includes a set of processors (such as Web Services) are connected together using data dependency links. According to [3] this is a hybrid model designed to create a balance between expressivity and simplicity, with the ultimate goal of empowering users, who may have some understanding of programming, to assemble complex workflows.

The new features introduced in SCUFL2 let users to use a JAVA API to programmatically create and access to the workflows. In SCUFL2 workflows are described as XML documents which let a lot of tools to analyse them as XML is currently a de facto and widely used standard. The workflows structure is described in OWL ontology and is annotated with URIs therefore; tools can get semantic information about all the components of a workflow. This can be strongly important and useful in a sense of scientific workflow patterns. Figure 1 shows the Taverna Workbench with a sample workflow in it and Figure 8 depicts an example of Taverna workflow which is acquired from myExperiment [37].

Taverna Workbench has a plugin architecture, which lets developers to extend the Taverna Workbench and add a new feature to it.
Figure 8 Example of a scientific workflow which search for the mouse gene pathways in a gene database

This workflow is designed in Taverna workflow management system and it is a biological workflow. According to its description it searches for genes pathways in a pathway database, which reside in the mouse.
It includes inputs and outputs in the top and bottom respectively. The processors types in Taverna and their Figures are listed in Table 1. The inputs, which are chromosome name or number, are passed to a Web Service to get the genome information from a gene database and the result of that is passed to group of local processors and Beanshells to clean it. The cleaned data includes a list of gene IDs which is passed to a nested workflow to generate the gene pathways and pathways IDs. The nested workflow outputs are passed to a set of local processors, Beanshells and Web Services to extract the desired data and get description of pathways, which are passed to final workflow outputs.

<table>
<thead>
<tr>
<th>Processor Figure</th>
<th>Processor type</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>workflow_input</td>
<td>Workflow input</td>
<td>Either data input or input parameters. They should be set to execute a workflow.</td>
</tr>
<tr>
<td>Beanshell</td>
<td>Beanshell</td>
<td>Beanshell is a place to write Java code and execute them within a workflow.</td>
</tr>
<tr>
<td>local_processor</td>
<td>Local processor or local worker service</td>
<td>Set of data manipulation and formatting processors which exist in Taverna Workbench.</td>
</tr>
<tr>
<td>XPath_Service</td>
<td>XPath service</td>
<td>Xpath service to acquire the desired data from XML file</td>
</tr>
<tr>
<td>Tool</td>
<td>Tool</td>
<td>An apparatus to invoke non-taverna local tools</td>
</tr>
</tbody>
</table>
Another primary element in a scientific workflows are arrows. Arrows are representing the way data moves in different elements of workflow. The data moves from one processor to one another when the computation is finished and data is available in the output of a processor.

Figure 9-a shows that data moves from input to the Beanshell and when the Beanshell finishes its computation pass the data to the output. The arrows from input to the Beanshell and Beanshell to the output clearly expose that.
Workflow Views in Taverna

Taverna provides different ways of displaying a workflow. There are switch buttons in Taverna Workbench design perspective, which lets the user to configure the view of a workflow. The main configurations are services display, services alignment and nested workflow display.

Workflow user can choose to show workflow processors with or without service ports which are basically the inputs and outputs of each processors. The default behaviour of the workflow is not to show them as you can see in Figure 9-a that the inputs and outputs of the services are not shown. Figure 9 shows a simple workflow with one workflow input, one Beanshell with one input and two outputs and two workflow outputs and its three different views.

The other type of configuration, which can be applied to a workflow in order to view it, is the alignment of the workflow elements. Workflow user can decide to align workflow elements vertically or horizontally.
The “no service port” view, which is a default type of views for a workflow, is often being used by workflow developers. They encapsulate complexity of processors by showing just a name and a coloured boxed. It is quite similar for the “services as circle” view. However, there are situations where a workflow user or developer needs to get more information about a processor such as number of a processor's inputs or outputs. A workflow user can usually guess the number of inputs or outputs by looking at the number of arrows connected to a processor but it is not the precise way of understanding about them because there is a possibility of unconnected inputs or outputs for a processor. Therefore, the second type of view, “with service port view” is a good choice of displaying processors.

2.2.7 Other Workflow Management Systems

Triana ([http://www.trianacode.com](http://www.trianacode.com)) [38] was originally developed as a visual workflow-based problem solving environment for the gravitation wave detection project GEO600 and used as a rapid analysis tool for wave data. [39] Triana workflows were initially built from Java tools and executed on local machines or remote ones using Java RMI. More recently, Triana components have evolved into flexible proxies that can represent a number of local and distributed primitives. For instance, they can represent a Java object, a legacy system, a web-service, a web-service resource framework, a Grid job, a local file or a remote file.

Kepler ([http://www.kepler-project.org](http://www.kepler-project.org)) [40] is an open-source scientific WfMS developed by the members of the Science Environment for Ecological Knowledge (SEEK) project and the Scientific Data Management (SDM) project. It extends Ptolemy II [15], a software system for modeling, simulating, and designing concurrent, real-time systems, developed
at UC Berkeley. Kepler inherits from Ptolemy II the support for multiple heterogeneous models of computations, captured by the notion of directors that allow the representation of different kinds of systems. The distinctive characteristic of Kepler is the separation between the adopted MoC from the structure of the workflow, which is built combining a set of polymorphic components, called actors. An actor implements a functionality of interest for a particular domain, and its behavior can change on the basis of the adopted MoC.

The main idea is that a complex model can be built hierarchically combining different heterogeneous models with different MoCs [5]. Actor model is adopted in this SWfMS and possible MoCs are investigated in [15].

Kepler uses the phrase *actor* for the processors and the actor-based design is being used in the development of Kepler workflows. There is a good point in the actor-based approach. When a data is produces by an actor it can be routed into other downstream actors that let computations happen simultaneously. But the disadvantage of this approach is the fact that it can become confusing for the user to understand the workflow.

Pipeline Pilot ([http://www.accelrys.com](http://www.accelrys.com)) provides services and a workflow engine basing on Service Oriented Architecture (SOA) [41] allowing very effective workflow life-cycle management, i.e. it ensures maximum reuse of already integrated modules. In addition, it supports SOAP with Web Services Description Language (WSDL) extensions for efficient decoupling of workflow management from services’ internal implementation. In this way, in addition to its built-in functionality, the architecture of Pipeline Pilot has been organized for integration and extensibility and designed to interoperate with external software objects and applications.
VisTrails ([http://www.vistrails.org](http://www.vistrails.org)) [42] is a workflow management system which provides a scalable mechanism for generating a large number of visualizations, comprehensive history management, and systematic maintenance of visualization provenance. VisTrails uses an XML-based dialect to represent visualization pipelines that allows the specifications to be shared and queried. In addition, these specifications are executable and can be used to re-generate images, possibly using different parameters. Last, but not least, the availability of formal specifications allows VisTrails to analyze and optimize these pipelines.[43]

Wings ([http://www.wings-workflows.org](http://www.wings-workflows.org)) [44] is a workflow system which lets the workflow designers encodes abstract scientific workflows. It relies on other workflow execution engine called *Pegasus* for the execution of workflow. Designer can develop a template of a workflow, validate it, select between the execution units to be inserted into the workflow template and finally execute it. It supports execution of a workflow in the distributed environment. Scientific workflows in Wings are described in web standards such as OWL or RDF [29].

### 2.3 Scientific Workflow Patterns

Workflow patterns are collections of units that frequently occur and model a requirement in a process. Workflow patterns are introduced in [45] and recently workflow patterns are getting more attention. They were used to identify business workflow patterns and were classified into *control-flow patterns* which describe how activities are executed and *data-flow patterns* which describe data movement and resource handling. The concept of workflow patterns is also available in the scientific workflows.

Patten
Here we are describing the approaches which are used to categorize patterns.

Patterns in computer software are defined and used in various previous works. [46] describe the pattern as “Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.” Scientists usually have data-oriented approach in their analyses.

Despite the fact that SWfMSs enable scientist to perform experiments over data, they provide crude, simple and low-level constructs for the process of data modeling. Therefore many opportunities in abstraction are missed. An example of this mentioned in [13] which states that if SWfMSs require the designer to model their DNA sequences, alignments, and phylogenetic trees as arrays, strings and other basic data types, then many opportunities for helping users design, understand, and repurpose workflows are lost. These kinds of requirements are observed a lot in the process identifying key components of workflow languages.

SWfMS and the workflows designed with them have been improved in terms of the features. The number of SWfMS users has been increased. These growths bring the increase in the system size and require them to be more reliable, extensible, reusable, scalable and maintainable. Patterns are identified to tackle these requirements. They can solve that problems by reducing the amount of labor [47]. Identification and implementation of patterns in the workflows can help us to improve the reusability of workflow fragments. Also, a working and tested fragment of workflow is a reliable piece of knowledge which can be reused in other experiments.

Scientific workflows are developed by scientists who are experts on their domain [8]. Capturing the repetitive elements of their workflow development behavior guides to the recognition of domain specific scientific workflow patterns.
Growth in the number of scientific workflow patterns necessitate us to classify them. They can be analyzed from different aspects and the three following parts describe few ways of looking at scientific workflow patterns. Figure 1 and 2 are showing an example of workflow patterns with their implementation in Taverna Workbench.

2.3.1 Abstract vs. Concrete Workflow Patterns

Abstract workflow pattern refers to those kinds of patterns which occur in workflows and are domain independent. They are designed with higher level of abstraction and are responsible to do more general tasks to guide the designer with the process of designing workflow in more standard and easier way. Concrete workflow patterns are designed to be responsible for solving small sized problems. Therefore, the frequency of using concrete patterns in workflows is much more than abstract patterns. There are several attempts to identify patterns in scientific workflow but most of them were focused on concrete workflow patterns [5] [7] [25] [6].

2.3.2 Scientific Workflows Control Patterns

Scientific workflow control pattern refers to a type of workflow pattern which is designed to abstract away the complexity of flow control in the execution of a scientific workflow. As described before due to data-oriented view of scientists, it is easier for them to only deal with data. Therefore, the abstraction in the control mechanism can help scientists to focus on the analysis of data.

A collection of twenty patterns describing the control-flow perspective of workflow systems has been identified by Russell et al. in [7] and [5] inspired by Russell’s works to identify more patterns. The following list is the name control-flow patterns recognized by [7] and [5].
- **Basic Control Flow Patterns**
  - Sequence
  - Parallel Split
  - Synchronization
  - Exclusive Choice
  - Simple Merge

- **Advanced Branching and Synchronization Patterns**
  - Multi-Choice
  - Structured Synchronizing Merge
  - Multi-Merge
  - Structured Discriminator
  - Blocking Discriminator
  - Cancelling Discriminator
  - Structured Partial Join
  - Blocking Partial Join
  - Cancelling Partial Join
  - Generalised AND-Join
  - Local Synchronizing Merge
  - General Synchronizing Merge
  - Thread Merge
  - Thread Split

- **Multiple Instance Patterns**
  - Multiple Instances without Synchronization
  - Multiple Instances with a Priori Design-Time Knowledge
  - Multiple Instances with a Priori Run-Time Knowledge
Scientific Workflow Patterns

- Multiple Instances without a Priori Run-Time Knowledge
- Static Partial Join for Multiple Instances
- Cancelling Partial Join for Multiple Instances
- Dynamic Partial Join for Multiple Instances

- State-based Patterns
  - Deferred Choice
  - Interleaved Parallel Routing
  - Milestone
  - Critical Section
  - Interleaved Routing

- Cancellation and Force Completion Patterns
  - Cancel Task
  - Cancel Case
  - Cancel Region
  - Cancel Multiple Instance Activity
  - Complete Multiple Instance Activity

- Iteration Patterns
  - Arbitrary Cycles
  - Structured Loop
  - Recursion

- Termination Patterns
  - Implicit Termination
  - Explicit Termination

- Trigger Patterns
  - Transient Trigger
  - Persistent Trigger
2.3.3 Scientific Workflows Data Patterns

Scientific workflow data patterns are designed to abstract the way data is represented, moved or used in workflows [48]. These patterns not only capture the way data is formed or used in the workflow but also they specify the interaction of data in a workflow with other workflows or workflow elements. According to [49] there are a group of characteristics which quite frequently happen in workflows from the data perspective. This group includes **data visibility**, which expresses the scope of data elements and the way they can be accessed and used by workflow constructs; **data interaction**, which describes the protocol that an active construct can interact or communicate with another external or internal element; **data transfer**, which shows the way that data can be transferred by the workflow elements and **data-base routing**, which characterises the way data elements can affect other aspects of the workflow, especially control-flow aspect. The following list is the name of workflow patterns identified in the [5] and [7].

- **Data Visibility**
  - Task Data
  - Block Data
  - Scope Data
  - Multiple Instance Data
  - Case Data
  - Folder Data
  - Workflow Data
  - Environment Data

- **Data Interaction**
  - Internal Data Interaction
    - Data Interaction - Task to Task
    - Data Interaction - Block Task to Sub-Workflow Decomposition
Scientific Workflow Patterns

- Data Interaction - Sub-Workflow Decomposition to Block Task
- Data Interaction - to Multiple Instance Task
- Data Interaction - from Multiple Instance Task
- Data Interaction - Case to Case
  - External Data Interaction
    - Data Interaction - Task to Environment - Push-Oriented
    - Data Interaction - Environment to Task - Pull-Oriented
    - Data Interaction - Environment to Task - Push-Oriented
    - Data Interaction - Task to Environment - Pull-Oriented
    - Data Interaction - Case to Environment - Push-Oriented
    - Data Interaction - Environment to Case - Pull-Oriented
    - Data Interaction - Environment to Case - Push-Oriented
    - Data Interaction - Case to Environment - Pull-Oriented
    - Data Interaction - Workflow to Environment - Push-Oriented
    - Data Interaction - Environment to Workflow - Pull-Oriented
    - Data Interaction - Environment to Workflow - Push-Oriented
    - Data Interaction - Workflow to Environment - Pull-Oriented

- Data Transfer Patterns
  - Data Transfer by Value - Incoming
  - Data Transfer by Value - Outgoing
  - Data Transfer - Copy In/Copy Out
  - Data Transfer by Reference - Unlocked
  - Data Transfer by Reference - With Lock
  - Data Transformation - Input
  - Data Transformation - Output

- Data-based Routing
Scientific Workflow Patterns

- Task Precondition - Data Existence
- Task Precondition - Data Value
- Task Postcondition - Data Existence
- Task Postcondition - Data Value
- Event-based Task Trigger
- Data-based Task Trigger
- Data-based Routing

These patterns are shared between scientific and business workflows but as scientific workflows are more data intensive than business workflows, they tend to be observed more in scientific workflows.

These identified patterns are low-level (from the abstraction level point of view) operational patterns. There are other high-level patterns in scientific workflows which might be domain specific or domain independent. Tasks such as data conversion or data cleaning happen in a lot of scientific workflows.

This dissertation is investigating patterns introduced by [49] in myExperiment repository which embraces three main areas:

- Investigating the existence of a pattern in repository.
- Possibility of a implementation of a pattern using Taverna.
- Evaluation and comparison of different methods of implementation.

### 2.3.4 Workflow Templates

Workflow templates provide an approach to design an abstract workflow. Their aim is to let workflow designers to encapsulate structure of workflows. Structure of workflows embraces the inputs and outputs of workflow and processors, place of each processor and the links between them. The only thing which is not important at this level of design is the
processors itself. Because, they are the elements which can be swapped with one another which follows the same protocol.

Two main types of users deal with templates: Template designers and workflow designers who use templates.

Template designers are those who develop a template to be able to classify a kind of workflow. They define a protocol on how concrete workflows created by using that template should look like.

Workflow designer who use templates can replace each abstract element with a concrete workflow processor. Using templates help them to develop a workflow rapidly by following template structure.

One of the reasons to develop a workflow template is the fact that designers can think about the big picture in the design stage and define the data movement within a workflow at design phase.

They let the designers who use templates to follow a general rule or interface when they design a new workflow so that the outcome is a set of uniform workflows which can help in different aspects such as workflow discovery or workflow reusability.

Workflow templates can help to separate workflow developers’ responsibility. Scientific workflows can be a result of collaboration of a team of scientists. Hence, two major ways of development can be applied using templates.

First, a scientist with more background and context can design an abstract workflow using templates. This scientist can compose abstract tasks and define the general purpose of workflow by using templates and abstract tasks and after the generation of abstract workflow it can be passed to the other scientists who might not have as enough knowledge
about that context but still can implement concrete parts of workflow to generate the concrete workflow. This approach can also help to develop different variation of a workflow with a little bit difference.

Second, the whole team of scientists can contribute on development of a template. This helps all of them to have an agreement on the main purpose of workflow and after the generation of template they can divide different tasks, assign them to scientists and finally put together all of them to make the final concrete workflow.

Workflow templates can help in both bottom-up and top-down design. In top-down design, we have to design workflow structure upfront and before we start implementing it. Therefore, designing a workflow template can be a useful tool in both phases.

In bottom-up design, developers want to create a fragment of workflow and then reuse it to create a large workflow. If the templating system in workflow management systems provides a way to change the structure of template on the fly, eventually when the large workflow is created, workflow management system can provide the template of workflow to be used for next time development of a workflow.
Scientific Workflow Patterns

Figure 10 Example of an abstract workflow from myGrid wiki website

Currently Taverna workflow management system is not providing a template system itself but, it provides a viewing technique for showing nested workflows.

Figure 10 represents an example of Taverna workflow from myExperiment [50] which the nested workflows are contracted on the left and expanded on the right. The arrows show the mapping of the contracted nested workflows to the expanded ones.

This feature of Taverna can be extended in future to create a system which provides templates for this WfMS.

Due to the fact that scientific workflow patterns can be captured using workflow structure, templates might be a suitable tool to encode pattern in order to reuse them.
To conclude, workflow templates might improve the way workflows are designed and used in different aspects.

- They can bring a separation of roles between the abstract workflow designers and concrete workflow designers.
- Two types of collaboration can be applied using templates. A scientist who design the template and engineers who can implement smaller parts or working on a team to develop the work template and then assign parts to members of the team.
- Both bottom-up and top-down workflow design can be used for workflow development.
- Scientific workflow patterns can be encoded using templates.

### 2.4 Related Works

Several attempts have performed in the field of workflow patterns. These attempts investigated workflows and scientific workflows from different approaches. Some of them were mentioned previously.

The Workflow Patterns initiative [12] is started in 1999. The aim of this initiative is to provide a conceptual basis for process technology. In particular, the research provides a thorough examination of the various perspectives such as control flow, data, resource, and exception handling that need to be supported by a workflow language or a business process modelling language. Several papers have been published in workflow patterns from this group. Papers mainly focused on control-flow based patterns [7] [51] [45] and data-flow based patterns [48] [49] have inspired us in this thesis.

A group from different researchers in universities (University of Verona, Queensland University of Technology, Eindhoven University of Technology) conducted a technical report [5] for evaluating patterns in various scientific workflow management systems.
However, information provided are not confirmed by the workflow management systems developers but their method have adopted was studied for this report.

An investigation [4] over shared workflows in the myExperiment repository has been conducted with the aim of identifying shared workflow elements. At the beginning, identical elements in the functionality have been described by analysis of inputs and outputs. Then they discuss various methods for comparing workflow elements at different levels.

To conclude, works mentioned above are mostly focused on concrete workflows patterns and an abstraction over the identified patterns cannot clearly be observed. However, their approach in investigating patterns is helpful because, identified concrete patterns can be composed in order to form the abstract scientific workflow patterns.
Chapter 3 An approach for investigating and acquiring patterns

3.1 An approach for investigating patterns

This dissertation studies the concept of patterns in the scientific workflows and to accomplish this goal, the investigation is divided into three general stages. The first stage is to collect the list of patterns in scientific workflows. The second stage is to analyse the collected patterns and the final stage is looking at the methods of pattern implementation in scientific workflows.

The rationale behind this type of separation is the fact that each of these phases has their own difficulties and they are required totally different methods of solving problem.

![Figure 11 Structure of investigation](image)

Figure 11 represents main stages of this investigation along with the dependency to their previous stage.
3.2 Pattern Acquisition

This thesis has been aimed to find recurring patterns in the scientific workflows. We have been looked at several resources for this purpose.

Acquiring scientific workflow patterns is the step where required data about existing and non-existing patterns are gathered. Two main resources have been used in this dissertation.

- A list of data workflow patterns introduced by [12] for existing patterns.
- The myExperiment repository to analyse and evaluate the existence of the patterns and to identify new patterns.

3.2.1 Catalogue of dataflow-based patterns

Workflow data patterns in [12] include a catalogue of dataflow-based patterns for workflows. All these 40 patterns represent the common ways for data movement within the workflows. Their names are presented previously in section 2.3.3. This rich resource of information has been studied to check the possibility of implementation of data patterns in Taverna Workbench.

3.2.2 An example of workflow data pattern

An example of data pattern has been chosen to demonstrate what these patterns look like and what kind of information this catalogue provides about them.

This pattern is called “Scope Data”. Its purpose is to define which task or processor inside a workflow can access the data in a case. In the Taverna Workbench a case can be interpreted as a nested workflow or dataflow. This pattern is a part of data visibility category.
Figure 12 is showing scope data pattern where a variable named “x” has been defined inside a case and the variable visibility has been scoped to be shared only between tasks A, B and C. Therefore, variable x cannot be accessed in other tasks or sub-workflows within a case. For instance tasks D and E in the case, and X, Y and Z in the sub-workflow cannot access the value of data X.

This catalogue includes more detailed information about a pattern such as a flash animation which interactively demonstrates what the pattern is; an example situation which this pattern might occur; motivation of pattern; ideas about the implementation; and a list of products that are usually commercial control-flow based workflow management systems that have implemented that pattern.

### 3.2.3 myExperiment repository as a resource of study

The myExperiment ([http://www.myexperiment.org](http://www.myexperiment.org)) repository is a rich resource of workflows. We have selected myExperiment because of:

- The large number of workflows that has been stored there. 2000 workflows at the time of writing this dissertation have been uploaded to the myExperiment.
- The good support for Taverna workflows. 1540 workflows out of 2000 workflows are Taverna workflows.
- The different ways of accessing its content
The myExperiment provides its information in 4 ways:

1. **Web access**

This is the common way of accessing myExperiment. Users can browse the website, search the repository, get the information about a workflow and download the workflow. The page related to each workflow provides the following information about a workflow:

   - Title
   - Version number with its creation date and last edited time
   - Author(s) name
   - Type of workflow: It is the WfMS which is used to create the workflow
   - Preview Image
   - Description
   - Inputs and outputs name and description
   - Processors name, type and description
   - Beanshells name, description, inputs and outputs
   - DataLinks
   - Comments and reviews

The website information is human readable. However, in order to make it machine readable different approaches can be used; For example, the website can be crawled by a program to encode the data into a machine readable format such as JSON or XML. However this might not the best approach in our research since myExperiment provides machine readable information itself.

2. **REST API**

REST is a style of providing information as a Web Service. The myExperiment is rendering REST request responses in XML format. The REST API is documented in [52].
3. Linked Data

Every myExperiment entity has a unique URI for the purpose of identification. Through an HTTP request, details of an entity can be retrieved in HTML, RDF and XML formats. The type of desired information should be defined in the HTTP request header as follows:

- “Accept: text/html” return the result in HTML format.
- “Accept: application/rdf+xml” returns the result in RDF format.
- “Accept: application/xml” returns the result in XML format.

XML and RDF formats are both machine readable. Furthermore, myExperiment publishes its ontology in RDF format so that the mixture of ontology and RDF linked data can be used for querying myExperiment entities.

Resource Description Framework (RDF) is a type of web specification and is designed as a metadata data model. RDF is used as a general way of describing information which is inside web resources.

Querying RDF files is not a trivial process. Therefore, query language[53] is being used in order to acquire the information from RDF files. myExperiment SPARQL endpoint [31] returns the query result in different formats such as XML and JSON.

There is a limit in the consumption of memory for each query in SPARQL Protocol and RDF Query Language (SPARQL) endpoint. SPARQL is a query language which provides searching over RDF data format. SPARQL allows for a query to consist of triple patterns, conjunctions, disjunctions, and optional patterns. SPARQL can search over linked data information. Linked data is a method of publishing structured data
so that it can be interlinked. There are 4 main components being used in Linked Data: HTTP, URIs, RDF and serialization format.

When a query reaches that limit the SPARQL endpoint engine (4Store Reasoner) stops the execution of query. This is important point when the result of a query is generating big results. Therefore, in the situations where SPARQL endpoint is called in this project, big queries are divided into smaller ones.

4. Downloadable Linked-Data in RDF format

All the myExperiment RDF entities and ontologies are packed into a tar.gz file and can be downloaded. The RDF query engines such as Apache Jena can provide an API or querying facility over the RDF files. The point of using this packed file is that a developer can define the memory usage limit and can execute queries which are using large amount of memory. Furthermore, due to the fact that these queries are running locally if the processing power of local machine is comparable to the myExperiment endpoint, the results might appear faster than remote endpoint because, there is no request to remote server which needs interactions over the network.

Executing a SPARQL with Apache Jena is as follows:

- Define the ‘JENAROOT’ environment variable which points to the Apache Jena root directory.
- Download the RDF dataset.
- Write the query in separate file.
- Run the ‘SPARQL’ command with following arguments:

  $ ./SPARQL –dataset=/path/to/dataset –query=/path/to/query

In this command, dataset argument should point to the file where dataset is located and query argument should point to the place where query file is.
This approach has been tested during this research and several conflicts with the result of online SPARQL endpoint and Apache Jena results have been observed. They have been the main reason which stopped the research to use this approach.

### 3.2.4 SPARQL Queries

Several SPARQL queries for myExperiment have been written in this project. Most of these SPARQL queries have been used in order to gather information about workflows and their structure. Therefore, two main part of myExperiment have been studied more than the other parts. These two parts are:

- The contributions ontology [54] provides information about contributions in the myExperiment.
- The Components ontology [55] presents the components within a workflow and their details.
Figure 13 shows two screenshots which have been taken from ‘Protégé ontology editor and knowledge base editor’ software. Figure 13-a represents the classes in components classes in the ontology and Figure 13-b shows the contribution classes in the ontology. By looking them the importance of these ontologies becomes clear because components ontology represents the structural information of a workflow and a workflow is a contribution which can be described through contributions ontology.

All SPARQL queries which has been written for this thesis has been included in the SPARQL queries part of the Appendix.
The myExperiment ontology has been well documented. The description of classes and annotations in the RDF files, the information the different myExperiment modules [56], along with the auto generated documentation [56] has been helpful to understand how the system has been working and what modules has been more important in our research.

Existence of conflicts between the information resulted from execution of query on local machine using Apache Jena and myExperiment SPARQL endpoint was a point of doubt. A comparative analysis between the both results about the components inside a sample workflow has shown that the myExperiment endpoint generates more accurate results than the local Apache Jena results. Since, the result of myExperiment SPARQL endpoint is more accurate, it was selected to be used for this dissertation. However, the difference in the results has been reported to the myExperiment developers a fixed version of downloadable linked-data for myExperiment has been produced.

### 3.3 myExperiment Scrapper

Executing SPARQL queries that generate lots of data and consume the endpoint memory can cause the SPARQL endpoint to crash. One solution can be the division of the query into smaller queries which might results in less consumption of resources.
In the pattern acquisition part of our research, this technique has been used to in order to gather information about workflows in myExperiment. For example, instead of writing a query which collects information about the all workflows in the repository, a query which collects information for only one workflow has been written, executed and the results have been saved. Another query has been used for the purpose of listing all the workflows. A script to execute a query over all the workflows has been written which has been used both queries.

The SPARQL query listed in Figure 14 is showing the query which has been used to list all the workflows, workflow versions and dataflows.

The query in Figure 15 has been used to gather information about workflow number 16 in myExperiment. It has included the information about a component, its type, the component it has been connected to with its type, the link which has been used to connect those component together and the title of the component. If the type of a component in the workflow has been equal to WSDL processor, the Web Service information has been shown in the optional part.

The WSDL processor is a processor in Taverna which creates a service based on a WSDL. WSDL is a language being used to describe Web Services.
Figure 15 An SPARQL Query for myExperiment which information about the first dataflow of the workflow number 16 in myExperiment.

The part of the scraper which has been responsible to gather this information for all the workflows has been parameterised to accept the dataflow URI as a parameter. It has been resulted in a technique to gather information of workflows one by one.

A ‘myExperiment Scraper’ written in Ruby, goes through all the workflows and gather the specified information about them.
Figure 16 shows that the myExperiment Scrapper uses two SPARQL query one for listing all the *Dataflows* which includes workflows and nested workflows inside them and another query to find the specific information about each dataflow. The application then invokes the remote myExperiment SPARQL endpoint per dataflow to fetch the XML results and parse them to be able to store them as CSV format. The output of this application is only one CSV file which means that all the acquired results have been appended to the CSV file.

### 3.4 Summary

This chapter has described the resources, tools and techniques which have been used in this dissertation to acquire patterns. The resources have included the myExperiment repository and a catalogue of data patterns. Furthermore, tools such as SPARQL for the purpose of querying myExperiment, myExperiment Scrapper, a Ruby application developed for the project to automate the task of querying have been employed. In addition, different
techniques and technologies for instance, Linked Data, REST requests, Apache Jena, RDF, Semantic Web Ontology and etc have been applied in this phase of dissertation.
Chapter 4 Pattern Analysis

The primary data which has been generated or acquired in the first phase of project is considerable. The result for execution of myExperiment Scraper using the queries in Figure 14 and 15 generated 34940 rows of data in the CSV file. There were 40 main queries for matching patterns inside the myExperiment repository which some of them generated results and some of them did not because there were not any instances of that pattern inside the repository. This results might have been useful for a computer but since, we have been in the analysis phase we have needed a more summarised and human readable results. Therefore, in the analysis phase, few approaches have been taken in order to summarize the generated results.

The results of running myExperiment Scraper with several SPARQL queries on Taverna data-flows has been enormous. As it has been mentioned before, execution of one pair of queries has generated around 35 thousands rows of information.

4.1 Catalogue of data patterns analysis

We reviewed every pattern listed in the catalogue of data patterns in. In order to be able to find those patterns in the repository few changes are necessary:

- The style of defining and using data in Taverna workflows is different from the style of defining data in the control-flow workflows used in the catalogue.
- The patterns in the catalogue have several structural differences from Taverna workflows.

An example of these is the difference between the way you can define and consume data in Taverna and the way mentioned in the catalogue. In the catalogue it has been observed that in the author has defined a variable inside a task and has used the variable in it. In Taverna
defining variables is by passing them through ports. The Beanshells define variables more freely similar to the style of catalogue inside the scripting area.

Figure 17-a demonstrates the way the pattern catalogue has used to define and use the variable x. It has shown that the both definition and usage of the variable has done inside the processor. On the other hand in the Taverna examples, Figure 17-b and Figure 17-c, the variable which has been used by the processors, have been initialized and assigned a value before the execution of the processors and the variable has been passed through the input ports to the processors. In Figure 17-b we used the Taverna Local Workers which are pre-defined processors provided by the Taverna they help on common tasks such as opening a file and saving to the database and they are local, which means they do not call remote services. As showed in Figure 17, the definition and usage of a variable in Taverna has been a two-step process which has been different from the one-step process of variable definition and usage in the catalogue of patterns example.

The observation of these differences has guided us to generate SPARQL queries in a way that patterns could have been captured in Taverna workflows. The results of queries over
myExperiment have shown that there have been only few patterns from the catalogue which has been occurred in the Taverna workflow of myExperiment repository.

Since, most of the patterns in catalogue were not found in the myExperiment, a process of looking for the reasons of that have been started. It has been found that due to the lack of support of special features and properties in Taverna Workbench, implementing all of those patterns has not been possible. Those features have been:

- Lack of support for definition and usage of the shared variables.
- Lack of support for scoping the visibility of the shared variables.

A plugin for Taverna Workbench developed to extend the features of the it in order to support the definition, scoping and usage of the shared variables.

### 4.2 Data cleaning

For the better realization of this results, a tool called “Google Refine” [57] has been used for cleaning and improving data. Google refine has been used for grouping by using the *text facets* functionality. This lets the user select a column and group the results. Figure 18 shows the Google refine user interface with a fragment of data on the right and facets on the left panel. In this example, text facets on the component types have been applied.
Google refine has also been used for improving the information. It provides a service called “Reconcile” to generate more data based on the current information stored in the columns. This service has been originally worked with Freebase (www.freebase.com) data. Freebase has provided an RDF end point which has let the user to get information about an entity in a programmatic way. Since, the information we have been looked for, have been stored in myExperiment repository, Freebase has not been a resource for our research. Therefore we sought to enable the reconcile service for myExperiment.

Using a plugin [58] which has let us define new RDF endpoints which could be a SPARQL endpoint, RDF file or Sindice service. After the installation of the RDF plugin the new button in the extensions part of the Google Refine has been appeared to let us define a RDF endpoint. This has been shown in Figure 19.

Since, myExperiment is currently supporting SPARQL endpoint we have used the reconciliation service based on the SPARQL Endpoint.
We have defined the types of columns using the RDF extension and reconciliation service and to create new columns. For example in Figure 20, we have defined the type of a column as *DataflowProcessor* and then created a new column which have represented the processor type, inputs, outputs and links.

### 4.3 Result of the analysis

The information which has been gathered from the myExperiment has been analysed to find the workflow patterns in them. We have been looked for the way data has been moved in a workflow. Therefore, we have been interested in the way the processors have been connected to each and how frequent they are.

The results of this analysis have shown us the occurrence of following 9 patterns:

- “*Task Data*” pattern where data has been defined and has been used inside a processor. An example of this pattern can happen when we define a Beanshell inputs. The Variable which we define as Beanshell can be used only inside a Beanshell processor.
• “Block Data” pattern where data has been defined in a nested workflow and has been used by the processors inside it. An example of Block Data pattern is the nested workflows inputs in Taverna workbench which are accessible for the nested workflow processors.

Figure 21 shows an example of Block Data pattern in part of a workflow [59] from myExperiment. We can see that the inputs in the nested workflow are accessible by the processors inside the nested workflow by their links.

• “Multiple Instance Data” pattern where a processor has been executed number of times with different inputs (using loop) and the final results has been merged.
Taverna has an implicit iteration system. It automatically iterates over the data if you pass a collection of data to a processor. Workflows using this feature can be examples of multiple instance data.

Figure 22 is a workflow [60] with the multiple instance data pattern from myExperiment. It accepts a list of data from the inputs and iterates through all the processors for each item in the list. The output is a list of strings with a result of each iteration as an item in the list.
Scientific Workflow Patterns

- "Workflow Data" pattern which data has been shared between all the workflow processors. All the processors in a dataflow are shared between the dataflow processor. For example if we have 3 services and a nested workflow with 4 service in a workflow, all the 3 services and the nested workflow can access to each other processors output. However, the services in the main workflow cannot access to the nested workflow processors output and vice versa.

- "Task to Task" pattern where two processors have been connected together. In all the workflows which we have more that processors that they are linked together we can see the Task to Task pattern. For example in Figure 22 The List_Emitter processor is connected to the Concat processor.

- "Block Task to Subworkflow Decomposition" pattern where a value in a workflow processor has sent to a nested workflow for computation. Figure 21 shows as example of this pattern where we passed a value of a processor to a nested workflow.

- "Subworkflow Decomposition to Block Task" where the computed results of a nested workflow has been sent back to the workflow. For example in Figure 21 the output of the nested workflow called bmuris is passed to another processor called Flatten_List.

- "Task to Environment and Environment to Task Push and Pull" pattern where data from an external resource such as MySQL database has been retrieved by a processor and has been stored to database.
Figure 23 shows an example of this pattern in a workflow [61] which executes a SQL query on MySQL database and returns the results. Taverna provides database connection local workers which can be as a guide to find this kind of patterns in the repository.

- **“Data Transfer by Value Incoming and Outgoing”** pattern when a dataflow processor has connected to another one to move data. From the user perspective in the Taverna Workbench all data is transferred by value. However we have found that inside Taverna workflow execution engine data is being transferred by reference.

The reason other patterns have not been observed might have been one of the reasons below:

- Workflow designers have chosen to encode the logic in Beanshell or R scripts. This has happened mostly when the logic has been complex or the designer wanted to evaluate a condition. This has happened where workflow designers wanted to evaluate a condition and connect to non-predefined external resources.

- A common form of implementing patterns has not been used. Therefore it has become hard to detect.
• Lack of support for shared variables, workflow templates and workflow components.

During this analysis more higher level patterns has been detected called motifs [62] but since we have focused on the lower level workflow patterns, those patterns have become out of the scope of this research.

4.3.1 Lack of templates and components in Taverna

This analysis has been made completely manually. Due to the lack of structural constructs in Taverna Workbench not many of workflows have followed the workflow design rules. For example it has been observed that in several situations where a combination of processors could have made the same effects, hardcoded Beanshells script have been used.

The lack of workflow templates and components might have been one of the most reasons where workflow designers have preferred to write Beanshell or R scripts instead of using or reusing a composition of workflow processors.

A workflow template could have helped the workflow designers to follow a pattern in their design. In addition, a component system could have let the workflow designers to select a fragment of already developed workflow and have defined it as a new component in Taverna and reuse it later.
Chapter 5 Pattern Implementation

In previous chapters we have identified what properties have made it difficult for the workflow designers to use or implement patterns. In addition, the places where Taverna must be extended to enable the usage of patterns have been identified and tools based on them have developed. Those are the variable manager, workflow templates and workflow components. In this chapter, we looked for the places where Taverna provides extension mechanisms to enable the usage of patterns in it and extended the Workbench in order to support those features.

5.1 Extension of Taverna Workbench

There are two main ways to extend Taverna features. The is to check out the source code from the SVN repository [63] and change it. This way has provided freedom for the developers by letting them to change the source code in any aspect. However, those changes might make the extended software un-updatable in future. A better alternative is to develop an extension plugin for Taverna Workbench.

Taverna uses a technique called the Service Provided Interface (SPI) in its architecture and implementation. Almost every Taverna Workbench module has been developed using the SPI and those which has not developed by using SPI, have been the core modules that has been providing the initialisation of Taverna Workbench and the ability to load SPI modules.

SPI brings the possibility of replacement of a module with another one or the possibility of extending it. The list of Taverna SPIs has been provided in [64]

5.2 Shared Variable Manager Plugin for Taverna Workbench

A feature which lets the workflow designers to define workflow shared variables has not been supported by Taverna. This feature has been one of the crucial parts of the patterns
introduced in the catalogue of data patterns. Therefore, Taverna Workbench should must be extended in the way which it supports shared variable definition and usage.

A simple solution with the ability of scale in requires two main extensions in the Taverna Workbench:

1. A new button in Taverna tools menu and a dialog which lets the developer to define variable names and their values.
2. Two processors which load and update the variables value when they have been called.

These plugins present the user with the ability to define a variable name as a string and an assigning value with the type of string. All these information is written to a file and separated by separator which has been the “~” character. Therefore if a user has defined the variable “x” with the value of “one”, “y” with the value of “two” and “z” with value of “three” in the variable manager, the result should have been similar to Figure 24.

```
x~one
y~two
z~three
```

*Figure 24 An example storage file of Shared Variable Manager plugin.*

Two new processors have been developed to retrieve and store variable from and into the storage file.

- The **load** processor accepts the variable name and returns the value of the variable.
- The **save** processor accepts two inputs: variable name and the new value. If the variable has existed before, it will change the value. If the variable has not existed before it will create a new row in the file to persist the variable.
5.2.1 Scoping Variables

The plugin needs to support the scoping of the variables. The granularity of scoping in workflow could have been defined in several ways:

- Variable visibility inside **processors** only.
- Variable is visible to a **group of processors**.
- Variable is visible to a **dataflow**
- Variable is visible to a **workflow**

We have chosen the **dataflow** for the granularity of variable visibility. This means whenever a variable is defined and assigned to a dataflow, the all elements and processors inside a dataflow can see and edit that variable. There have been three reasons for choosing dataflow for this purpose:

- Processors as a level of granularity have been small.
- A dataflow has included a group of processors, so if an author wants to limit the visibility of a variable to a group of elements he could have defined a new dataflow.
- Workflows as a level of granularity have been too big for this purpose. As, the variable have been shared inside the whole workflow and scoping could not have been applied to this level of granularity.

5.2.2 Dataflow Unique ID

Every dataflow in Taverna owns a unique identification key which has been generated by the Taverna Workbench once a designer has created a dataflow. This ID has been used to distinguish dataflow from one another. This value has been written in the storage file after the variable value and has been separated by the defined separator.
Some data-flows within a workflow should have read permission and not the write permission. Hence, we have added two main properties to each variable which lets the workflow designer to set the read/write access for a variable in a dataflow. The workflow designer could have set a variable to be readable in one dataflow and readable/writable in another dataflow. These two values have been written in the storage file after the dataflow ID. Since, these the have been Boolean values, we used “1” and “0” to store their value.

5.2.3 Defining a variable

Figure 25 shows the new icon for the Shared Variable Manager added to the Taverna Workbench. Once the user clicks on it, a new dialog appears on the Workbench which lets the used to define a new variable and assign the permissions to it.

![Figure 25 Shared Variable Manager icon added to the Taverna toolbar](image)

The dialog in Figure 26 is opened by clicking on the added icon and it presents a table with five columns on the top and a panel on the bottom to add a new variable. The table columns are:

1. Variable name
2. Variable value
3. Dataflow: It refers to the dataflow unique ID in this picture.
4. Read: A Boolean value which is true when checked.
5. Write: A Boolean value which is true when checked.
The user can use the bottom panel to add a new variable, fill the variable name and variable values text boxes and click on the add button. Then he can define the read/write permissions and click on the save button to update the storage file.

The example file which has stored the variables and their information has presented in the Figure 27. The variable information has been separated with “~” character.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Dataflow ID</th>
<th>Read</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>one</td>
<td>28edf477-9f33-4905-9574-5a1e9c63dcd0<del>1</del>1</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>two</td>
<td>28edf477-9f33-4905-9574-5a1e9c63dcd0<del>0</del>1</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>three</td>
<td>28edf477-9f33-4905-9574-5a1e9c63dcd0<del>1</del>1</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>four</td>
<td>4</td>
<td>28edf477-9f33-4905-9574-5a1e9c63dcd0<del>0</del>1</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>five</td>
<td>5</td>
<td>28edf477-9f33-4905-9574-5a1e9c63dcd0<del>1</del>1</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>six</td>
<td>6</td>
<td>28edf477-9f33-4905-9574-5a1e9c63dcd0<del>0</del>0</td>
<td>✔</td>
<td></td>
</tr>
</tbody>
</table>

Figure 27 Example of variable storage file

Each row in this file has represented a variable with all its properties. For example, the first line of this file has been related to the variable named “x” and the assigned value has been “one”. The data flow ID has been “28edf477-9f33-4905-9574-5a1e9c63dcd0” and the variable has been both readable and writable in the defined dataflow since, both values have been set to “1”.

Figure 26 Shared Variable Manager dialog added to Taverna Workbench through plugin extension
5.2.4 Variable Manager Services

The plugin has added two new services to the services panel of Taverna Workbench. These two variables have been used to load and store a variable’s value when the workflow has executed.

![Service panel showing added services](image)

**Figure 28 Two added services to load and save variables**

User can drag the “Load Variable” service and drop it on the Workbench to add the service to the workflow. This service has one input and one output. The input is the variable name and output is the variable value.

![Diagram of variable loading](image)

**Figure 29 loading a variable with the load service.**

Figure 29 shows a workflow which a string constant named “x” has been connected to the load service input. The load service accepts the variable name and checks if the dataflow which is loading the value has the read permission. If it has the permission it returns the variable value, if not it generates an “access denied” error.
Saving a variable is a similar process to load a variable. The difference is the load service, accepts two inputs: variable name and variable value.

![Diagram](image)

**Figure 30** Save a variable with the save service.

In Figure 30 saving a variable has been presented. The string “x” has been connected to the variable name input and the new value has been connected to the variable value. In the save service, in contrast to the load service, it checks the dataflow has the write access to variable and it generates the new value in the output or it generate an error if it fails to write the variables value.

**Using dataflow names instead of dataflow IDs**

As it is shown in Figure 26 dataflow IDs have been used to distinguish dataflows because of its uniqueness. However, it has been observed that the dataflow ID is not a good choice for the end users, because, they do not deal with the dataflow ID and they only see the workflow name and nested workflow name. Therefore we represent the dataflow name in the Shared Variable Manager, but stored the dataflow ID in the storage file and convert it to the dataflow name on the user interface. This conversion is encoded using the Taverna API.

The catalogue of data patterns has included a big list of dataflow patterns. Implementing them has required us an analysis step to figure out which features have been missed in the
Taverna Workbench and what are the Workbench extension points. It has been concluded that a Shared Variable Manager plugin in Taverna can enable us to implement them. The creation process of this plugin and its features has been described in this part.

5.3 Development of the tools

Development of the tools has had several complexities. Extension of the Taverna Workbench has been undertaken by using “Taverna Extension Points”. One plugin has been developed for enabling both Shared Variable Manager and workflow templates in Taverna Workbench and one plugin developed by Taverna team to support the workflow components.

5.3.1 Development of the Shared Variable Manager

This plugin has been responsible for bringing the functionality of defining and using shared variables with the ability of visibility scoping of a variable.

Taverna has been developed using the Java language and the development of plugins in Taverna has been supported only in Java with the usage of SPIs.

The Shared Variable Manager has been developed using four main SPIs.

1. `net.sf.taverna.t2.servicedescriptions.ServiceDescriptionProvider`
2. `net.sf.taverna.t2.ui.menu.MenuComponent`
3. `net.sf.taverna.t2.Workbench.activityicons.ActivityIconSPI`
4. `net.sf.taverna.t2.Workbench.ui.views.contextualviews.activity.ContextualViewFactor`

The first SPI, “ServiceDescriptionProvider” has been used to add two “Load” and “Save” services to the service panel.
The second SPI, “MenuComponent” has been used for the purpose of adding the variable manager icon to the Taverna top toolbar as it has been presented in Figure 25.

The “ActivityIconSPI” has been used to change the default icon of the services to the desired one. The last SPI “ContextualViewFactory” has been used to define the activities information such as inputs, outputs and user interface options.

5.3.2 Development of Workflow Components

A workflow component plugin has been developed by the Taverna team. At the time of writing this dissertation, it has not been added to the Taverna core functionality yet. However, we have used the 0.1.2 version of this plugin for creating components in this thesis.

This plugin has enabled the feature of defining a group of processors as a component. A component in Taverna is an encapsulated workflow and it can be defined as a reusable service in the services panel. The process of creating a component with this plugin has been as below:

1. Develop a processor, nested workflow or a workflow.
2. Right click on the entity that you want to be considered as component and select create component.
3. Store the component either locally in your machine or remotely in myExperiment repository.
4. Import the component stored component to the service panel.

After the above steps have been done, a new component has appeared in the service panel and could have been dragged into the workflow. Figure 31 has presented a component added to the service panel.
In order to install this plugin in Taverna we have added [65] to the Taverna update sites.

The process of installation has been describe in [66].

5.3.3 Development of Workflow Templates

Creating a template for a workflow has been identified as one of the obstacles which have reduced workflow reusability. In addition, a pattern could have been implemented as template and reused.

There have been a number of ideas about how a workflow template system could have been designed for a workflow management system. However, we have found a solution which with the minimum amount of changes we could have enabled the workflow template system for the Taverna Workbench.

Our solution has included two parts:

1. A “dummy” processor which has been added to the services panel and has no responsibility for computation.
2. A method of swapping available workflow processors with the dummy processor.
With the above functionalities, we can design a workflow template and replace the real processors with the dummy processor.

A problem is that the dummy processor has had static number of inputs and outputs. Therefore a lot of dummy processors with different combinations of number of inputs and outputs have been created to be used in different situations of workflow templates.

In our project, we have limited the implementation of these combinations to only one input and one output. Since, the purpose of this part of the implementation has been to prove it has been possible to implement this feature in Taverna Workbench.

Figure 32 A dummy processor which we have added to the service panel

Figure 32 shows a dummy processor which we have added to the service panel to be used as an abstract construct. This processor has been used in Figure 33 for creating a simple workflow template. This dummy processor can be replaced with any other processor inside the workflow.
In Figure 33-a we have created a workflow template. The dummy processor has been name “dummyProcessor1”. We have added a concrete processor named “realProcessor1” and it has been demonstrated that in Figure 33-b the concrete processor has been swapped with the dummy processor. We added a new item to the right click menu when a user has right clicked on the white area of an opened workflow in the Workbench. We named this item, “Replace with” in Figure 34. When a user has clicked on that item, she can select the two processors to be swapped with each other.

**Figure 33 Creating a workflow from template**

- **a. Workflow Template with dummy processor.**
- **b. Concrete workflow created by replacing real processor with dummy processor**
For swapping two processors, user selects one processor from the left list and one processor from the right then the second processor swaps with the first processor. Figure 35 has depicted the dialog where “dummyProcessor1” has been selected to be swapped with “realProcessor1”.

Figure 34 Replace with item added to right click menu.

Figure 35 The dialog to swap two workflow processors
Figure 33-a shows before a processor being swapped with the dummy processor, and Figure 33-b shows the same workflow after they are swapped. It is important to mention both processors are remained in workflow and removing the dummy processor from the workflow in our solution is the designer responsibility. However, the user experience can be improved by using drag and drop which is mentioned in the “future works” part of this thesis.

5.4 Development of Patterns

To prove the implementation possibility of the patterns which we have not implemented before or have not found in the repository, we have tried several approaches.

We have chosen a sample pattern from the catalogue of the patterns to implement. This pattern has not been found in our analysis over the myExperiment analysis. The reason for that might have been the complexity of implementation or lack of access to the appropriate tools.

We use four different ways of implementations for this pattern:

1. Subworkflow implementation of the pattern
2. Implementing the pattern with Shared Variable Manager plugin
3. Implementing the pattern as a workflow component
4. Implementing the pattern as a workflow templates

Subworkflow solution has been previously available but has not been observed in the repository. The three other solutions have been new and have been introduced in this dissertation.

The selected pattern is “Pattern 29 (Data Transfer - Copy In/Copy Out)” from [12] presented in Figure 39. It is used to load/save a variable from/to a repository.
5.4.1 Implementing the pattern using a subworkflow

In Taverna sub-workflows have been called “Nested workflows”. Through a nested workflow, a workflow designer could have re-used a workflow in another one. We have used this mechanism to read a file and have extracted the value of a variable. In addition, another nested workflow has been used to store the variable value in the file.

We have presented the subworkflow implementation of the “Copy In/Out” pattern in Figure 37. This workflow reads two “M” and “N” variables form the repository which in this case has been a file and update their values. The name of the variables and the path to the storage file along with their new values has been passed through the workflow inputs. The two sub-workflows on the right side of the figure are responsible for loading the variables and pass it to the workflow output. The sub-workflows on the left update the variables in the storage file. The variables new values are passed to the workflow outputs. The differences between a subworkflow implementation and component implementation are

1. Components are listed in the services panel but nested workflows are not.
2. Taverna workbench is not showing the details of workflows but it shows the details of the nested workflows.

This workflow has shown us the complexity of the loading and updating a variable using nested workflows. We have seen all the processors which might not have been important for the workflow users.

We have concluded our experience working with this method as below:

**Pros:**

- This method has been the only method which without installing a new plugin we could have implemented this pattern.
- The details of the implementation can be.

**Cons:**

- The implementation process is time consuming and complex.
Scientific Workflow Patterns

- Workflow designer has had to have an upfront idea for where the nested workflow is going to be used and changing is hard.
- There is no level of abstraction over the process of retrieving and storing data.
- It uses a manual process of loading and storing data.
- The workflow is not easily readable and understandable.

5.4.2 Implementing the pattern using the Shared Variable Manager

Shared Variable Manager has added two main features: the variable manager and two processors. Both of these features have been used for the implementation of this pattern. The load processor has been used for the “Copy In” Pattern to retrieve the variable’s value from the repository, and the save processor has been used for the “Copy out” pattern to store the variables new value to the variables repository.

![Diagram](image)

**Figure 38 Implementation of the “Copy In/Out” value with the variable manager plugin**

In this method of implementation we have used 2 load and 2 save services from the Shared Variable Manager services as in the Figure 38. Since we have used the variable manager plugin, we have not defined the path the variable storage file. However, we have defined the “M” and “N” variables in the variables repository and assigned the read and write permission to the workflow. **Figure 39** have used to represent the variables in the variables repository.
Our experience of using this method for this pattern has been summarised as below:

**Pros:**

- Easy and quick to load and store variables with the abstraction level provided by the plugin.
- Rapid implementation
- Readable workflow
- No need to define the storage file path

**Cons:**

- The need to install a new plugin on the Workbench.
- Not supporting lists and other types of data.
This method is used for all the patterns which we needed to implement a Scope, Case, Folder and Environment data where their implementation without using this method has not been possible or it has been very hard.

5.4.3 Implementing the pattern as a workflow components

When a workflow has been developed, through the component plugin it could have been converted to a workflow component. This workflow might have include any type of workflow processor such as a Web Service to call an external service to compute, a Beanshell or R script to run the developed script inside the workflow, mixture of local processors and a nested workflow.

However, when the workflow or fragment of workflow has been introduced as a component, the end user can only see the whole elements as one workflow activity. A workflow component has wrapped the complexity into itself and has provided a simple interface for the end user.

![Figure 40 Copy In/Out pattern implemented as only one workflow component](image)

We wanted only one workflow processor responsible for the “Copy In/Out” pattern. The final shape of the pattern implemented with this method has been shown in Figure 40. We have connected the variable names, values and the storage file path to the pattern component and we have connected the component outputs to the workflow outputs.
One of the major benefits of using this method has been the ability to reuse the pattern as a workflow service. In Figure 31 we have demonstrated the “Copy In/Out” service added to the service panel of the Taverna Workbench.

In order to create a component with only one processor we have taken three steps to convert the previously created workflow which has been implemented with subworkflow to a single processor component.

First we have converted all the nested workflows to the workflow components (Figure 41).

![Diagram](image)

**Figure 41 Copy In/Out Pattern Implemented with Components**

Secondly, wrapped the whole workflow into a new nested workflow as it has been depicted in Figure 42. In this step we have added two new string constants containing the names of the variables.
Finally, we converted the nested workflow to a workflow component (Figure 42). The final component has been easier for the workflow designers’ reuse, since, they have had to only define the variable names, file path and the new values. Everything else has been encapsulated in the component.

We have listed the pros and cons of this step as follows:

**Pros**

- It is easy to use a component.
- The complexity is wrapped and hidden away from the workflow designer or user.
- The ability to create a pallet of patterns in the service panel
- Dealing with only one component at the end.

**Cons**

- Lack of the functionality to define colors or icons for the components.
- User is not aware about what exactly is happening inside component.
5.4.4 Implementing the pattern as a workflow templates

The previous implementations of the patterns have been concrete, for example the “Copy In/Out” has been storing and retrieving data from a file. However, the template approach has had a big difference. Because the templates have an abstract constructs, the patterns which has been developed by template have not been executable themselves and they needed concrete constructs to be swapped with the dummy processors.

The point of using templates in the implementation of the patterns has been the fact that we have implemented an skeleton of the patterns which could have filled with any other processors.

For example, we could replace the processor responsible to loading data from file with another processor loading data from data base like a MySQL or NoSQL data store or a web service with the same functionality.

We have concluded our experience of using this method as follows:
Pros:

- Construction of a skeleton of patterns which can be filled with any processors with the same responsibility.
- Abstract way of implementing patterns.
- A method to create a family of patterns.

Cons:

- Templates are not executable
- It is not the artifact that a workflow end user is expecting and it only helps workflow designers.

The source code of the extensions developed for this project is available at GitHub [67].
Chapter 6 Evaluation of the approaches

In this project we have developed patterns by using the mixture of tools developed for Taverna Workbench. This part of the thesis aims to check the validity of results of each method. We have evaluated our methods in two ways:

1. A method to evaluate the pattern implementation using nested workflows, Shared Variable Manager and component methods.

2. A method to evaluate template method.

The pattern which we have used to evaluate both groups of methods has been the “Copy In/Out” pattern. We have described this pattern in the chapter 5.4 and Figure 36 has been used to show the pattern schema.

In our evaluation we have designed workflows based on the above methods and executed them. Since, the template method has not been executable we have not included that in this method of evaluation.

6.1 Evaluations of the first 3 methods

First, we have evaluated the nested workflow, shared variable and component methods. We have defined two variables named “M” and “N” into the storage repository. The repository for the nested workflow and component methods have been a file in the file system. Moreover, we have set the initial variables values from the repository feature of the Shared Variable Manager. The value defined for the variable M has been “3.14” and it has been “AB01” for the variable N. We have checked the load processors to return the right values in the outputs.

Second, we have updated the variables values and have checked to see the variables are updated. We have expected a change in the values of M to “6.28” and N to “CD23”.
All the three methods here have worked successfully and the final results of all the methods have been as expected values. However, there have been some differences between the execution time of the three methods.

The average workflow execution time has been reported by the Taverna Workbench in the progress report tab. Figure 44 shows the Taverna progress report for the execution of Copy In/Out pattern with nested workflow method.

We have executed each workflow 3 times and Table 2 shows the execution time of the workflows implemented with different methods.

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Execution time (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nested Workflow Method</td>
<td>10</td>
</tr>
<tr>
<td>Shared Variable Manager Method</td>
<td>1.02</td>
</tr>
<tr>
<td>Component Method</td>
<td>9.8</td>
</tr>
</tbody>
</table>

Table 2 Execution time of the Copy In/Out pattern with different implementations

We have seen that the minimum execution time has allocated to the Shared Variable Manager method with 1.02 seconds average time.
In addition, the results from the component method have shown us that the component method not only has no any overheads on the execution time but also the average execution time has been less than the nested workflow method. This reduction in the average execution time might have been related to the decline in the number of graphical elements in the workflow which might have reduced the computation.

6.2 Evaluation of the template method

The workflows created with the template method are executable. Therefore, we have designed another evaluation method for this method.

We have replaced the abstract processors with the concrete nested workflows which have been designed for the nested workflow methods and then we have executed the workflow.

We have checked two functionalities with this method of evaluation:

1. The processor swap functionality
2. The process of creating an executable concrete workflow from an abstract workflow.

We have placed a nested workflow to load a variable and another nested workflow to save variable. Then we have swapped them with the abstract processors, we have removed the abstract processors and we have executed the workflow.

We have done this for the variable “N” since our template has been designed only for one variable. However the template elements could have been duplicated.

The workflow has returned the variable value and has updated the variable value with the new value. The average execution time for the workflow generated from the template has been 6.86 seconds. This execution time has been only for one variable.
6.3 Comparison of the methods

We compare the methods from different aspects and present the results of this comparison in the Table 3. The first row of this table has listed the four methods and the first column of this table has listed the different properties.

<table>
<thead>
<tr>
<th></th>
<th>Subworkflow</th>
<th>Shared Variable Manager</th>
<th>Components</th>
<th>Templates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Need for plugin</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>View the details of</td>
<td>Yes</td>
<td>Yes/No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>implementation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level of Abstraction</td>
<td>Concrete</td>
<td>Concrete</td>
<td>Abstract</td>
<td>Abstract</td>
</tr>
<tr>
<td>Automatic loading and saving data</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Easy</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Understandable Output Workflow</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Need for definition of the file path</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Support for the lists</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Support to add to the service panel</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>One processor artifact</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Define Family of patterns</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
We have seen that each of these methods have had some benefits for the users. However, choosing between one of them has been related to the situation. This table can be used to choose between the methods in different situations.

### 6.4 Summary

We have conducted two different methods of evaluation for the validating our methods of pattern implementation in Taverna scientific workflow manager system. The evaluations have been based on the expectation on update of values when the workflows have been executed.

We have also measured the execution time in order to see which method has been the most efficient method.
Chapter 7: Conclusion and Future Works

This thesis has covered both background research and a new approach to investigate patterns in Taverna workflows. We have introduced the previous works which have focused on the scientific workflow patterns and discussed the types of workflows, the method they have been categorized, their characteristics and related tools and websites.

We have also designed a method to investigate patterns in scientific workflows which has included the following stages:

- Acquisition of the patterns
- Analysis of the patterns
- Implementation of patterns

In the first stage we have acquired the data which we have had to analyse. We have designed our own tools to gather information. The myExperiment repository has been used as a resource of Taverna workflows for us during this thesis.

In the second stage we have analysed the acquired information and have identified the places which the Taverna workflow management has needed to be extended.

In the third phase we have implemented the scientific workflow patterns. This part of our thesis has had two parts.

- First, we have extended the Taverna workflow to support the features which has stopped the workflow designer to use or design patterns.
- Second, we have implemented the patterns using the tools we have developed and extended. We have introduced four different methods of implementation.
In the final part we have evaluated the results of the implemented patterns to check all the methods have returned the same expected results. We have also compared the execution time of the workflows in this chapter.

We concluded that the lack of the abstraction mechanism in the Taverna Workflow Management System has stopped users from designing reusable workflows. The workflow templates and components mechanisms can help to improve the reusability of workflows.

In addition, a tool to define the shared variables can help users to design workflows using the workflow patterns which formalize the best practices in the workflow development.

Future Works

The results of this thesis have pointed to several interesting directions for future work:

1. **Using objects instead of strings for Shared Variable Manager**

The solution for the Shared Variable Manager that we have developed in this thesis has only accepted “Strings” as the type of variable values. This solution can be improved by enabling the acceptance of any type of objects for the value of a variable. If this solution improves, there might be a support for lists, tables, big data or Web Service call for the value of variables which might be very useful for the scientists.

2. **Template Designer**

In the solution that we have proposed, creation of the templates has been a naïve solution. A better solution might have been the development of a new perspective for the Taverna Workbench which has let the workflow designer to compose different elements of templates and create a template. Those elements might have been a list of abstract or dummy processors with different icon and colours grouped by their usage.
3. Better component replacement solution

Components have been swapped not in the most efficient way in our solution. A better way of replacement of components might have been the ability to drag one processor or nested workflow from the service panel or from the workflow and to drop it over the abstract processor.

Another way of replacement might have been the ability to right click on one abstract processor and select from the services in the right click menu to get replaced with the selected processor.

4. Ability to define colours and icons for the components

Components have been presented in one colour and one shape in this thesis. If the user could have the ability to define the colour or an icon for the component, it might have helped her later in finding them in services panel or in understanding the workflow easier.

For example instead of a long text an icon might have been much more meaningful to the end user.

5. Usage of Templates

Templates have been saved as a template in our solution. We could have saved them in other format and designed a more appropriate way of loading them. For example a wizard of creation of a workflow might have been created and when a new workflow has been created from the template, it could have presented different groups of available templates and have let the users to choose between them. Furthermore, the user could have selected the processors or sub-workflows which she has wanted to replace with the abstract processors.
6. User Evaluation

We evaluated our approach technically by checking if we can rewrite a workflow using our tools to apply the patterns in them. We did not ask any workflow designer to use our developed tools to see if they help them in workflow designs and we did not have time to publish our tools for the users and check the myExperiment repository if users are designing workflows using the developed tools and see the success of this approach in practice.

We recommend the user evaluation of our approach and measurement of the success both by asking the workflow designers and checking the myExperiment repository.

7. Refactoring feature in Taverna Workflow Management System

In software development, refactoring is the process of restructuring the existing body of code without changing its behaviour in order to improve the readability, reusability and architecture of the software. There is a list of known refactoring available and IDEs provide the refactoring functionality for the code. During this dissertation we observed that changing a workflow structure is hard and Taverna Workbench does not support any refactoring feature.

We recommend improvement of the Taverna Workbench features to support refactoring therefore, the workflow designers can change a workflow and improve its designs.
Appendix

7.1 SPARQL Queries

7.1.1 Workflow versions with their URL

BASE <http://www.myexperiment.org/>

PREFIX meannot: <http://rdf.myexperiment.org/ontologies/annotations/>

PREFIX mebase: <http://rdf.myexperiment.org/ontologies/base/>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX mecontrib: <http://rdf.myexperiment.org/ontologies/contributions/>

SELECT ?wf ?p

WHERE {


?wf meannot:citation-url ?p

}

7.1.2 Workflows added after September 2009

PREFIX mebase: <http://rdf.myexperiment.org/ontologies/base/>

PREFIX mecontrib: <http://rdf.myexperiment.org/ontologies/contributions/>/

PREFIX dcterms: <http://purl.org/dc/terms/>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?wfv ?added

WHERE {

}
7.1.3 List of all data-flows

PREFIX mecomp: <http://rdf.myexperiment.org/ontologies/components/>

SELECT ?d
WHERE{
?d a mecomp:Dataflow
}

7.1.4 List of dataflow processors

PREFIX mecomp: <http://rdf.myexperiment.org/ontologies/components/>

SELECT ?d
WHERE{
?d a mecomp:DataflowProcessor
}

7.1.5 Latest versions of workflows

PREFIX mebase: <http://rdf.myexperiment.org/ontologies/base/>

PREFIX mecontrib: <http://rdf.myexperiment.org/ontologies/contributions/>

PREFIX mecomp: <http://rdf.myexperiment.org/ontologies/components/>

SELECT ?wfv
WHERE{
?wf a mecontrib:Workflow.
?wfv mebase:has-current-version ?wfv;
dcters:created ?added
FILTER ( ?added >= xsd:dateTime('2009-09-01T00:00:00Z') )
}
7.1.6 List of current versions workflows with their data flows

PREFIX mebase: <http://rdf.myexperiment.org/ontologies/base/>
PREFIX mecontrib: <http://rdf.myexperiment.org/ontologies/contributions/>
PREFIX mecomp: <http://rdf.myexperiment.org/ontologies/components/>

SELECT ?wfv ?d
WHERE{
  ?w a mecontrib:Workflow.
  ?w mebase:has-current-version ?wfv.
  ?wfv mecomp:executes-dataflow ?d.
}

7.1.7 List of all processors with their types

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX mebase: <http://rdf.myexperiment.org/ontologies/base/>
PREFIX mecontrib: <http://rdf.myexperiment.org/ontologies/contributions/>
PREFIX mecomp: <http://rdf.myexperiment.org/ontologies/components/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT ?uri ?objType
WHERE {
  ?subClass rdfs:subClassOf mecomp:Processor .
  ?uri rdf:type ?subClass .
  ?uri rdf:type ?objType .
}
7.1.8 List of WSDL processors

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX mebase: <http://rdf.myexperiment.org/ontologies/base/>

PREFIX mecontrib: <http://rdf.myexperiment.org/ontologies/contributions/>

PREFIX mecomp: <http://rdf.myexperiment.org/ontologies/components/>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?p

WHERE{
  ?p a mecomp:WSDLProcessor.
}

7.1.9 List of subworkflows

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX mebase: <http://rdf.myexperiment.org/ontologies/base/>

PREFIX mecontrib: <http://rdf.myexperiment.org/ontologies/contributions/>

PREFIX mecomp: <http://rdf.myexperiment.org/ontologies/components/>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT ?p

WHERE{
  ?p a mecomp:DataflowProcessor.
}

7.1.10 Service information of a Dataflow

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX mebase: <http://rdf.myexperiment.org/ontologies/base/>

PREFIX mecontrib: <http://rdf.myexperiment.org/ontologies/contributions/>

PREFIX mecomp: <http://rdf.myexperiment.org/ontologies/components/>
7.1.11 Dataflow processors and theirs types

BASE <http://www.myexperiment.org/>

PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX mebase: <http://rdf.myexperiment.org/ontologies/base/>
PREFIX mecontrib: <http://rdf.myexperiment.org/ontologies/contributions/>
PREFIX mecomp: <http://rdf.myexperiment.org/ontologies/components/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?d ?c ?t

WHERE{
  ?d rdf:type mecomp:Dataflow.
  ?d mecomp:has-component ?c.
}
7.1.12 Processors of a defined workflow

BASE <http://www.myexperiment.org/>

PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX mebase: <http://rdf.myexperiment.org/ontologies/base/>
PREFIX mecontrib: <http://rdf.myexperiment.org/ontologies/contributions/>
PREFIX mecomp: <http://rdf.myexperiment.org/ontologies/components/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?c ?t

WHERE{
  ?c a ?t.
  ?p rdf:type mecomp:Processor
  FILTER (?t=?pt)
}

7.1.13 List of a component inputs

BASE <http://www.myexperiment.org/>

PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX mebase: <http://rdf.myexperiment.org/ontologies/base/>

PREFIX mecontrib: <http://rdf.myexperiment.org/ontologies/contributions/>

PREFIX mecomp: <http://rdf.myexperiment.org/ontologies/components/>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?d ?c ?t

WHERE{
  ?d rdf:type mecomp:Dataflow.
  ?d mecomp:has-component ?c.
  {?c rdf:type mecomp:ConstantProcessor} UNION
  {?c rdf:type mecomp:OtherProcessor} UNION
  {?c rdf:type mecomp:WSDLProcessor} UNION
  {?c rdf:type mecomp:BeanshellProcessor}
}

7.1.14 List of a defined workflow’s data links

BASE <http://www.myexperiment.org/>

PREFIX dcterms: <http://purl.org/dc/terms/>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX mebase: <http://rdf.myexperiment.org/ontologies/base/>

PREFIX mecontrib: <http://rdf.myexperiment.org/ontologies/contributions/>

PREFIX mecomp: <http://rdf.myexperiment.org/ontologies/components/>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT ?c ?o ?title
WHERE{

<http://www.myexperiment.org/workflows/16/versions/7#dataflows/1> mecomp:has-component ?c.

?c mecomp:to-input ?o.

?c rdf:type mecomp:Link.


?input a mecomp:Input.

}

7.1.15 Processors and their type with links for a defined workflow

BASE <http://www.myexperiment.org/>

PREFIX dcterms: <http://purl.org/dc/terms/>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX mebase: <http://rdf.myexperiment.org/ontologies/base/>

PREFIX mecontrib: <http://rdf.myexperiment.org/ontologies/contributions/>

PREFIX mecomp: <http://rdf.myexperiment.org/ontologies/components/>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>


WHERE{

<http://www.myexperiment.org/workflows/16/versions/7#dataflows/1> mecomp:has-component ?link.

?link rdf:type mecomp:Link.

?link mecomp:from-output ?from.


7.1.16 Processors and their type with links for a defined workflow with the WSDL info

BASE <http://www.myexperiment.org/>

PREFIX dcterms: <http://purl.org/dc/terms/>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX mebase: <http://rdf.myexperiment.org/ontologies/base/>

PREFIX mecontrib: <http://rdf.myexperiment.org/ontologies/contributions/>

PREFIX mecomp: <http://rdf.myexperiment.org/ontologies/components/>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>


WHERE{

<http://www.myexperiment.org/workflows/16/versions/7#dataflows/1> mecomp:has-component ?link.

?link rdf:type mecomp:Link.

?link mecomp:from-output ?from.


OPTIONAL {?fromc rdf:type mecomp:WSDLProcessor;}
Scientific Workflow Patterns

mecomp:processor-uri ?fromc_uri;

mecomp:service-name ?fromc_sname.

}

?link mecomp:to-input ?to.

?to mecomp:for-component ?toc.


?input a mecomp:Input.

}

7.2 Example myExperiment RDF Data

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE rdf:RDF >
<!ENTITY mebase 'http://rdf.myexperiment.org/ontologies/base/'>
<!ENTITY meac 'http://rdf.myexperiment.org/ontologies/attrib_credit/'>
<!ENTITY meannot 'http://rdf.myexperiment.org/ontologies/annotations/'>
<!ENTITY mepack 'http://rdf.myexperiment.org/ontologies/packs/'>
<!ENTITY meexp 'http://rdf.myexperiment.org/ontologies/experiments/'>
<!ENTITY mecontrib 'http://rdf.myexperiment.org/ontologies/contributions/'>
<!ENTITY mevd 'http://rdf.myexperiment.org/ontologies/viewings_downloads/'>
<!ENTITY mecomp 'http://rdf.myexperiment.org/ontologies/components/'>
<!ENTITY mespec 'http://rdf.myexperiment.org/ontologies/specific/'>
<!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
<!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
<!ENTITY owl 'http://www.w3.org/2002/07/owl#'>
<!ENTITY dc 'http://purl.org/dc/elements/1.1/'>
<!ENTITY dcterms 'http://purl.org/dc/terms/'>
<!ENTITY cc 'http://web.resource.org/cc/'>
Scientific Workflow Patterns

<!ENTITY foaf 'http://xmlns.com/foaf/0.1'/>
<!ENTITY sioc 'http://rdfs.org/sioc/ns#'>
<!ENTITY skos 'http://www.w3.org/2004/02/skos/core#'>
<!ENTITY ore 'http://www.openarchives.org/ore/terms'/>
<!ENTITY dbpedia 'http://dbpedia.org/ontology'/>
<!ENTITY snarm 'http://rdf.myexperiment.org/ontologies/snarm'/>
<!ENTITY xsd 'http://www.w3.org/2001/XMLSchema#'>
>
<rdf:RDF xmlns:mebase ="&mebase;"
   xmlns:meac ="&meac;"
   xmlns:meannot ="&meannot;"
   xmlns:mepack ="&mepack;"
   xmlns:meexp ="&meexp;"
   xmlns:mecontrib ="&mecontrib;"
   xmlns:mevd ="&mevd;"
   xmlns:mecomp ="&mecomp;"
   xmlns:mespec ="&mespec;"
   xmlns:rdf ="&rdf;"
   xmlns:rdfs ="&rdfs;"
   xmlns:owl ="&owl;"
   xmlns:dc ="&dc;"
   xmlns:dcterms ="&dcterms;"
   xmlns:cc ="&cc;"
   xmlns:foaf ="&foaf;"
   xmlns:sioc ="&sioc;"
   xmlns:skos ="&skos;"
   xmlns:ore ="&ore;"
   xmlns:dbpedia ="&dbpedia;"
   xmlns:snarm ="&snarm;"
   xmlns:xsd ="&xsd;"
>
<rdf:Description rdf:about="http://www.myexperiment.org/workflows/59.rdf">
   <foaf:primaryTopic rdf:resource="http://www.myexperiment.org/workflows/59"/>
</rdf:Description>

<main>

</main>
The latest (and greatest) version of the QR code generator workflow. This workflow uses the QR code service provided by the ChemTools project.

This workflow was created on 2007-10-07T17:03Z and modified on 2008-07-16T01:46:13Z.

The workflow can be downloaded at http://www.myexperiment.org/workflows/59/download/qr_code__matrix_code__generator_23815.xml

The workflow can be viewed and downloaded 5511 times.

The workflow is licensed under the terms of the Creative Commons license.

The workflow is executed as part of the dataflow http://www.myexperiment.org/workflows/59#dataflow

The workflow has been rated 97 times.

The workflow has been tagged with the following tags:
- taggings/170
- taggings/171
- taggings/172
- taggings/1676
Taverna 1 Workbench is an earlier version of the Workbench which is still available for download. Taverna 1 workflow descriptions are produced and consumed by this version of the Workbench. They are in an XML format also known as SCUFL. The Taverna 1 Workbench is an earlier version of the Workbench which is still available for download. Taverna 1 workflow descriptions are produced and consumed by this version of the Workbench. They are in an XML format also known as SCUFL. Note that the latest version of the Workbench is Taverna 2 and is the recommended version. Most Taverna 1 workflows can also be read using the Taverna 2 Workbench. Users of the Taverna 1 Workbench can access myExperiment from the Workbench using a plugin which can be downloaded within the Workbench. The plugin allows users to browse, download and open workflows from myExperiment within Taverna.
This workflow creates QR code (matrix code) generator. It takes a URL and description as inputs and generates a QR code bookmark using the supplied URL and description. The QR code is then displayed on the user's device when scanned.
Creative Commons Attribution-Share Alike 3.0 Unported License

You are free:

- to Share — to copy, distribute and transmit the work
- to Remix — to adapt the work

Under the following conditions:

- Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- Share Alike — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

With the understanding that:

- Waiver — Any of the above conditions can be waived if you get permission from the copyright holder.
- Other Rights — In no way are any of the following rights affected by the license: your fair dealing or fair use rights, the author's moral rights, rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.
- Notice — For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to http://creativecommons.org/licenses/by-sa/3.0/.
Scientific Workflow Patterns

<cc:permits rdf:resource="http://creativecommons.org/ns#DerivativeWorks"/>
<cc:requires rdf:resource="http://creativecommons.org/ns#ShareAlike"/>
<dcterms:created rdf:datatype="&xsd;dateTime">2009-07-08T14:09:36Z</dcterms:created>
<dcterms:modified rdf:datatype="&xsd;dateTime">2009-07-08T14:09:37Z</dcterms:modified>
</mebase:License>

<dcterms:hasFormat rdf:resource="http://www.myexperiment.org/workflows/59/policies/108.rdf"/>
</snarm:Policy>

<mecomp:Dataflow rdf:about="http://www.myexperiment.org/workflows/59/versions/4#dataflow">
</mecomp:Dataflow>

<mecomp:Source rdf:about="http://www.myexperiment.org/workflows/59/versions/4#dataflow/components/1">
<dcterms:title rdf:datatype="&xsd;string">url</dcterms:title>
<dcterms:description rdf:datatype="&xsd;string">Uniform Resource Locator (URL) to be encoded as a QR code.</dcterms:description>
</mecomp:Source>
Scientific Workflow Patterns

<meannot:Rating rdf:about="http://www.myexperiment.org/workflows/59/ratings/97"
               <dcterms:hasFormat rdf:resource="http://www.myexperiment.org/workflows/59/ratings/97.rdf"/>
               <dcterms:hasFormat rdf:resource="http://www.myexperiment.org/workflows/59/ratings/97.xml"/>
               <mebase:annotates rdf:resource="http://www.myexperiment.org/workflows/59"/>
               <meannot:rating-score rdf:datatype="&xsd;positiveInteger">1</meannot:rating-score>
               <mebase:has-annotator rdf:resource="http://www.myexperiment.org/users/258"/>
               <dcterms:created rdf:datatype="&xsd;dateTime">2008-01-28T23:53:50Z</dcterms:created>
               </meannot:Rating>

<meannot:Rating rdf:about="http://www.myexperiment.org/workflows/59/ratings/349"
               <dcterms:hasFormat rdf:resource="http://www.myexperiment.org/workflows/59/ratings/349.rdf"/>
               <dcterms:hasFormat rdf:resource="http://www.myexperiment.org/workflows/59/ratings/349.xml"/>
               <mebase:annotates rdf:resource="http://www.myexperiment.org/workflows/59"/>
               <meannot:rating-score rdf:datatype="&xsd;positiveInteger">2</meannot:rating-score>
               <mebase:has-annotator rdf:resource="http://www.myexperiment.org/users/58381"/>
               <dcterms:created rdf:datatype="&xsd;dateTime">2012-08-15T10:40:51Z</dcterms:created>
               </meannot:Rating>

<meannot:Tagging rdf:about="http://www.myexperiment.org/tags/536/taggings/170"
               <dcterms:hasFormat rdf:resource="http://www.myexperiment.org/tags/536/taggings/170.rdf"/>
               <dcterms:hasFormat rdf:resource="http://www.myexperiment.org/tags/536/taggings/170.xml"/>
               <meannot:uses-tag rdf:resource="http://www.myexperiment.org/tags/536"/>
Scientific Workflow Patterns
References

60. Williams, A., *myExperiment Workflow: Pipelined list iteration.*
61. Hulme, H., *myExperiment Workflow: Example of Execute_SQL_Query with parameter.*
64. *Taverna Service Provided Interfaces.*